The Smart Home meets the Web of Things

Article *in* International Journal of Ad Hoc and Ubiquitous Computing · May 2011

DOI: 10.1504/IJAHUC.2011.040115 · Source: DBLP

3 authors, including:

Andreas Pitsillides
University of Cyprus
302 PUBLICATIONS **3,049** CITATIONS

SEE PROFILE

Vlad Trifa
EVRYTHNG
49 PUBLICATIONS **2,812** CITATIONS

SEE PROFILE

The Smart Home meets the Web of Things

Andreas Kamilaris*

Department of Computer Science,
Networks Research Laboratory,
University of Cyprus,
P.O. Box 20537, Nicosia, CY 1678, Cyprus
E-mail: kami@cs.ucy.ac.cy
*Corresponding author

Vlad Trifa

Department of Computer Science,
Distributed Systems Group,
Institute for Pervasive Computing,
ETH Zurich (Swiss Federal Institute of Technology Zurich),
Universitatstrasse 6, H 103.1, CH-8092 Zurich, Switzerland
E-mail: trifa@inf.ethz.ch

Andreas Pitsillides

Department of Computer Science,
Networks Research Laboratory,
University of Cyprus,
P.O. Box 20537, Nicosia, CY 1678, Cyprus
E-mail: andreas.pitsillides@ucy.ac.cy

Abstract: In recent years, the merging of computing with physical things, enabled the transformation of everyday objects into information appliances. We propose to reuse the central principles of the modern Web architecture to fully integrate physical objects to the Web and build an interoperable Smart Home. We present an application framework that offers support for multiple home residents concurrently. We show that by using the Web as application layer we can build flexible applications on top of heterogeneous embedded devices with only a few lines of code, transforming home automation into a trivial task. We address many issues related to Web-enabling these devices, from their discovery and service description to the uniform interaction with them. Our evaluation efforts indicate that our framework offers acceptable performance and reliability.

Keywords: Embedded Devices; Information Appliances; Smart Home; Ubiquitous Computing; Web of Things; Application Framework; Web Services; REST; Web Mashups.

Biographical notes: Andreas Kamilaris received the BA degree in Computer Science from University of Cyprus in 2007 and the MS degree in Distributed Systems from the ETH University of Zurich, Switzerland in 2009. Currently, he is a doctoral candidate at the Computer Science department of the University of Cyprus. He is a member of the Networks Research Laboratory (NetRL), which is part of the Computer Science department. His research interests include ad-hoc, wireless sensor networks, distributed systems, ubiquitous computing and Web technologies.

Vlad Trifa is a PhD student at ETH University of Zurich, Switzerland at the Institute for Pervasive Computing, and a Research Associate at SAP Research in Zurich. He graduated in 2006 with a MS degree in Computer Science at the Ecole Polytechnique Federale de Lausanne (EPFL). His research interests include lightweight middlewares for embedded devices, Web technologies and how to use them for networked devices and semantic technologies.

Andreas Pitsillides is a Professor of Computer Science, University of Cyprus (UCY) and heads the Networks Research Lab (NetRL) at UCY. His research interests include fixed and mobile/wireless networks, Internet technologies and their application in Mobile e-Services, especially e-health, and security. He serves on the editorial board of the Journal of Computer Networks (COMNET), served on executive committees of international conferences (e.g. INFOCOM, WiOpt, ISYC, MCCS, and ICT) and many technical committees. Andreas is also a member of the International Federation of Automatic Control (IFAC) Technical Committee (TC 1.5) on Networked Systems, IFAC TC 7.4 on Transportation Systems and the IFIP working group WG 6.3.

1 Introduction

Physical things, such as household appliances, have been becoming smarter and smarter. They are equipped with embedded microprocessors and they offer some limited, wireless communication abilities. Everyday objects are settled with small, cheap, mobile processors, sensors and embedded radio modules.

This merging of computing with physical objects introduces the concept of information appliances (Bergman, 2000), defined as devices or machines, designed to perform some specific functionality but are usable, at the same time, for the purposes of computing. Typical examples are smartphones, embedded devices, RFID chips and smart cards. As Donald A. Norman (Norman, 1999) points out, the trend in computing is towards simplicity through specialization. This trend seems to justify Mark Weiser's vision of the *Disappearing Computer* (Weiser, 1999), according to which *"the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it"*.

In the near future, homes will offer new automation possibilities to their residents. Changes will happen to the way people live and interact with their home environments, when technology recedes into the background of their lives, when information processing is thoroughly integrated into everyday objects and activities. Coffee machines began to appear that are able to prepare coffee automatically, according to user preferences. Fridges that offer dedicated APIs for their control are being produced. The fact that these appliances can be administered through the Web, brings the vision of a Web-enabled *Smart Home* into reality.

Let us consider a real-life scenario in which a family with many members interacts concurrently with the objects of its house through the Web. The father observes the consumption of the electrical appliances from his work while the mother, remotely, adjusts the temperature in the rooms. The children are able to turn the lights off, which they forgot on when they left hastily the building to go to school. In case of theft or fire, all family members are immediately informed and can take appropriate actions.

In order to materialize this vision, many issues must be resolved. The heterogeneous embedded devices that represent physical things need to be discovered, their capabilities have to be understood and uniform interaction possibilities must be developed. More powerful computing devices, acting as mediators, can be employed to facilitate administration and control of these ubiquitous environments and to create complex functionality on top of them. Failures that easily appear in pervasive spaces due to their ad hoc, unpredictable and highly mobile nature should be masked away.

In this paper we propose to use the Web as a standard, to realize the notion of the Smart Home. Based on the success of the Web 2.0, our proposal is about reusing well-accepted and understood Web principles to interconnect the quickly expanding ecosystem of embedded devices, built into everyday smart things. It is about taking the Web as we know it and extending it so that anyone can plug devices into it. This idea is commonly reffered to as the *Web of Things* (Wilde, 2007).

Until today, most of the approaches that have been used do not involve open systems and have been developed to provide solutions to some application-specific scenarios (medical, care for the elderly) while focus was more on the environment's state than on the devices themselves and the specific tasks they could perform. Very few projects have been inspired by the Web's scalable operation, which seems to be the key to address heterogeneity, and almost none of them grants the flexibility to the home resident, in order to fully adjust his house to his own needs.

The contribution of this paper is twofold. On one hand it presents an application framework with concurrent, multi-user support that facilitates the development of advanced ubiquitous applications by the habitant, who may probably have no programming experience to automate his house and on the other it proposes the use of Web-based standards to achieve interoperability at the application layer.

The rest of the paper is organized as follows: Section 2 defines the requirements of our approach. In Section 3, solutions to the challenges our framework must resolve are proposed. Section 4 describes the implementation of the system while Section 5 deals with a performance evaluation of the whole infrastructure. In Section 6 related work is presented and finally, Sections 7 and 8 discuss future work and conclude the paper.

2 Requirements for future Smart Homes

We believe that future generations of embedded devices will be much more ubiquitous, highly integrated in our everyday lives, something that would create the need for a greater degree of flexibility in order to be manageable. In particular, future Smart Homes shall be open and accessible for simultaneous users (family members), who could pull easily the data they need directly through the Web and use the data right away in their own applications. We envision three distinct interaction possibilities with information appliances that cover the most important use cases that should be supported by our framework:

- Ad-hoc interaction. Users directly access sensors and actuators (in a request-response model).
- Continuous monitoring. Devices stream data at regular intervals (eg. Smart Meters that measure energy consumption of electrical appliances).
- Event-based systems. Events are sent sporadically when something important happens (eg. indication of fire).

Additionally, we propose a list of requirements that most of the existing application frameworks for embedded devices lack, and will be needed in large-scale, open, and heterogeneous pervasive environments:

- Data is easily exported into Web applications in standard formats.
- Interoperable programming interface that provides the primitives to end users with very little programming experience to perform some advanced tasks.
- Concurrent, multi-user support through the Web.
- Uniform access to heterogeneous embedded devices. The ubiquitous space becomes a cloud where any device can be individually accessed in a standardized way.
- Particularities of resource-constrained environments get abstracted, offering limited reliability (not always reachable devices, variable QOS, device mobility, device failure).
- Acceptable Performance.

3 Web-oriented Application Framework

Based on existing solutions for interconnecting physical things (Section 6), along with the requirements for future Smart Homes (Section 2), we suggest to use the Web as application layer, because it is ubiquitous and scales particularly well. Therefore, we leverage the existing Web to integrate *by design* everyday objects.

In this section, we propose a Web-oriented application framework for embedded devices, which enables these devices to speak the same language as any other resource on the Web. First, we describe the core principles of the modern Web architecture and discuss our motivation to apply them in the field of Smart Homes and more generally, in ubiquitous environments. Then, we show how we have reused these concepts to build a lightweight infrastructure for Web-based interaction with embedded devices.

3.1 REST and Resource Oriented Architectures

Web Services have been extensively used during the last few years to provide integration across heterogeneous, distributed systems. Web Services tend to fall into one of two camps: Big Web Services and REpresentational State Transfer (RESTful) Web Services.

Big Web Services or WS-* (Alonso et al., 2004) are a set of complex standards and specifications for enterprise application integration. They form the foundational elements for the building of Service-oriented Architectures (SOA), which introduce new possibilities for large-scale software design and software engineering.

More recently, RESTful Web Services (Pautasso et al., 2008) have been gaining popularity. The notion of REST has been conceptualized in Roy Fielding's PhD thesis (Fielding, 2000). Rather than being a technology or standard, REST is an architectural style which basically explains how to use HTTP as an application protocol. REST advocates in providing services directly based on the HTTP protocol itself. An Application Programming Interface (API) fulfilling the REST architectural style is said to be RESTful.

Beyond the basic design criteria provided in Roy Fielding's thesis, the REST community has been working on refining the notions to create Resource Oriented Architectures (ROA), in order to provide and connect together services on the Web. Because of the underlying simplicity of this architecture, we believe that they could be adapted in order to interconnect the physical world and, in particular, embedded devices. Basically a Resource Oriented Architecture is about four concepts (Richardson and Ruby, 2007):

1. **Resources**. A resource in a ROA is anything important enough to be referenced and linked by other resources. It can be seen as an entity that provides some service.
2. **Their Names**. A resource has to be addressable, i.e. it needs to have a URI (Uniform Resource Identifier) that uniquely identifies it. This concept is known as the requirement for addressability of RESTful APIs.
3. **The links between them**. Thanks to the hyperlinking structure of the Web, resources can be related and connected to one another. This is the requirement for connectedness of RESTful APIs.
4. **Their Representations**. Resources can be represented using various formats. The most common one for resources on the Web is (X)HTML, due to its intrinsic support for browsing and human readability. However, when machine readability is required, XML and JSON are often chosen. Note that JSON is a data interchange format which stands as a lightweight alternative to the sometimes too verbose XML.

REST proposes the use of a uniform interface. Resources can only be manipulated by the methods specified in the HTTP standard. Amongst these verbs, four are most commonly used:

- **GET** is used to retrieve a representation of a resource. Concretely, sending a GET request along with the URI *http://.../sensors/officeLamp/Illumination* would return the light level currently observed by *officeLamp*.
- **POST** is a method to alter the state of a resource. As an example, it could be used to change the

state of an actuator. Sending a POST request to *http://.../actuators/bedroomLamp/Light* with the parameter *state=on*, causes the effect of *bedroomLamp* to be turned on.

- **PUT** represents an insert or update of a resource.
- **DELETE** is used to delete resources.

Comparing the two trends in Web Services, we believe that RESTful Web Services are more appropriate for resource-constrained, ad hoc environments. Thanks to their simplicity, the use of a uniform interface and the wide availability of HTTP libraries and clients, RESTful services are truly loosely coupled. This means that services based on RESTful APIs can be reused and recombined in a quite straightforward manner. They are adopted by the authors to fulfill the requirement of an interoperable and uniform interface of the application framework. In this paper, we name embedded devices and the services they offer as resources.

3.2 Ad Hoc Device Discovery

Device discovery is a crucial issue in distributed, ad hoc networks. A dynamic, flexible discovery process is needed to find new embedded devices and register their basic information, since devices are highly mobile, representing physical things that move inside the Smart Home. There exist many standardized approaches for device discovery such as Bonjour, UPnP and WS-discovery. However, REST and HTTP do not have currently any mechanism for device discovery, as discovery in REST is done by following links. Because of that, we propose in this section a simple process for discovering RESTful devices and integrating them into the Web. Fig. 1 shows the general message interaction pattern, followed at the device discovery procedure.

As soon as an embedded device is powered on, it will attempt to connect to our framework (which will be hosted at some computing machine) by broadcasting periodically a HELLO message. As soon as this message is received by the framework, it is immediately acknowledged (*Network Discovery Phase*). At that time, the embedded device responds with a single message that contains some device description information and a URL. The URL points to a Web page, where description information of the resources that are offered by that particular device is located. Transmission of a single URL for service description not only accelerates performance but also saves energy of the embedded devices, which do not need to follow a time- and energy-consuming procedure of transmitting consecutive messages to describe in detail all the resources they possess. When the framework receives this message, it checks the validity of the URL, parses its contents and adds the newly-found embedded device in its list of devices. If the parsing procedure is successful, a second acknowledgment message is sent to the device (*Device Description Phase*). From now on, the embedded device operates normally, coupled with the framework, waiting to receive requests from it (*Device Operation Phase*).

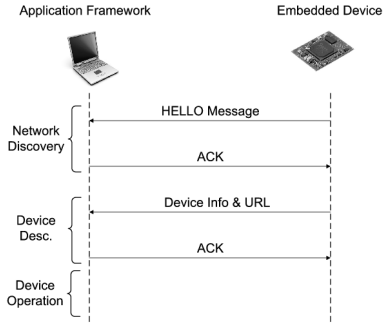

Figure 1 Device Discovery Procedure.

3.3 Resource Description

As we mentioned in the previous subsection, an embedded device needs to transmit a URL, in which the services offered by it are described. We searched for a standardized way to perform resource description, in order to achieve high interoperability with heterogeneous embedded devices and services. We concluded to adopt Web Application Description Language (WADL, 2005). WADL is an XML-based file format that provides a machine-readable description of HTTP-based Web applications. It can be characterized as the unofficial description language of RESTful Web Services. It is the equivalent of Web Services Description Language (WSDL, 2001), which is the official description language of Big Web Services.

WADL is intended for applications that are based on the existing architecture of the Web. Like WSDL, it is platform and language independent, and aims to promote reuse of applications beyond the basic use, inside a Web browser. WADL models the resources provided by an embedded device and the relationships between them.

4 Implementation

Here we describe how we implemented our application framework for future Smart Homes. Due to the fact that most of the embedded devices that are used today to represent physical things such as household appliances do not offer TCP/IP networking by default but dedicated communication protocols and technologies, we decided to develop our framework in order to behave like a gateway, from the Web to the pervasive environment and vice-versa. We also implemented some RESTful Web Services on REST-enabled embedded devices to show the feasibility of our approach. We present in detail our implementation efforts in the following subsections

and we depict how simple application development and home automation can be, by utilizing our framework.

4.1 RESTful Application Framework

As illustrated in Fig. 2, our application framework follows a modular architecture and is composed of three principal layers. *Device Layer*, which is responsible for the management and control of embedded devices, *Control Layer*, which is the central processing unit of the system and *Presentation Layer*, which generates dynamically a representation of the available devices and their corresponding services to the Web, enabling the uniform interaction with them over a RESTful interface. It is implemented in Java because of the versatility and portability of the language which allows the framework to run on virtually any device that has a Java Virtual Machine (JVM).

Driver module holds the technology-specific drivers that are used to enable the interaction with embedded devices. It has been designed with flexibility and simplicity in mind, as we would like to encourage programmers to write their own drivers for new device types that appear in the market. Each time a new device is discovered through the discovery procedure described in subsection 3.2, a new thread dedicated to the device is created. From now on, this thread is responsible for this particular embedded device and it runs until the physical loss of it, in its own execution environment. Each thread keeps track of the static and dynamic properties of the device it represents, such as its health status (through aliveness checks) and a list of the resources it offers, inside a *Resource Registry*. Thread programming simplifies the management of embedded devices and keeps the implementation logic as simple as possible.

In order to support multi-user support we attach a *Request Queue* to each device, to enqueue concurrent requests to it. Requests are stored in a FIFO manner and are transmitted sequentially to the device. The time a request will stay in the queue depends on many factors, such as the device type, its throughput, processing power and network load.

Upon reception of a response from the physical device through the *Driver* module, this response is forwarded to the appropriate thread, in order to be further forwarded to the Web client, who created the respective request. If a response doesn't arrive in a predefined time interval, it is retransmitted to the device for a few iterations, in order to mask away possible losses of messages in the unpredictable wireless medium. If after all the attempts no response is received, then the embedded device is considered unavailable and it is removed from the system.

Devices module is the "official representative" of the embedded devices to the upper layers of the application framework. It holds a list of all the devices that are available, inside a *Device Registry*.

Control Layer holds the *Core* module, which runs in the background and maintains system's threads, initializes all the other modules and checks that everything operates according to its specifications.

Presentation Layer represents the access point to the framework from the Web. A *Web Server* allows Web clients to interact with their smart environment using any Web browser. Thanks to the links between the resources, Web clients can easily explore available devices and their corresponding services. They can find the desirable service by clicking links, as they would browse Web pages. They are capable of interacting with any resource through a RESTful API, which is provided by a *REST Engine*. (Restlet, 2007) was used to implement the REST Engine, as it has very stable performance in Java environments.

4.2 RESTful Embedded Devices

We decided to select sensor motes to represent embedded devices in our implementation efforts, since they are very easy to program and they offer some basic sensing capabilities. We programmed them to follow the discovery pattern we illustrated in subsection 3.2. For that reason, we uploaded on a Web page a WADL file that describes their resources. The URL of this Web page is transmitted to the application framework during the discovery process.

We implemented a small number of RESTful Web Services in TinyOS (Levis et al., 2005) and we uploaded the software on Tmote Sky sensor motes. These sensor motes are equipped with a 250kbps, 2.4GHz, IEEE802.15.4-compliant Chipcon CC2420 Radio, integrated onboard antenna and a 8MHz TI MSP430 microcontroller with 10 kB RAM. 802.15.4 is an IEEE standard that defines a MAC and PHY layer targeted to Wireless Sensor Networks (WSN). We made use of Active Messages for our networking needs, which form the native radio communication model of TinyOS. In Table 1, we can see a general description of the RESTful Web Services we created. The first three resources sense the environmental state, the fourth actuates some LED lamps while the last resource involves streaming of the energy consumption of some electrical appliance. All the resources have real effects except the last one which is virtual.

Once the application framework receives a RESTful HTTP request from a Web client, it makes the necessary validity checks and encapsulates the request as plain ASCII, in the data field of an Active Message. It then sends it through a serial cable to a sensor mote, which is acting as a base station and is directly connected to the framework. The base station forwards the message over the radio to remote sensor motes by broadcasting. The message will be accepted by those motes, whose ID matches the one specified in the request.

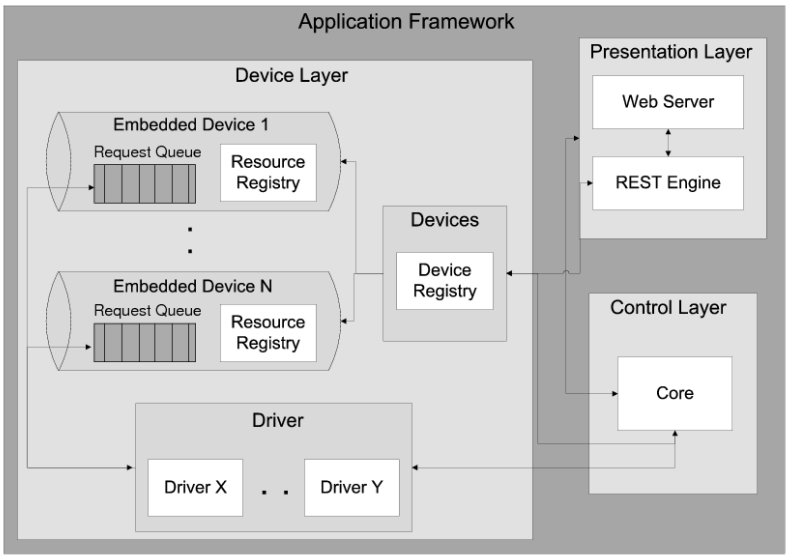

Figure 2 Application Framework Architecture.

Table 1 A list of the RESTful Web Services offered by sensor motes.

Index	*Resource URI*	*REST Verb*	*MIME (Return) Type*	*Parameter (Type)*
1	Temperature	GET	text/plain	-
2	Humidity	GET	text/plain	-
3	Illumination	GET	text/plain	-
4	Light	POST	text/plain	color (Character)
5	Electricity/Streaming	POST	text/plain	interval (Integer) & iterations (Integer)

4.3 Web Mashups

The uniform, RESTful interface facilitates home automation through the development of smart applications, from people with very little programming experience. For example, with a few lines of code a database could be created and used locally, or mashups of data can be implemented that offer an abstracted view of the pervasive space's functionality, as a composition of functionalities of individual embedded devices.

Mashups, in general, are Web-based resources that include content and application functionality through reuse and composition of existing resources. They can be parallelized with Business Process Execution Language (BPEL, 2002), which is the formal language for service composition in Big Web Services.

Our application framework supports the creation of mashups and advanced rules very simply, in any programming language that supports HTTP such as Perl, PHP, JavaScript etc. In Fig. 3, we can observe a function written in JavaScript in just a couple of lines that implements a locker on a virtual door, which only unlocks when the right RFID tag is presented to the *tikitag* device, which is coupled with the framework (located at *localhost* in port *8080*). In Fig. 4, a shell script is displayed that implements a distributed rule on sensor motes, in just seven lines of code. This rule checks the temperature of the room, measured by *sensor8* and if it is less than 20 degrees Celsius, then the green LED of *sensor5* is automatically turned on.

5 Evaluation

In this section, we perform a measurement-based evaluation of our approach. Initially, we describe the experimental setup we used during our tests and afterwards, we present a number of different experiments we committed, in order to measure the performance of the RESTful application framework.

5.1 Experimental Setup

The experimental setup we used is shown in Fig. 5. The framework has been installed on a laptop (Intel Core Duo 2.2 GHz). Around it, in a radius of 30 cm, we deployed a variable number of TMote Sky sensor motes, installed with the TinyOS RESTful software. One sensor mote was plugged in one of the USB ports of the laptop to serve as a base station, to forward IEEE802.15.4 wireless packets from/to the laptop through the serial-over-USB port. The details of the implementation regarding memory usage are given in Table 2. Our framework can be considered lightweight taking into account the fact that it requires only a few Mbytes for its full installation.

```
function getHttpRequest() {
   var xmlhttp = null;
   xmlhttp.open('GET', 'http://localhost:8080/tikitag/tags', true);
   xmlhttp.onreadystatechange = function() {
      if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
         var items = eval('(' + xmlhttp.responseText + ')');
         var secret_key = "04:BA:4A:B9:23:25:80";
         for (var i=0; i<items.length; i = i + 1) {
            if (items[i].t == secret_key)
             $('locked').innerHTML = "Door is unlocked!";
         }
         if (!unlocked)
          $('locked').innerHTML = "Door still locked.";
      }
   }
   xmlhttp.send(null);
}
```

Figure 3 Sample JavaScript code that checks a number of RFID tags in order to find the right one to unlock the door.

```
function check {
   if [ $? -le 20 ] ; then
      curl -d "color=GREEN" -X POST localhost:8080/sensors/sensor5/Light/
   fi
}
curl -s -X GET localhost:8080/sensors/sensor8/Temperature/ $1
check;
```

Figure 4 Sample shell script that implements a distributed rule on sensor motes.

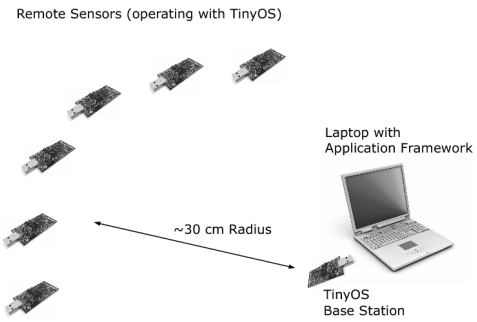

Figure 5 Experimental Setup used in Evaluation Procedure.

Table 2 Implementation memory sizes.

Module	*Memory Footprint*
RESTful Application Framework	3.2 Mbytes
RESTful TinyOS Software	24266 Bytes
Active Message	60 Bytes

5.2 Device Discovery Timing

In the first experiment, we were interested in measuring the time required at the device discovery procedure, that is how much time it takes for a number of sensor motes operating concurrently, to perform the interaction pattern we defined. We measured the amount of time until the discovery process was finished successfully and all the embedded devices were properly added to the framework's lists. We used a variable number of sensors motes to examine how well the device discovery procedure scales. We performed the experiment six times for each different number of sensor motes. The results are shown in Fig. 6. We can observe from the graph that in the worst case, when 16 devices are switched concurrently on, just 15 seconds are enough for the discovery procedure to be accomplished. The results indicate that our ad hoc approach is efficient and scalable, with a linear delay increase as the number of devices increases.

5.3 Application Framework Performance

In the next two experiments we tested the performance of the framework under realistic workloads. We created two different simulated use-cases to fulfill our purpose. In the first, we tested the system under ad hoc interaction with the sensor motes from multiple Web

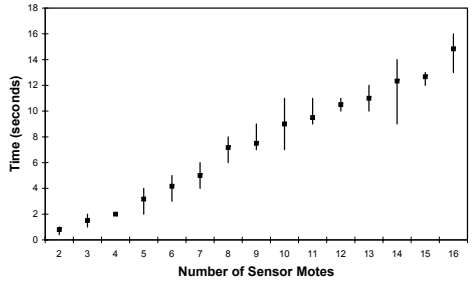

Figure 6 Device Discovery Execution Time.

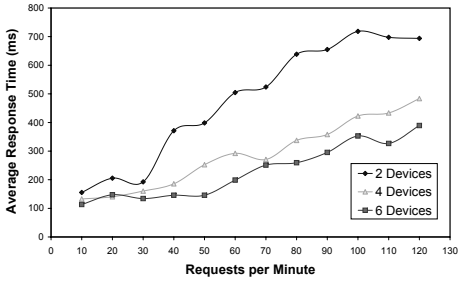

Figure 7 Average Request/Response Times.

clients while in the second, we created a continuous monitoring application scenario.

5.3.1 A Family with many Members

In this experiment we created a simulated scenario, in which multiple family members are interacting with their home through the Web, by sending concurrently requests to the framework. The home residents are choosing randomly embedded devices and RESTful Web Services that are offered by these devices (see Table 1, Resources 1-4) at random times, but with a frequency of one request/minute (for every resident). This randomization in request selection does not affect our evaluation efforts, since the difference in response time is negligible for different requests.

We performed a large number of different tests with variable numbers of family members and sensor motes. Each test ran for five minutes. We measured the amount of time needed, from the creation of a request from a family member to the arrival of the response, transmitted by the embedded device. Network delay was negligible. We focused our efforts on large numbers of home residents and at the same time, small numbers of sensor motes, to test the system under heavy workload.

Fig. 7 presents the results of the simulation executions focusing on timing. We must note that in all the tests, all the requests were satisfied. Examining the graph, we suggest that as the number of devices increases, the average request/response time is reduced significantly, since concurrent requests are less likely to target the same sensor mote. On the other hand, as the number of the requests increases, the average request/response time is increased as well. In the worst case, in which we have only two sensor motes and 120 requests per minute, that time is around 700 ms. In a rational workload of 60 requests per minute, which suffices for a typical family, mean times of 150 and 250 ms, for four and six motes respectively, constitute some indications of acceptable performance.

Fig. 8 depicts the percentage of failed attempts that have been made to satisfy a request, in comparison to the total attempts performed, during each execution of the simulation. A failed attempt can indicate loss

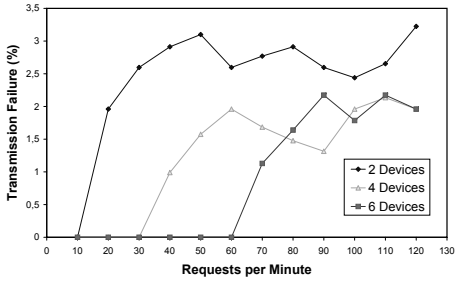

Figure 8 Percentage of failed Attempts.

of the request message from the framework or loss of the response message from the sensor mote. Causes of this effect are mostly collisions from simultaneous transmission attempts. Results indicate that increase of the device number has a small effect in the reduction of failures. This phenomenon happens because more devices can satisfy the fixed (each time) number of requests more effectively. In general, failed attempts are just a small percentage, less than 3.5% in the worst case. This fact denotes that, for this single-hop topology, collisions play a secondary role in the implementation's overall performance.

Since distance is an important parameter in such deployments, we increased the distance between the laptop and the remote sensor motes to 8 and 16 meters, and we performed again our tests, using a variable number of family members and a fixed number of four motes. Each test ran again for five minutes and the results can be observed in Fig. 9.

Distance plays a considerable role in the system's performance, as we can see from the graph. In the case the motes are placed in 8 meters distance, the degradation in performance is rather small and fluctuates between 10 and 20%, in comparison to the default simulation, when the distance is just 30 cm. The degradation in performance becomes significant when the motes are deployed in the distance of 16 meters and fluctuates between 20 and 70% from the default case of 30 cm.

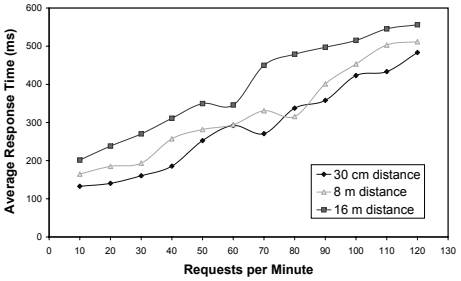

Figure 9 Average Request/Response Times in different distances.

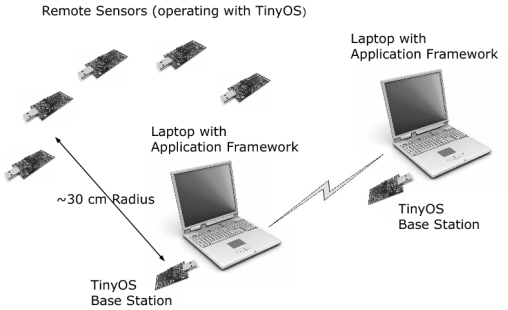

Figure 10 Experimental Setup for the Smart Metering Network.

However, even in the case of 16 meters distance, the performance is considered affordable, since less than 600 ms are needed to satisfy all requests in the worst-case scenario of 120 concurrent family members. We expect that bigger distances will increase even more the request/response times. We must note that the maximum indoor range of TMote Sky sensor motes is between 20 and 30 meters, so for larger deployments we need to consider multi-hop topologies. We leave this issue for future work. At last, we need to mention that transmission failures, in the case when the distance is increased, constitute still a negligible percentage.

5.3.2 A Smart Metering Network

In this last experiment, we tested our implementation in a streaming scenario. We invoked the last resource from Table 1, forcing the sensor motes to act as Smart Meters. Smart Meters, in general, are wireless devices that are able to measure the energy consumption of various electrical devices and control their operation.

We set the motes to perform streaming of energy consumption measurements from virtual electrical appliances in Watts, with frequency of one message per second. We employed several motes to obtain a considerable number of simulated Smart Meters. Since it was not adequate to test our framework's ability of receiving those messages, we decided to extend its structure in order to provide some advanced eventing functionality. We created a simple topic-based publish/subscribe infrastructure (Eugster et al., 2003) with *push* (Franklin and Zdonik, 1998) technology. We changed slightly our experimental setup, as shown in Fig. 10. We used a second laptop, where we also installed the RESTful application framework and attached to it a TinyOS base station. We settled this laptop near to the first one, connected to the same LAN, in a distance where its base station would be able to sense the messages that were transmitted from the remote sensor motes to the first laptop. This second laptop, as soon as it started, it expressed its interest to the first, through a POST request, for events of topic *Electricity*. The notion of *Web Hooks* was followed to facilitate event

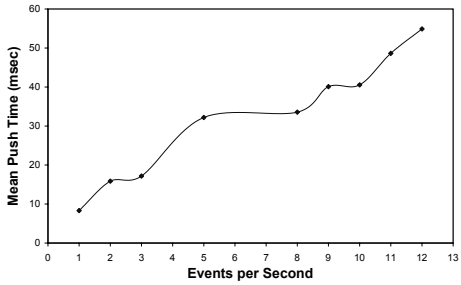

Figure 11 Eventing Push Performance.

notification. A Web Hook is a simple event notification via an HTTP callback. In the request's payload, a *callback URI* was added, to indicate where possible events would be forwarded. From now on, whenever there was a transmission of an energy consumption measurement, this message was considered as a new event and it was forwarded through the LAN from the first laptop (publisher) to the second (subscriber). The time difference between the moment the second laptop senses through its base station the event and the moment the very same event is received through the Internet is measured and the results are presented in Fig. 11, with a variable number of Smart Meters and correspondingly events per second.

From the graph we can infer that our framework performs well also in the case of a streaming-based situation with reasonable workload. In all the cases, events need less than 60 ms to be pushed from the publisher to the subscriber. This time delay can be considered tolerable, even for emergency and time-critical deployments. Obviously, this delay would be larger if the subscriber was not in the same LAN as the publisher or if there was a multi-hop metering topology but still, the results indicate our system's ability to handle a large amount of simultaneous streaming data effectively.

6 Related Work

Web-enabling physical objects is not a new topic and early approaches used physical tokens (such as barcodes or RFID tags) to retrieve information about objects they were attached to (Roy et al., 1999). In the Cooltown project (Kindberg et al., 2002) each thing, place and person had an associated Web page with information about it.

JXTA (Traversat et al., 2003) is a set of open protocols for allowing devices to collaborate in a peer-to-peer fashion. JXTA was among the first real attempts to bridge physical objects in the world with the Internet. Big Web Services have also been used to interconnect devices on top of standard Web protocols (Priyantha et al., 2008). These approaches are rather tightly coupled and not sufficient to deal with the constraints of mobile embedded devices.

Several systems for integration of sensor systems with the Internet have been proposed. SenseWeb (Kansal et al., 2007) is a platform for people to share their sensory readings, using Big Web Services to transmit data on a central server. Unfortunately, such approaches are based on a centralized repository and devices need to be registered before they can publish data, thus are not sufficiently scalable.

During the last two decades, many middleware infrastructures have been developed that targeted pervasive environments. Most of the these are rather complex and closed-system, application-specific approaches. Shamann (Schramm et al., 2004) was an early gateway system that enabled low-power devices to be part of wider networks. Its architecture motivated the implementation efforts in this paper, concerning mainly the integration of request queues and thread programming for the representation of devices inside our framework. Gaia (Romn et al., 2002) has aimed to provide an operating system for smart places.

Web-based middleware solutions, such as (Prehofer et al., 2007), have recently appeared, but most of them use the Internet as a transport protocol and do not fully exploit Web architecture as application protocol. Projects that specifically focus on re-using the founding principles of the Web as an application protocol have began to appear lately. In (Wilde, 2007), a path towards the integration of things into the Web is theoretically presented, where physical objects are made available through RESTful principles. As pointed out, no need for any additional API or descriptions of resource/function would be necessary when embedded devices are Web-enabled *by design*. This is one of the first projects that envisions the notion of the Web of Things. Following this direction, (Stirbu, 2008) also enables heterogeneous sensor devices in the Web, but it focuses mainly on the discovery of these devices. TinyREST (Luckenbach et al., 2005) offers a RESTful gateway that has some similarities to our implementation but it violates REST principles by introducing the extra verb *SUBSCRIBE*. In addition, no evaluation of the system is provided. An early prototype of the Web of Things has been recently developed in (Guinard and Trifa, 2009). In that paper, sensors capable of monitoring and controlling the energy consumption of household appliances offer a RESTful API to their functionality. Our paper differs from that work in that we extend their RESTful gateway into an application framework that supports multiple users concurrently, providing solutions to the problems of device discovery and service description in REST-oriented environments.

A large bibliography exists dedicated to Smart Homes. The majority of the solutions examines the integration of technology and services through home networking for a better quality of life. Microsoft's EasyLiving (Brumitt et al., 2000) is a middleware for building intelligent environments based on XML messaging, integrating geometric knowledge of people, devices and places. The adaptive house (Mozer et al., 2005) allows the home to program itself by observing the lifestyle of inhabitants and then learning to predict their needs, by means of neural networks. The Gator Tech Smart House (Helal et al., 2005) develops and deploys extensible smart house technologies, employing a service-oriented OSGi framework that facilitates service composition. Big Web Services have recently appeared in Smart Home projects. In (Aiello, 2006), an infrastructure for domestic networks based on Big Web Services is proposed, to address device heterogeneity.

Our work, profoundly influenced by the concept of the Web of Things, differs from all of these approaches in that it utilizes the Web as application layer to provide an interoperable Smart Home, based on an application framework that operates according to REST principles, capable of supporting a large number of simultaneous home residents, offering to them simplicity and flexibility in order to fully customize their home to their own needs. It addresses many issues related to Web-enabling everyday objects, from their discovery and description of functionality, to the uniform interaction with them.

7 Future Work

Here we describe our future research directions, beyond a more extensive performance evaluation in various environments, based on this current work. At first, this paper follows a device-centric approach. We will investigate ways of supporting a multi-hop architecture of information appliances, such as that of WSN.

Our device discovery procedure is currently ad hoc and not based on any standards. We will invest some effort in standardizing this process to fully comply with the Web-oriented architecture we envision.

Recently, TCP/IP networking has been ported in WSNs (Hui and Culler, 2008; Dunkels et al., 2004) and seems like the trend in future smart objects. This achievement brings interoperability at the network layer. We intend to explore interaction and integration

possibilities of our framework and TCP/IP-enabled embedded devices. We will examine how IP and non-IP devices (eg. RFID tags) could be combined to perform some advanced functionality.

Finally, a Graphical User Interface (GUI) will be implemented on top of our framework, to further abstract home automation procedure. Task scheduling and rule handling will be integrated to the framework, to facilitate the transition to a more powerful and user-friendly Smart Home.

8 Conclusion

In this paper, we have proposed to use the Web and its core architectural principles, in order to enable the Smart Home. We developed an application framework, based on REST architectural style, which offers a uniform, efficient, and standardized way of interacting with information appliances. Tremendous interoperability is offered, by exploiting the Web's infrastructure and by reusing the expertise and tools available, for building highly scalable Web applications that can serve a large number of simultaneous residents, who are able to fully automate their house.

Flexible applications on top of heterogeneous devices can be built with little effort and acceptable performance, with any programming language that supports HTTP while Web-based solutions for traditional challenges of ad hoc environments, such as device discovery and service description, are provided. Abstracting the actual complexity of embedded devices will enable a much larger ecosystem of everyday smart objects that can be re-tasked dynamically to solve any request at hand and to achieve advanced home automation.

Through this work, we envision physical things as first-class citizens of every house. Our project constitutes a contribution towards the full realization of a Web-based Smart Home.

References

Marco Aiello. The role of web services at home. In *IEEE Web Service-based Systems and Applications (WEBSA*, page 164, 2006.

G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures.* Springer, 2004.

Eric Bergman. *Information Appliances and Beyond.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000. ISBN 1558606009.

BPEL. Web services business process execution language, 2002. http://www.oasis-open.org/committees/wsbpel/.

Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven Shafer. EasyLiving: Technologies for Intelligent Environments. In *Handheld and Ubiquitous Computing*, pages 97–119, 2000.

Adam Dunkels, Thiemo Voigt, and Juan Alonso. Making TCP/IP Viable for Wireless Sensor Networks. In *Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN 2004), work-in-progress session*, Berlin, Germany, January 2004.

Eugster, Patrick Th., Felber, Pascal A., Guerraoui, Rachid, Kermarrec, and Anne-Marie. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003. ISSN 0360-0300.

Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, Irvine, California, 2000.

Michael Franklin and Stan Zdonik. Data in your face: push technology in perspective. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 516–519, New York, NY, USA, 1998. ACM. ISBN 0-89791-995-5.

Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in Proceedings of WWW (International World Wide Web Conferences)*, Madrid, Spain, April 2009.

Sumi Helal, William Mann, Hicham El-Zabadani, Jeffrey King, Youssef Kaddoura, and Erwin Jansen. The gator tech smart house: A programmable pervasive space. *Computer*, 38:50–60, 2005. ISSN 0018-9162.

Jonathan W. Hui and David E. Culler. IP is dead, long live IP for wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-990-6.

Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: an infrastructure for shared sensing. *IEEE Multimedia*, 14(4):8–13, 2007.

Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, and Philippe Debaty. People, places, things: web presence for the real world. *Mob. Netw. Appl.*, 7(5):365–376, 2002. ISSN 1383-469X.

P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. pages 115–148. 2005.

Thomas Luckenbach, Peter Gober, Stefan Arbanowski, Andreas Kotsopoulos, and Kyle Kim. TinyREST - a protocol for integrating sensor networks into the internet. *in Proc. of REALWSN*, 2005.

M. Mozer, R. Dodier, D. Miller, M. Anderson, J. Anderson, D. Bertini, M. Bronder, M. Colagrosso, and R. Cruickshank. The adaptive house. *IEE Seminar Digests*, 2005(11059):v1–39–v1–39, 2005.

Donald A. Norman. *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*, volume 1. The MIT Press, 1 edition, 1999.

Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful Web Services vs. "big" web services: making the Right Architectural Decision. In *Proceedings of the 17th International World Wide Web Conference*, pages 805–814, Beijing, China, April 2008.

Christian Prehofer, Jilles van Gurp, and Cristiano di Flora. Towards the web as a platform for ubiquitous applications in smart spaces. In *Second Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI), at Ubicomp 2007*, 2007.

Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 253–266, New York, NY, USA, 2008. ACM.

Restlet. Restlet, lightweight rest framework for java, 2007. http://www.restlet.org/.

Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, 2007.

Manuel Romn, Christopher Hess, Renato Cerqueira, Roy H. Campbell, and Klara Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1:74–83, 2002.

Want Roy, Fishkin Kenneth P., Gujar Anuj, and Harrison Beverly L. Bridging physical and virtual worlds with electronic tags. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 370–377, New York, NY, USA, 1999. ACM. ISBN 0-201-48559-1.

P. Schramm, E. Naroska, P. Resch, J. Platte, H. Linde, G. Stromberg, and T. Sturm. A service gateway for networked sensor systems. *Pervasive Computing, IEEE*, 3(1):66–74, Jan.-March 2004. ISSN 1536-1268.

Vlad Stirbu. Towards a restful plug and play experience in the web of things. *Semantic Computing, 2008 IEEE International Conference on*, pages 512–517, Aug. 2008.

B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J. Hugly, and E. Pouyoul. Project JXTA-C: Enabling a Web of Things. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pages 282–290, 2003.

WADL. Web application description language, 2005. https://wadl.dev.java.net/.

Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999. ISSN 1559-1662.

Erik Wilde. Putting things to REST. Technical Report UCB iSchool Report 2007-015, School of Information, UC Berkeley, November 2007.

WSDL. Web services description language, 2001. http://www.w3.org/TR/wsdl.