HomePort: Middleware for Heterogeneous Home Automation Networks

Thibaut Le Guilly, Petur Olsen, Anders P. Ravn, Jesper Brix Rosenkilde, Arne Skou
Department of Computer Science
Aalborg University, Denmark
{*thibaut,petur,apr,jbr,ask*}*@cs.aau.dk*

***Abstract*—Ambient Intelligence systems use many sensors and actuators, with a diversity of networks, protocols and technologies which makes it impossible to access the devices in a common manner. This paper presents the HomePort software, which provides an open source RESTful interface to heterogeneous sensor networks, allowing a simple unified access to virtually any kind of protocol using well known standards. HomePort includes means to provide event notification, as well as a tracing mechanism. The software is implemented and we report on initial experiments and provide an evaluation that shows the feasibility and scalability of the approach.**

***Keywords*-Service Delivery; REST; Middleware; Smart environment; Heterogeneous network;**

I. Introduction

Designing and setting up an Ambient Intelligence (AmI) system is a tedious task because it has a variety of sensors and actuators connected to different networks.

One may think that this is a once only problem when a system is installed. Nevertheless, when a building is planned, its AmI technologies are chosen based on the current availability, performance and price; but after some years a building administrator may wish to update or renew the system, and by then the technologies that were chosen are no longer the best ones, are outdated, or no longer supported. If the design and implementation then relies heavily on the technological choices, it might be impossible to change the system, and therefore it cannot be updated. However, if the design has a clear interface to heterogeneous networks from the beginning, technology changes will not affect the main system, and updating or changing a component is localized. Thus, it is justified to invest in a middleware with a service oriented interface to technologies.

In the following we present re-design and implementation of the HomePort middleware which was presented conceptually in [1]. Their HomePort enabled intercommunication between devices belonging to different sub-networks through a bridging layer. This layer was implemented and evaluated, but it turned out that it was easier to program interaction among devices with dedicated control software. The effort was then redirected to develop a middleware providing access to services through a REST interface. The result of the initial experiments helped design a device abstraction layer with a common interface for controllers; it makes it easier to interact with heterogeneous devices. This redesign of the HomePort system offers a new middleware for solving heterogeneity issues in an AmI environment. The main novel contributions in this middleware are:

- implementation of REST interfaces for home automation applications,
- security and confidentiality resolved through standard web-protocols, and
- an integrated tracing service that forms the basis for resolving accountability issues.

The architecture is implemented and tested and is now available as open source software for Linux platforms[1].

II. Background and related work

The REpresentational State Transfer (REST) architecture was introduced by Roy Fielding [2] in the year 2000. The term RESTful describes a system adhering to the REST architecture. The World Wide Web contains a large number of REST interfaces among which we mention Twitter and Flickr. REST relies on four principles:

- Use of Unique Resource Identifiers (URIs) to access resources provided by the web service.
- A uniform interface to access and modify resources, with GET, PUT, DELETE and POST requests.
- Self-descriptive messages, to enable representation of resources in different formats, such as HyperText Markup Language (HTML), Extensible Markup Language (XML), or JavaScript Object Notation (JSON).
- Interaction with resources are stateless, which means that requests are independent.

The power of REST comes from the fact that it is easily implemented using well known standards such as HTTP and XML. Moreover, the stateless principle ensures high scalability, as it reduces the server complexity since it does not need to maintain client state information.

The other well-known web service technology is the Simple Object Access Protocol (SOAP) [3]. The main difference between REST and SOAP is that REST is resource oriented while SOAP is remote method invocation. As the HomePort

[1]Available at https://github.com/home-port

software has to provide access to resources, REST is a natural choice.

An architecture comparable to HomePort is the DomoNet architecture [4]. It is based on two main notions: the TechManagers (TMs), that translate data to and from the sub-networks, and Device Web Services (DeviceWS), that are used by TMs to communicate with each other. Each DeviceWS handles a different device type, and provides a set of commands to access and control them. They also route messages between different TMs. The DomoNet architecture is similar to the original HomePort architecture which was changed, because control was not clearly isolated, in the sense that the system provides communication between different sub-networks rather than a common interface to them. However, the current HomePort system uses an adapter layer similar to the TMs to translate to and from the sub-networks.

The REST architecture has been used in other projects on the Web of Things [5]–[8] which reports good experiences with implementation of this architecture. The paper [5] gives a good overview and demonstrates how to use REST for the Web of Things; also it introduces the Server Sent Event notion, used in HomePort as an option for dealing with events. The HomePort software follows a similar architecture, but deals with the problems of heterogeneous networks as well. Another good example of a REST web server is found in [6]. A good overview of the REST architecture and the SOAP technology can be found in [7], as well as an interesting implementation of REST web services for Wireless Sensor Networks (WSNs), where a web server is deployed on each sensor node. Finally, [8] describes the TinyREST protocol, providing REST capabilities to WSNs.

In [9] an approach is found which is similar to HomePort; it streams data from sensors to clients using the HTTP protocol. They provide a good overview of different available methods and describe *stream feeds*, a Web primitive extending the traditional XML feeds to sensor streams.

Finally, we mention the Smart Energy Profile (SEP) [10] developed by the ZigBee alliance. SEP aims at standardizing communication to home automation devices from a Smart Grid. SEP uses the REST architecture for its communication model, and provides several function-sets that represent the minimum device behavior necessary to deliver a functionality. Among these we find a metering functionality, a pricing functionality, and a load control functionality. SEP is a proposed standard and thus provides guidelines for implementations rather than an actual one. HomePort differs from SEP in the way events are handled, which will be discussed more in Section III-E.

III. HomePort Components

One of the requirements for HomePort as middleware is that it should be possible to deploy it on a wide range of resource constrained devices with a minimum cost. Thus it is implemented in C, and the targeted platforms are Linux

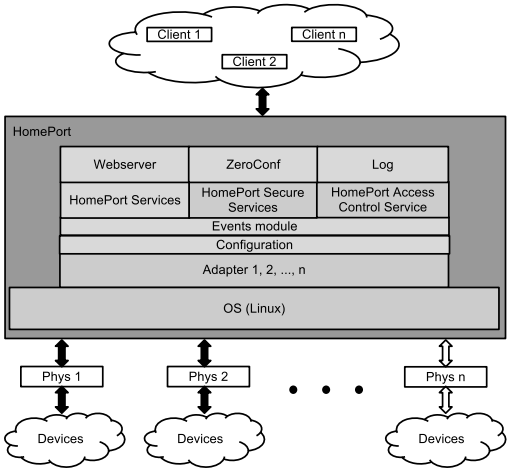

Figure 1. The HomePort architecture

based systems. Because the aim is to improve communication with different protocols and standards for interaction with AmI devices, the release of the software is made open source so that anybody can contribute to it, use it for free or reuse parts of the code.

The system, illustrated in Figure 1, has seven main modules that are described in more details below.

A) The service discovery module, that enables discovery of services inside a Local Area Network (LAN).
B) The web service module, providing the REST interface to access and control the services.
C) The HomePort services module, that holds the information about the services and what they provide.
D) The HomePort adapters, that implement access to the different heterogeneous networks connected to the system.
E) The event processor module, accepting and dispatching events from the services to the clients.
F) The access control module, that manages permissions to access the different services.
G) The log module, keeping a trace of requests and events.

It is interesting to note that different HomePort systems can cooperate with each other using their REST interfaces. Thus it is possible to have several systems handling different parts of an AmI installation, which makes it more fault resistant by avoiding a single point of failure. Moreover, a group of HomePort services can be aggregated into one and present a single system image to the clients.

A. Service Discovery

Service discovery establishes connections between service providers and service consumers. Service providers can publish their services on the network, along with their description and information on how to access them. The

service consumers can then browse for available services and use the ones they are interested in.

Several service discovery protocols are available. Popular ones are the DNS Service Discovery (DNS-SD) [11] protocol, and the Simple Service Discovery Protocol (SSDP). DNS-SD is part of the Zero Configuration networking (ZeroConf) [12] techniques to enable IP network, hostname resolution and service discovery without any need for DHCP or DNS servers. SSDP is part of the Universal Plug and Play (UPnP) [13] networking protocols enabling UPnP compatible devices to seamlessly discover each other. UPnP also provides control, event notification and a presentation layer. We note that UPnP supports the ZeroConf protocol for IP addressing, hostname resolution and service discovery layers.

ZeroConf is small compared to UPnP, thus it is used in HomePort. In fact, the UPnP control, event and presentation layers are not relevant as HomePort already includes them. Also the ZeroConf IPV4 Link Local, for automatic IP addressing is useful when a DHCP server is not available.

The selected implementation of ZeroConf is Avahi[2]. It implements fully the three techniques of ZeroConf, it is open source and free to use. Another advantage of Avahi is that it has two implementations. One targeting personal computers and similar devices, making use of an Avahi daemon enabling several applications to use it for publishing and browsing DNS-SD services. Another one for stand alone applications targeting embedded devices with limited power and memory. HomePort may use either one, enabling both workstations and embedded devices to publish their services.

All services registered in HomePort are published on the network through ZeroConf. Thus, ZeroConf enabled clients can browse them easily and access them directly. Clients that are not ZeroConf enabled, may use a GET request on the URL */devices*; it returns a list of all devices connected to the network, as well as the services that they provide.

B. Web Server

The web server is the external entry point to HomePort. It lets clients connect to the system through HTTP(S) requests, and follows the REST architecture. It processes client requests and redirects them to a service adapter. It can return error codes, either due to malformed requests from a client or errors detected in an adapter.

C. HomePort Services

The HomePort services hold information about all services that it knows, and works as an abstraction layer for the client. The information allows a client to determine what a service is offering, and how it can be accessed and controlled. An adapter must provide this information when registering a new service. Services are usually linked to devices that include information about the services that they provide. However, it is possible to define virtual services. An important example is the log facility. Some information is mandatory, and other is optional. For devices, a name and type is mandatory. Optional items include location, vendor ID, product ID, and a version number. For services, the name and the type is mandatory as well. Optional items include the unit used by the service, and a list of parameters including min and max values, scale, step, or possible values that the service can accept. These parameters enable a client to send commands to access or control the service.

D. HomePort adapters

The adapters are the MAC layer of the HomePort system. They provide translation to and from devices located on the sub-networks. The adapters communicate with the rest of the system through a public API. This API provides functions for registering and un-registering services, sending events on services, and retrieving a service from the system service list. When registering a service, an adapter provides callback functions that are invoked when a client performs an HTTP request on it. However, only a GET callback is mandatory. If a request is issued with a method that is not implemented by the addressed service, the service layer returns an error code. When a request is issued that modifies the state of a service, the updated state is returned to the client as a confirmation that the request was successful. If the adapter cannot fulfill the request, it returns an error indication.

A design goal for HomePort is to be able to dynamically load new adapters. This enables integrating new sub-networks without restarting the system. For this, the system has an internal virtual service, where a client can request download of a new adapters from a specified repository. The adapter corresponding to the request is then downloaded and executed dynamically by the system, enabling seamless integration of a new network. In the same way, a client is able to deactivate adapters. Also adapters can be updated (deactivated and reactivated) in order to bring in new functionalities or to fix issues.

E. Event processing

A main task in an AmI system is to deliver events from services to clients. Here, different approaches can be taken.

A basic approach is polling: a client polls a resource and detects the moment when the state of this resource changes. The advantage is that it is really simple to implement, as no particular event handler is required. However, it has some well known drawbacks. First, depending on the polling interval of the client, a delay is induced from the moment an event occurs until the client is notified. Second, the server load may experience local peaks if it receives polling requests from many clients at the same time. Finally, the polling requests may overload the network.

[2]http://www.avahi.org

In order to resolve these issues, the SEP standard [10] proposes a model in which clients register their interest in some events in the server, and get contacted by the server when an event for which they registered occurs. This is essentially the Observer Pattern known from Object Oriented Programming. This approach solves the issue of high load on the server and the network, as well as the delay for receiving events. However, this approach has three main drawbacks: first, it undermines the REST architecture principle, which states that the system should not keep state information about the clients. In fact, it might not be scalable in case of a large number of clients registering for events. Second, it requires clients to support a web server, which is a large overhead for small systems. Finally, a client may be connected to the server behind a router, which makes it impossible for the server to send notifications.

The approach that we adopt in the HomePort system is based on the Server-Sent events draft standard from W3C [14]. The Server-Sent events draft defines an API for using an HTTP connection to receive so-called push notifications from servers. The concept is that a client opens an HTTP connection to the server, and keeps it open, so the server can push information to the client when an event occurs. In order to register interest for events, a client creates a custom pipe on the server. When creating the pipe, the client specifies which services it wants notification from. It then connects to this pipe and the server pushes the information into it. The client can also register for events such as inclusion or exclusion of services in the system in order to be informed when a service appears or disappears. This approach solves both the REST issue, the network load and the event delay issues. However, some might argue that it does not solve the server load issue, as the server needs to have an open thread for each connected client. This problem can be solved by using an event driven architecture. It is expected that the next release of HomePort will adopt such an architecture in order to improve its scalability.

F. Access control

Access to services consist of read operations (GET requests), to access a service state, or write operations, to modify the state of a service, add or remove a service (PUT, POST and DELETE requests). Because different clients have different roles, it is necessary to delimit what each client can do. The access control module controls the client's rights through an access table. This table contains information about what kind of request each client can perform on each service. So when a client is requesting access to a service, a lookup is made in this table in order to determine if the client has the right to perform this request. If it has not, an error code is sent back by the server indicating the lack of permission.

Clients are differentiated and identified by using their SSL certificates. Therefore, each certificate will give different access rights. In order to set up the rights, it is necessary to have a "master" certificate which gives the rights to modify the access control table.

G. Log

The log module is an essential component of HomePort. If it is an important tool for debugging, it is also necessary in order to enable accountability measures [15]. The log can be accessed remotely from a client through the url */log*, and can also be monitored dynamically through the event system by registering to log events. If security is enabled on the server, the log will be accessible only through a secured connection as it can contain sensitive information. Two type of log messages are defined. The first type record a client request, and contains: *timestamp - client ip address - HTTP method - queried URL*. The second type register an error that occurs on the server, and is structured as follows: *timestamp - error token*.

The log is stored in a circular buffer in order to avoid consuming too much memory. The size of the log can be configured at run time through a configuration file.

IV. Security

Security is important when accessing and controlling home automation equipment. For that reason, the communication between HomePort and the clients can use the Transport Layer Security (TLS) protocol.

HomePort has three different configurations for security. The first one is without security, which means that the communication is through the open HTTP protocol. The second one has both secured and non-secured communication. In that case, two separate web servers are running, one communicating with HTTP and the other one communicating with HTTPS. This configuration enables access control for sensitive devices (such as door locks or windows), while still allowing access to less sensitive devices (such as switches or dimmers) for clients without secured communication. In that case, it is the responsibility of the adapters to register their services as secured or non secured. The last configuration is the one where only HTTPS communication is enabled. This is certainly the one we would recommend for real deployment.

Note that HomePort does not handle issuing of certificates used for secured communications. It is the responsibility of the application providers to have procedures for issuing them.

Another important point is that HomePort is only as secure as the sub-networks are. In fact, the system cannot increase the security of a sub-network. If one of them has security issues, they will remain even when integrated in the system. For example, if a wireless network does not encrypt its data, it is not possible for the system to provide encryption for communication with devices on that network. However, the lack of security of one sub-network will not

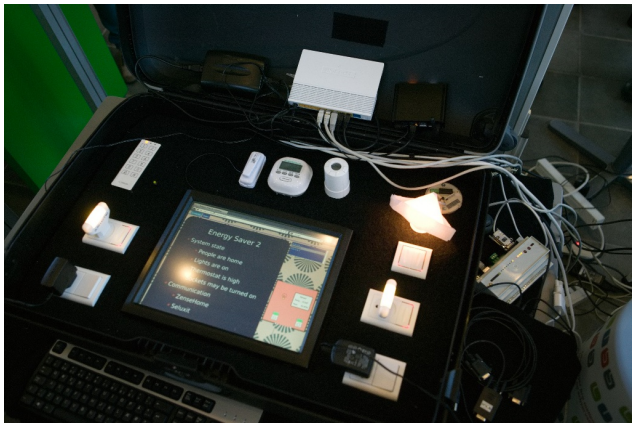

Figure 2. Picture of the experimental suitcase

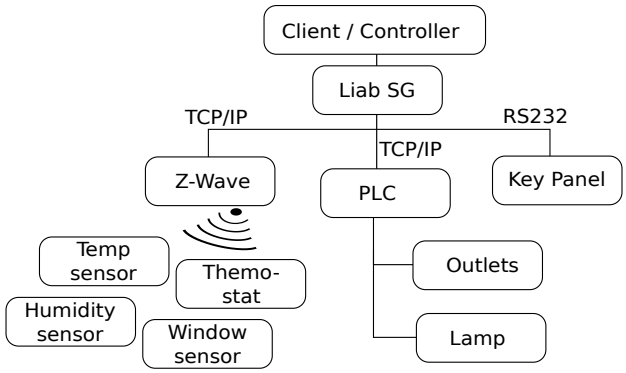

Figure 3. Experiment set-up

affect others, as they are well separated and cannot directly interact with each other.

A last important security issue in the system concerns dynamic loading of adapters. When downloading code from the internet, it may be malicious and break the system. Even without malicious code, untested code can contain bugs that could shut the system down if no separation of concerns is implemented. In order to solve this issue, it is necessary to limit the rights of the adapters on what they can actually do and what they can access, thus implementing a sandbox [16] for the adapters. However, it is not clear yet how to implement such a sandbox, and more investigation is needed in order to define what limitations are required.

V. Experiments

A. Description

A first version of HomePort has been implemented, including the functionalities described above except for dynamic loading of adapters and the access control module. For the experiments, we set up an in-house home automation system inside a suitcase, shown in Figure 2, so we can carry the system around for demonstration purposes. The experiment includes three sub-networks, shown in Figure 3: a wireless network connected to the HomePort system through a Z-Wave gateway via TCP/IP, a Power Line Communication network also communicating with HomePort through TCP/IP, and an access control panel connected through RS232. The HomePort system runs on a LIAB SG Linux computer with an Atmel AT91SAM9260 CPU, 64MB of SDRAM, 256MB of NAND flash, an RS232 port, and an Ethernet port.

The Z-Wave network includes a temperature and a humidity sensor, a thermostat, and a window sensor. The PLC network includes two outlets, and one dimmable lamp.

A web client interface was used to interact with the system. We used the Google Web Tool Kit API to implement it in order to accelerate the development process. Thanks to this interface, we were able to implement use case scenarios to demonstrate the use of HomePort services in an AmI system. One scenario simulates regulation of room temperature in modes where the window is opened or it is closed. The thermostat is lowered when the window is opened, and set back to its previous point when the window is closed. Another example scenario regulates the room temperature and light depending on the presence of people in the house. The access control panel is used to simulate people leaving or entering the house. When people leave the house, the thermostat is set to low and all the lights are turned off. When they re-enter, the thermostat and all the lights are set back to the state they were. A third scenario simulates burglars entering the house. The window sensor is triggered simulating a break in. When the alarm is triggered, all the lights are switched on, and the access control buzzer is triggered, until the correct pin code is entered.

B. Outcomes

From experiments, we were able to draw some conclusions. First, by implementing the adapters for each of the sub-networks, we could work with adapter implementation and see how much effort is required to implement them. Interfacing with the HomePort system is easy, but implementing an adapter can be more complex depending on the underlying protocol. It took for example two workdays to implement an adapter for the RS-232 control panel, while it took five workdays to implement one for the Z-Wave network for the first time. Registering services and devices, sending events or receiving requests on them is intuitive. However, we noticed that it was useful for the adapters to be able to attach some data when registering a service, in order to help identifying it. We thus added a field in the service description that can be used to store any data. Overall we can say that implementation of adapters went smoothly. Note that the HomePort repository[1] provides developers with code examples in order to help them develop their own adapters.

On the client side we could experience the power of the REST architecture. The development of a REST client is

Table I

(a) Influence of requests per second

Requests per sec.	30	150	300	600
Mean Response time (ms)	2.79	3.39	4.30	5.07

(b) Influence of number of services

Services	5	50	100	1000
Mean Response time (ms)	0.51	0.63	0.83	3.33

really easy, as all that is required to interact with the services is an HTTP library, already existing in the Google Web Tool kit. Similarly, as a library for Server-Sent events was available, we could interact with service events with less than 10 lines of code. The parts that required the most work were as usual implementation of the GUI for the application. We can thus affirm that the use of a REST architecture makes it easy to implement clients.

By experiencing with the event module from the client, we have been able to see that it was resource efficient from a client perspective as only one HTTP request was necessary in order to receive notification from services.

The experiment also enabled us to test the maturity of the HomePort software. We have been running this experiment for 7 days without interruption, and without experiencing any issue with it. The client interface was reconnected once per day in order to simulate normal use.

C. Performance

In order to evaluate the scalability of the server, we ran some load tests on the server[3]. For the purpose of this experiment, a virtual adapter, returning a static value for each request, was used in order to avoid the influence of the communication with end-devices on the results. The experiments evaluated the response time of the server depending on the number of clients, and the number of services served by the server. Table I(a) shows the influence of the number of requests per second on the response time, while Table I(b) shows the influence of the number of services. For these experiments, each client performs three requests on the server, one on the URL */devices*, to retrieve the list of services, one on the first registered service, and one on the last registered service. For the evaluation of the number of requests per seconds, fifty devices, each providing five services were registered on the server, while for the evaluation of the number of services, ten clients per seconds were performing the set of requests.

The number of requests that can be handled per seconds is probably sufficient for most applications. We note that scalability for the number of services registered in the web server could be improved by modifying the data structure used for services. In fact, a linked list is now used for that purpose, and we believe that by using a trie structure the performance will improve. This change is planned for the next release of the software.

Overall, the implementation has an acceptable performance.

[3]Testing environment: Lenovo x230, 8GB of RAM, Intel Core i7-3520M CPU @ 2.90GHz 4, Ubuntu 12.04 64bits, homeport v0.2, libmicrohttpd v0.9.23. Testing tool is Tsung v1.4.1

VI. Future work

A. Remaining modules

As mentioned in the experiments, the access control module as well as the dynamic loading of adapters have not yet been implemented, thus the current version of HomePort does not enable addition or removal of sub-networks on the fly. However, devices and services can be dynamically added and removed if the sub-network supports it. Concerning the access control module, the implementation should not be a problem, as the concept is rather simple. For the dynamic loading of adapters, we need to define a good way of sandboxing the adapters and separate them properly from the core of the system. Dynamic loading and execution of code has already been implemented in the system, but it has not been included yet because the security issues have not been solved.

B. Services Representation

In its current form, HomePort presents services through XML. But depending on the controller, XML might not be the best language. A web client implemented in JavaScript for example, would process JSON data more efficiently as there would be no need for parsing. The XML language has also been criticized for its overhead compared to other description languages. A solution is the Efficient XML Interchange (EXI) being developed by the W3C. EXI is a binary format of XML, and reduce both the verbosity of XML as well as the cost for parsing.

It would be useful for HomePort to be able to send data in different formats depending on the *"Accept-Language"* field of the HTTP requests sent by the clients.

C. Control language

In order to cater for critical controllers, the software could have an internal control module that could be managed by a client through a virtual service. In order to make it simple for the user to define control operations, a controller language is being developed. This language will enable clients to specify control over the AmI environment, and upload it to the controller module. The model defined by the user will be compiled into a language that the system can understand before uploading it. The model currently under consideration is based on timed automata.

VII. Conclusion

We have presented a middleware to provide a uniform service interface to heterogeneous networks, in order to improve maintainability and diversity of equipment in AmI

environments. Notable features are a REST architecture and event notifications to clients that reduces both the network and client load, and under certain conditions the server load. By choosing a low level implementation language, an open source and free target platform and by making the software open source as well, the system can be implemented at low cost on limited resources hardware.

With the increasing research in the Smart Grid area, where AmI environments can provide data and control over appliances to the energy provider in order to help them improve the energy distribution and consumption, having a common interface for communicating with different technologies and protocols is necessary. The HomePort system aims to provide such an interface.

ACKNOWLEDGMENT

The research presented in this paper has been partially supported by the EU Artemis project ENCOURAGE[4] and the Danish ForskEL project TotalFlex[5].

REFERENCES

[1] J. Brønsted, P. Madsen, A. Skou, and R. Torbensen, "The homeport system," in *Proc. of 7th IEEE Consumer Communications and Networking Conference (CCNC)*, Jan. 2010, pp. 1 –5.

[2] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[3] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, Mar. 2005.

[4] V. Miori, L. Tarrini, M. Manca, and G. Tolomei, "An open standard solution for domotic interoperability," *Consumer Electronics, IEEE Transactions on*, vol. 52, no. 1, pp. 97 – 103, feb. 2006.

[5] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *In Proc. of Internet of Things (IOT)*, 29 2010-Dec. 1 2010, pp. 1 –8.

[6] M. Weiss and D. Guinard, "Increasing energy awareness through web-enabled power outlets," in *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, ser. MUM '10. New York, NY, USA: ACM, 2010, pp. 20:1–20:10.

[7] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the web of things," in *Networked Sensing Systems (INSS), 2009 Sixth International Conference on*, june 2009, pp. 1 –4.

[8] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim, "Tinyrest - a protocol for integrating sensor networks into the internet," in *in Proc. of REALWSN*, 2005.

[9] R. Dickerson, J. Lu, J. Lu, and K. Whitehouse, "Stream feeds: an abstraction for the world wide sensor web," in *Proceedings of the 1st international conference on The internet of things*, ser. IOT'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 360–375.

[10] ZigBee Alliance, "Smart energy profile 2.0," Working Draft, ZigBee Alliance, Internet-Draft, 2011.

[11] S. Cheshire and M. Krochmal, "DNS-based service discovery," Working Draft, IETF Secretariat, Fremont, CA, USA, Internet-Draft draft-cheshire-dnsext-dns-sd-09.txt, Feb. 2011.

[12] D. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[13] M. Jeronimo and J. Weast, *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003.

[14] W3C, "Server-sent event," Working Draft, W3C, Internet-Draft, 2012.

[15] D. L. Métayer, M. Maarek, E. Mazza, M.-L. Potet, S. Frénot, V. V. T. Tong, N. Craipeau, and R. Hardouin, "Liability issues in software engineering: the use of formal methods to reduce legal uncertainties," *Commun. ACM*, vol. 54, no. 4, pp. 99–106, 2011.

[16] H. Kaiya and K. Kaijiri, "Specifying runtime environments and functionalities of downloadable components under the sandbox model," in *Principles of Software Evolution, 2000. Proceedings. International Symposium on*, 2000, pp. 138 – 142.

[4] http://www.encourage-project.eu
[5] http://totalflex.dk/In%20English/