2011 Eighth IEEE International Conference on Mobile Ad-Hoc and Sensor Systems

REST Enabled Wireless Sensor Networks for Seamless Integration with Web Applications

W. Colitti, K. Steenhaut and N. De Caro
Dept. ETRO-IBBT, Vrije Universiteit Brussel
Dept. IWT, Erasmushogeschool Brussel
Pleinlaan 2, 1050, Brussels, Belgium
e-mail: walter.colitti@etro.vub.ac.be

Bogdan Buta and Virgil Dobrota
Dept. of Communications
Technical University of Cluj-Napoca
26-28 Baritiu Street, 400027 Cluj-Napoca, Romania
e-mail: Bogdan.Buta@net.utcluj.ro

***Abstract*—The use of Internet Protocol (IP) technology in Wireless Sensor Network (WSN) is a key prerequisite for the realization of the Internet of Things (IoT) vision. The IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard enables the use of IPv6 in networks of constrained devices. 6LowPAN enables the use of Service Oriented Architectures (SOAs) in WSN. The Internet Engineering Task Force (IETF) has defined the Constrained Application Protocol (CoAP), a web transfer protocol which provides several Hypertext Transfer Protocol (HTTP) functionalities, re-designed for constrained embedded devices. CoAP allows WSN applications to be built on top of Representational State Transfer (REST) architectures. This considerably eases the IoT application development and facilitates the integration of constrained devices with the Web. This work describes the prototype design and development of a Web platform which integrates a REST/CoAP WSN with a REST/HTTP Web application and allows a user to visualize WSN measurements in the Web browser. Since the WSN Web integration relies on a non-transparent gateway/server, we also show how to provide transparent cross-protocol resource access by means of an HTTP-CoAP proxy. The paper describes the major building blocks, functionalities and the implementation approach.**

Keywords-component; Web applications, REST, HTTP, CoAP

I. INTRODUCTION

The Internet of Things (IoT) community has envisioned that trillions of sensor equipped smart objects will connect physical environments to the Internet, unleashing exciting possibilities and challenges for a variety of application domains, such as smart metering, e-health, logistics, building and home automation [9].

The use of Internet Protocol (IP) technology in Wireless Sensor Network (WSN) is a key prerequisite for the realization of the IoT vision. This has been widely accepted by major research and industrial players which, through the IP for Smart Objects (IPSO) Alliance[1], are promoting the use of IP technology in embedded devices.

Among the standardization community, the Internet Engineering Task Force (IETF) has done substantial standardization activity on IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [10]. This new standard enables the use of IPv6 in Low-power and Lossy Networks (LLNs), such as those based on the IEEE 802.15.4 standard [12].

Although 6LowPAN has paved the way for the integration between WSN and the Internet, the mere interconnection at the network layer is not sufficient to fully exploit the possibilities offered by IoT applications. Nowadays, the vast Internet application world is largely dominated by Web applications based on Representational State Transfer (REST) architectures and on the Hypertext Transfer Protocol (HTTP).

Reusable web services based on REST and HTTP technologies in WSN would considerably ease the IoT deployment and management. In addition, the software reusability would reduce the complexity of the application development and smooth the integration of constrained devices with the Web. The use of standard Web technologies in the IoT is commonly known as the Web of Things (WoT).

The WoT has started playing a major role in the research community. Various research papers proposing REST/HTTP architectures for WSNs have recently appeared. The work in [1] proposes a RESTful architecture which allows instruments and other producers of physical information to directly publish their data. In [2], the authors propose a REST/HTTP framework for Home Automation. The work in [3] proposes a toolkit which allows the user to create web services provided by a specific device and to automatically expose them via a REST API. The authors in [4] show how different applications can be built on top of RESTful WSNs. The work in [5] illustrates a real world implementation of a RESTful WSN. The network is deployed across various university buildings and it is designed for the development of applications and services for the university community.

The use of web services in the IoT is not straightforward as a consequence of the differences between Internet applications and IoT applications. IoT applications are short-lived and web services reside in battery operated devices which most of the time sleep and wake up only

[1] http://ipso-alliance.org/

978-0-7695-4469-4/11 $26.00 © 2011 IEEE
DOI 10.1109/MASS.2011.102

when there is data traffic to be exchanged. In addition, such applications require a multicast and asynchronous communication compared to the unicast and synchronous approach used in standard Web applications [13]. This problem has triggered the need of new standard protocols which are based on standard Web technologies but that consider the limitations of constrained devices. The Internet Engineering Task Force (IETF) Constrained RESTful environments (CoRE) Working Group has defined a REST based web transfer protocol called Constrained Application Protocol (CoAP). CoAP includes several HTTP functionalities re-designed for constrained embedded devices such as sensor nodes [14].

The first contribution of this work is to show the prototype design and development of a Web platform for WSN monitoring. The platform integrates a REST/CoAP WSN with a REST/HTTP Web application and allows a user to visualize WSN measurements directly in the Web browser. The interaction between the WSN and the Web application is enabled by a machine acting as web server and gateway. The paper describes the main Web application functionalities and the main gateway/server building blocks.

The fact that the Web platform relies on an intermediate Web server implies that the Web client does not access the WSN transparently. The lack of transparency increases the complexity of the application development (client and server tightly coupled) and hampers interoperability and scalability. In order to provide the Web client with transparent WSN resource access, the gateway should also offer proxy functionality. It should be able to map an HTTP request into a CoAP one and vice versa. The IETF CoRE group is discussing on HTTP-CoAP mapping issues and has given guidelines on how to execute a transparent HTTP CoAP translation [21].

The second contribution of this work is to describe the design and development of a preliminary HTTP-CoAP proxy. The proxy allows an HTTP client to transparently access resources in a CoAP server. At the moment the transparent proxy module has been deployed and tested as standalone module and is going to be integrated in the aforementioned WSN monitoring Web application. This paper describes the main mechanisms of the transparent cross-protocol resource access and our implementation approach.

The rest of the paper is organized as follows. Section II describes how CoAP provides WSNs with REST functionalities and describes the major differences between CoAP and HTTP. Section III illustrates the main functionalities of an end-to-end Web platform and describes the major building blocks of the gateway/server which enables the integration of the WSN with the Web. Section IV shows the functionalities and implementation approach of a transparent HTTP-CoAP proxy. Section V concludes.

II. RESTFUL WIRELESS SENSOR NETWORKS

One of the major benefits of IP based networking in WSNs is to enable the use of standard web services based on the widely known REST architecture. REST platforms allow applications to rely on loosely coupled services which can be shared and reused. In a REST architecture a resource is an abstraction controlled by the server and identified by a Universal Resource Identifier (URI). The resources are decoupled from the services and therefore resources can be arbitrarily represented by means of various formats, such as Extensible Markup Language (XML) or Java Script Object Notation (JSON). The resources are accessed and manipulated by an application protocol based on client/server request/responses. REST is not tied to a particular application protocol. However, the vast majority of REST architectures nowadays use HTTP [8].

In March 2010, the IETF CoRE Working Group has started the standardization activity on CoAP. CoAP is a web transfer protocol optimized for resource constrained networks typical of IoT applications. CoAP is based on a REST architecture in which resources are server-controlled abstractions made available by an application process and identified by Universal Resource Identifiers (URIs). The resources can be manipulated by means of the same methods as the ones used by HTTP being GET, PUT, POST and DELETE.

CoAP is not a blind compression of HTTP. It consists of a subset of HTTP functionalities which have been re-designed taking into account the low processing power and energy consumption constraints of small embedded devices such as sensor motes. In addition, various mechanisms and have been modified and some new functionalities have been added in order to make the protocol suitable to IoT applications. The HTTP and CoAP protocol stacks are illustrated in Figure 1.

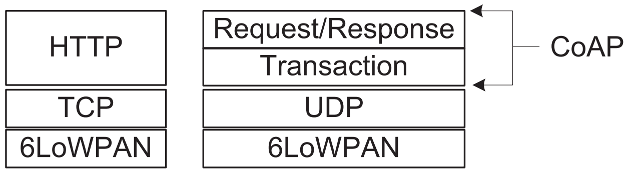


Figure 1. HTTP and CoAP protocol stacks.

The first significant difference between HTTP and CoAP is the transport layer. HTTP relies on the Transmission Control Protocol (TCP). TCP's flow control mechanism is not appropriate for LLNs and its overhead is considered too high for short-lived transactions. In addition, TCP does not provide multicast support. CoAP is built on top of the User Datagram Protocol (UDP) and therefore has significantly lower overhead and supports multicast.

Figure 1 also shows that CoAP is organized in two layers. The Transaction layer handles the single message exchange between end points. The Request/Response layer is responsible for the transmission of requests and responses for the resource manipulation and transmission. This is the layer where the REST based communication occurs. REST request and responses are piggybacked on the messages exchanged on the Transaction layer.

The dual layer approach allows CoAP to provide reliability mechanisms even without the use of TCP as transport protocol. In fact, the Transaction layer provides a default timeout and exponential back-off retransmission mechanisms. In addition, it enables asynchronous communication which is a key requirement for IoT applications. When a CoAP server receives a request which it is not able to handle immediately, it first acknowledges the reception of the message and sends back the response in an off-line fashion. Tokens are used for request/response matching in asynchronous communication.

The transaction layer also provides support for multicast and congestion control [16].

One of the major design goals of CoAP has been to keep the message overhead as small as possible and limit the use of fragmentation. HTTP has a significantly large message overhead. This implies packet fragmentation and associated performance degradation in the context of LLNs. CoAP uses a short fixed-length compact binary header of 4 bytes followed by compact binary options. A typical request has a total header of about 10-20 bytes. The CoAP message format is illustrated in Figure 2.

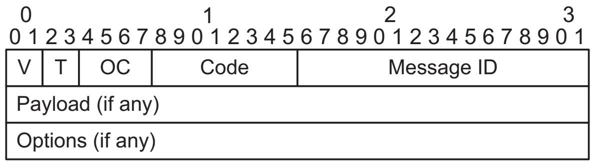


Figure 2. CoAP message format.

Since a resource on a CoAP server likely changes over time, the protocol allows a client to constantly observe the resources. This is done by means of observations: the client (the observer) registers itself to the resource (the subject) by means of a modified GET request sent to the server. The server establishes an observation relationship between the client and the resource. Whenever the state of the resource changes, the server notifies each client having an observation relationship with the respective resource. The duration of the observation relationship is negotiated during the registration procedure [15].

Although CoAP is work in progress, various open source implementations are already available. The two most known operating systems for WSNs, Contiki and Tiny OS, have already released a CoAP implementation. In addition, there are three other open source implementations not specifically designed for WSNs: an implementation in C language called *libcoap*[2], one in Python language called *CoAPy*[3] and one in Java called *jcoap*[4].

III. A Web Application for WSN Monitoring

The use of an IP based communication and a REST based Web architecture in the LLNs facilitates the integration of WSNs with Internet based Web applications. This Section describes the prototype design and development of an end-to-end architecture integrating a CoAP over 6LowPAN Contiki based WSN with an HTTP over IP based application. The aim of the Web application is to allow a user to access WSN data via the Internet directly from a Web browser, as illustrated in Figure 3.

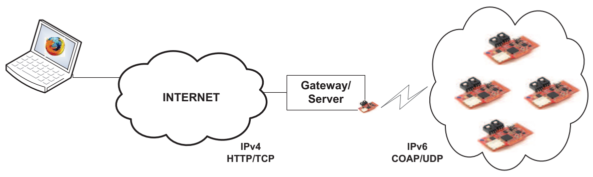


Figure 3. Integration between WSNs and the Web.

As shown in Figure 3, the interaction between the WSN and the Web application is enabled by a gateway which has an RPL border router interface to communicate with the WSN. The prototypes of the application and of the gateway have been deployed in a testbed consisting of 10 Zolertia Z1 sensor motes running Contiki. This section firstly illustrates the main building blocks of the gateway (section A) and secondly it describes the application and its main operations (section B).

A. A Gateway Architecture for Web-WSN integration

In this work, the gateway integrating the CoAP based WSN with the HTTP based Web application consists of a Linux Ubuntu machine with a Contiki-gateway attached via USB port. The main building blocks of the software running on the gateway are illustrated in Figure 4.

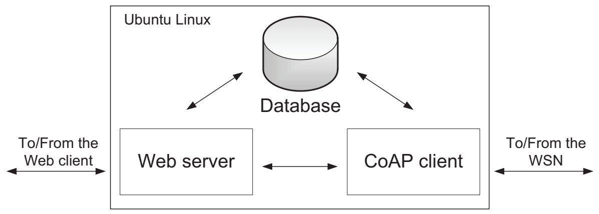


Figure 4. Gateway's main building blocks.

As illustrated in Figure 4, the gateway's main building blocks are the web server, the database and the CoAP client. For simplicity, the first gateway prototype runs all the building blocks on the same machine.

[2] http://libcoap.sourceforge.net/
[3] http://coapy.sourceforge.net/
[4] http://code.google.com/p/jcoap/

The web server includes a set of services which are used to retrieve data, either from the database or directly from the CoAP client. When the Web server sends the Web client historical data already available in the database, the web server directly accesses the database without communicating with the CoAP client and sends the data back to the Web client. When the Web Server needs to send fresh data coming from the WSN (upon client request or upon changes of the WSN resources), the web server bypasses the database and directly communicates with the CoAP client. Since the Web application has been developed with Google Web Toolkit (GWT), the web server needs to be a servlet container. In our implementation we have used Apache Tomcat 6[5].

The database stores data coming from the CoAP client and makes them available to the web server. The database chosen is Apache CouchDB [6]. Apache CouchDB is a document-oriented database which can be queried and indexed with the MapReduce technique using JavaScript. It stores JSON documents and provides a RESTful API. Since the CoAP client receives WSN data already in JSON documents, the storage operation is rather simple and does not require any intermediate data manipulation.

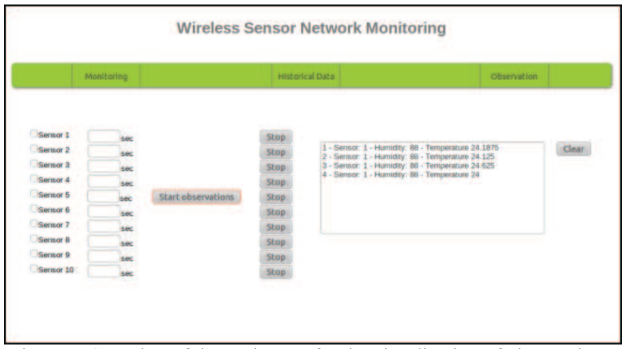
The *libcoap* CoAP client module is responsible for communicating with the WSN.

B. *Monitoring application*

The monitoring application has three main functionalities:

- **Current measurements**: the user can query the WSN in order to obtain the current temperature and humidity data. The user can either query a single mote or can ask for the average data. In the former case, the CoAP client in the gateway only queries the single mote, while in the latter case the CoAP client queries all the motes in the WSN and the web server calculates the average.
- **Historical measurements**: the user can visualize daily or monthly average temperature and humidity. The application allows the user to choose the past day or month which he is interested in.
- **Observations**: the user can subscribe to a single mote in order to receive notifications when the data change. The user can select the length of the observation period and can stop the observation.

A snapshot of the Web page for the visualization of observations is illustrated in Figure 5.



```
1 - Sensor: 1 - Humidity: 88 - Temperature 24.1875
2 - Sensor: 1 - Humidity: 88 - Temperature 24.125
3 - Sensor: 1 - Humidity: 88 - Temperature 24.625
4 - Sensor: 1 - Humidity: 88 - Temperature 24
```

Figure 5. Snapshot of the Web page for the visualization of observations.

IV. TRANSPARENT HTTP-COAP PROXY

In the Web application described in the previous section, the HTTP client does not access the WSN resources in a transparent way. This is due to the presence of the Web server which knows how to access WSN resources on CoAP servers. The lack of transparency implies that client and servers are tightly coupled, which increases the complexity of the application development and hampers interoperability and scalability. In order to provide the Web client with transparent WSN resource access, the gateway should also offer proxy functionality, as suggested by the IETF CoRE group [21]. Following the guidelines given in [20] and [21], this section discusses on a preliminary implementation of an HTTP-CoAP proxy (HC proxy) which provides cross-protocol resource access. It transparently maps an HTTP request coming from an HTTP client into a CoAP request to be forwarded to a CoAP server.

The HC proxy has been implemented as one-way proxy, meaning that it can translate one protocol into another but not vice versa. A two-way would also be able to map a CoAP request into an HTTP one. Since it is not requested in our scenario, its implementation is out of the scope of this work.

An HC proxy can be placed in the same network domain as the server (server-side proxy), in the same network domain as the client (client-side proxy) and in in a network domain external to both endpoints (external proxy). Our choice is to use a server-side proxy positioned at the edge of the WSN, which is the most common scenario. In addition, the server-side position is needed because the HC proxy needs to execute a TCP-UDP translation, needs to provide caching functionalities for the CoAP traffic and needs to provide multicast support for the constrained network. As suggested in [21], all these functionalities should be executed as close to the WSN as possible and this is only guaranteed by the server-side position. The scenario considered in this work is illustrated in Figure 6.

[5] http://tomcat.apache.org/
[6] http://couchdb.apache.org/

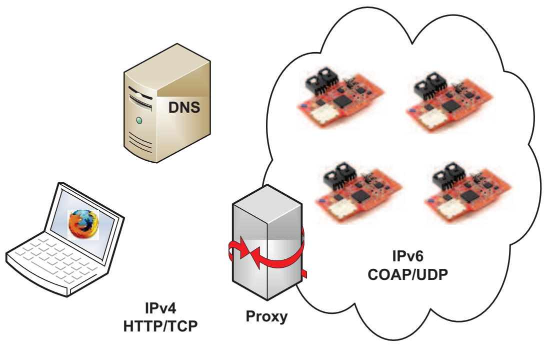


Figure 6. Position of the proxy at the edge of the WSN (server-side proxy).

Figure 6, considers the case in which the HTTP client is connected in an IPv4 network and wants to monitor resources in CoAP servers connected in an IPv6 network. In order to decrease the dependency between the HTTP client and the proxy and between the proxy and the CoAP servers, the system relies on the Domain Name System (DNS) which guarantees that the client does not need to know the IPv4 address of the proxy and the proxy does not need to know the IPv6 addresses of the CoAP servers.

According to [21], the cross-protocol resource access can be executed in two different ways:

- protocol-aware access: the HTTP client knows that the request is accessed by means of the CoAP protocol (e.g. the *coap:* scheme is embedded in the HTTP request);
- protocol-agnostic access: the HTTP client is not aware of the protocol translation and operates as it were accessing the request on an HTTP server.

The protocol-aware access does not require URI mapping and therefore simplifies the complexity of the overall HC mapping operation. However, this resource access method does not guarantee the complete transparency of the proxy because the proxy is aware that the resource is accessed via a different protocol. For this reason, the cross-protocol resource access considered in the proxy prototype implemented in this work is protocol-agnostic and guarantees complete transparency between client and server. The cross-protocol resource access operation is based on a homogeneous URI mapping. This implies that during the mapping operation, the authority and the path of the resource's URI do not change. The only translation undergone by the URI is in the scheme (*http:* translated into *coap:*).

An example of the message exchange executed during the end-to-end cross-protocol resource access is illustrated in Figure 7.

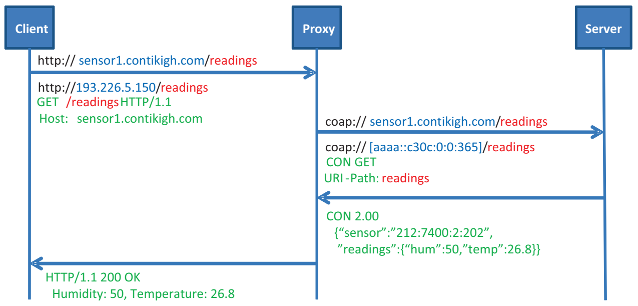

Client
Proxy
Server
http:// sensor1.contikigh.com/readings
http://193.226.5.150/readings
GET /readings HTTP/1.1
Host: sensor1.contikigh.com
coap:// sensor1.contikigh.com/readings
coap:// [aaaa::c30c:0:0:365]/readings
CON GET
URI-Path: readings
CON 2.00
{"sensor":"212:7400:2:202",
"readings":{"hum":50,"temp":26.8}}
HTTP/1.1 200 OK
Humidity: 50, Temperature: 26.8

Figure 7. Message exchange in the cross-protocol resource access.

Figure 7 shows the main operations executed from the moment the HTTP client sends the request to the moment the HTTP client receives the response with the requested resource. The HTTP client in an IPv4 networks wants to access a resource which is in a CoAP server in an IPv6 WSN through an HC proxy. The client generates the URI *http://sensor1.contikigh.com/readings*. In the URI, *http:* is the scheme, which indicates that the request is generated by using the HTTP protocol, *sensor1* is the name of the sensor which contains the request of interest (the CoAP server), *contikigh.com* is the network domain of the WSN and *readings* is the requested resource. In our case, *readings* indicates the sensor's temperature and humidity.

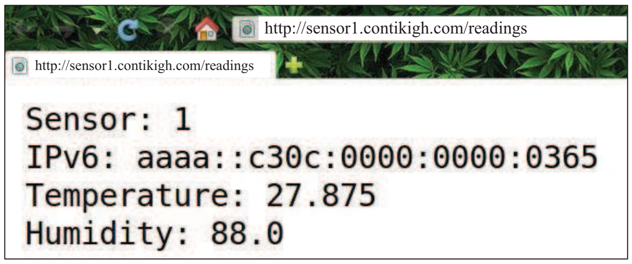
When the client sends out the request, the domain name in the URI needs to be resolved by a DNS server in order to get the IPv4 address of the HC proxy. As suggested in [20], we use the DNS Service Discovery (DNS-SD) [22], in order for the client to discover the proxy and obtain its IPv4 address. Once received the HC proxy's IPv4 address, the HTTP client sends the HC proxy the GET request. The GET request indicates the access method (*GET*), the requested resource (*/readings*), the protocol version (*HTTP/1.1*) and the CoAP server (*sensor1.contikigh.com*).

Once received the GET requests, the proxy generates the URI *coap://sensor1.contikigh.com/readings*. It is important to notice how the URI mapping operation only replaces the scheme (*http:* is replaced with *coap:*), while leaving the rest of the URI unchanged. The proxy contacts the DNS to resolve the name *sensor1.contikigh.com* and obtain the IPv6 address of the CoAP server to be accessed. At this point the proxy has all the information needed to generate the confirmable GET request (CON GET) and to send it to the CoAP server. Note that the CoAP server will consider the proxy as the client accessing the resource (the IPv6 address of the border router mote connected to the proxy).

Once the CoAP server has the requested resource, it generates a 2.00 response with the resource embedded into a JSON file. If the proxy successfully receives the JSON file, the proxy extracts the values and embeds them in a 200 OK response sent to the HTTP client.

The HC proxy has been implemented by modifying the *com.sun.net.httpserver* package included in the Java JDK 1.6. It provides a simple high-level HTTP server

Application Programming Interface (API) which has allowed us to make the modification needed to implement the HC proxy functionalities. The proxy has been deployed in the Zolertia motes testbed described in the previous section. At the moment we have only tested the proxy as a standalone module and not yet integrated in the Web application. For proof of concept purpose, we have implemented a simple Web page which is visualized when the proxy returns the HTTP response with the value of the requested resource. A snapshot of the simple resource visualization is illustrated in Figure 8.


http://sensor1.contikigh.com/readings
http://sensor1.contikigh.com/readings

```
Sensor: 1
IPv6: aaaa::c30c:0000:0000:0365
Temperature: 27.875
Humidity: 88.0
```

Figure 8. Snapshot of the page visualized by the HTTP client when accessing CoAP resources via the HC proxy.

It is important to underline that the HC proxy presented in this work is only a preliminary implementation and does not implement all the CoAP functionalities. In particular, it does not yet provide support for observations and multicast communications and it does not yet provide mapping for all the CoAP request and response codes [14]. The implementation of these functionalities is left as future work.

V. CONCLUSIONS

This work discussed on the integration of WSNs with the Web. This is being facilitated by the development of CoAP, an HTTP like IETF protocol providing WSNs with a RESTful architecture. We showed the prototype design and development of a Web platform for WSN monitoring. The platform integrates a REST/CoAP WSN with a REST/HTTP Web application. Since the client-server connection is not transparent because it relies on a Web server which knows how to access WSN resources on CoAP motes, we described the implementation of a transparent HTTP-CoAP proxy.

ACKNOWLEDGMENT

This work has been done in the scope of the ITEA project Interoperable Sensor Networks (ISN) in collaboration with the company Freemind [7]. The authors acknowledge INNOVIRIS Brussels[8] for their financial support.

REFERENCES

[1] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler, “sMAP – a Simple Measurement and Actuation Profile for Physical Information,” Proc. ACM Conference on Embedded Networked Sensor Systems (SenSys 10), 2010.

[2] M. Kovatsch, M. Weiss, and D. Guinard, “Embedding Internet Technology for Home Automation,” Proc. IEEE Conference on Emerging Technologies and Factory Automation (ETFA 10), 2010.

[3] S. Mayer, D. Guinard, and V. Trifa, “Facilitating the Integration and Interaction of Real-World Services for the Web of Things,” Proc. Urban Internet of Things – Towards Programmable Real-time Cities (UrbanIOT 10), 2010.

[4] D. Guinard, V. Trifa, and E. Wilde, “A Resource Oriented Architecture for the Web of Things,” Proc. Internet of Things 2010 International Conference (IoT 10), 2010.

[5] A. P. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby, and M. Zorzi, “Architecture and Protocols for the Internet of Things: A Case Study,” Proc. International Workshop on the Web of Things (WoT 10), 2010.

[6] S. Duquennoy, N. Wirström, N. Tsiftes, and A. Dunkels, “Leveraging IP for Sensor Network Deployment,” Proc. Extending the Internet to Low Power and Lossy Networks (IP+SN 11), 2011.

[7] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Beker, and C, Görg, “Implementation of CoAP and its Application in Transport Logistics,” Proc. Extending the Internet to Low Power and Lossy Networks (IP+SN 2010), *2011.*

[8] Z. Shelby, Z. Embedded Web Services. IEEE Wireless Communications, pp. 52-57, December 2010.

[9] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” Computer Networks, Elsevier, pp. 2787-2805, October 2010.

[10] N. Kushalnagar, G. Montenegro, and C. Schumacher, “IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals,” RFC 4919.

[11] J. P. Vasseur, and A. Dunkels, “Interconnecting Smart Objects with IP: The Next Internet,” Morgan Kaufmann, 2010.

[12] Z. Shelby, and C. Bormann, “6LoWPAN: The Wireless Embedded Internet,” Wiley, 2009.

[13] V. Trifa, S. Wieland, D. Guinard, and T. Bohnert, “Design and Implementation of a Gateway for Web-based Interaction and Management of Embedded Devices,” Proc. International Workshop on Sensor Network Engineering (IWSNE 09), 2009.

[14] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, “Constrained Application Protocol (CoAP),” Internet-Draft, draft-ietf-core-coap-04.

[15] K. Hartke, and Z. Shelby, “Observing Resources in CoAP,” Internet-Draft, draft-ietf-core-observe-01.

[16] Eggert, L., Congestion Control for the Constrained Application Protocol (CoAP). Internet-Draft, draft-eggert-core-congestion-control-01.

[17] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, Demo abstract: Software-based sensor node energy estimation. Proc. ACM Conference on Networked Embedded Sensor Systems (SenSys 07), 2007.

[18] P. Hurni, B. Nyffenegger, T. Braun, and A. Hergenroeder, “On the Accuracy of Software-Based Energy Estimation Techniques,” Proc. European Conference on Wireless Sensor Networks (EWSN 11), 2011.

[19] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-level sensor network simulation with COOJA,” Proc. IEEE Conference on Local Computer Networks (LCN 06), 2006, pp. 641-648.

[20] A. Castellani, and S. Loreto, “Best Practice to map HTTP to COAP and viceversa,” Internet-Draft, draft-castellani-core-http-coap-mapping-00.txt.

[21] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk, “Best practices for HTTP-CoAP mapping implementation,” *Internet-Draft*. draft-castellani-core-http-mapping-01.txt.

[22] S. Chesire, and M. Krochmal, “DNS-Based Service Discovery,” Internet-Draft, draft-cheshire-dnsext-dns-sd-10.txt.

[7] http://www.freemind-group.com/fmc2/
[8] http://www.irsib.irisnet.be/