Web Services for the Internet of Things through CoAP and EXI

Angelo P. Castellani*, Mattia Gheda*, Nicola Bui*†‡, Michele Rossi*†, Michele Zorzi*†‡
Email: {castellani, ghedamat, bui, rossi, zorzi}@dei.unipd.it
*Department of Information Engineering (DEI), University of Padova, Italy
†Consorzio Ferrara Ricerche (CFR) - 1, via Saragat, 44122 Ferrara, Italy
‡Patavina Technologies S.r.l. - 112, via Croce Rossa, 35129 Padova, Italy, http://www.patavinatech.com

***Abstract*—According to the Internet of Things (IoT) vision, everyday objects such as domestic appliances, actuators and embedded systems of any kind in the near future will be connected with each other and with the Internet. These will form a distributed network with sensing capabilities that will allow unprecedented market opportunities, spurring new services, including energy monitoring and control of homes, buildings, industrial processes and so forth.**

In this paper, we concentrate on the actual implementation of the communication technology, adopting the Representational State Transfer (REST) approach. REST only relies on the HTTP methods such as GET and POST. Embedded communication devices are addressed using Universal Resource Indicators (URI) and data is exchanged through standard XML.

We present our TinyOS design and implementation of two components that will play a fundamental role in the communication stack of REST-based devices. First, we focus on the Constrained Application Protocol (CoAP), which allows REST–based communications among applications residing in distributed and networked embedded systems. Second, we present our lightweight implementation of a EXI library: an efficient binary compressor for XML data files. Experimental results are provided in terms of compression, decoding (EXI) and access time (CoAP) performance.

I. Introduction and Related works

Starting from the eighties, worldwide communications popularity has always been increasing. The Internet paradigm has become a common denominator for networking applications. Nowadays, many information and communications services rely on IP technology: web-shopping, online-databases, social networks being three notable examples.

The actual trend for the ICT community revolves around two main scenarios: Smart Grids (SM) [1], [2] and Internet of Things (IoT) [3]. Even though these have been born from different needs, they have quite a few aspects in common and are characterized by similar challenges. Key objectives for both scenarios are: **seamless integration with IP**, system **scalability** and **interoperability**.

Both scenarios comprise millions of heterogeneous embedded devices featuring different technologies, each having been developed to satisfy a particular need. Just to name a few, wireless sensor and actuator networks (WS&ANs) [4] adopt low–power radios and simple CPUs, Radio Frequency Identifiers (RFIDs) [5] and Near Field Communication (NFC) [6] rely on little computational power and very short range radios, whereas wired embedded devices are equipped with Power Line Communication (PLC) [7] and ARM CPUs. The seamless and scalable interworking of such diverse technologies is crucial to the success of SM and IoT.

The Internet world is now in its mature age: the Internet Protocol (IP) is the most used network protocol along with the Hyper–Text Transmission Protocol (HTTP). However, only recently standardization bodies started to play a decisive role in interconnecting constrained devices with the Internet.

Recent research efforts explore the performance and the practical feasibility of a REST-based approach [8] on top of a 6LoWPAN stack [9] in WSNs. Other recent papers [10], [11], [12] have already highlighted the benefits of lightweight Web-based protocols accessing sensors resource data through Uniform Resource Identifiers (URI) and request methods (GET, PUT, POST, DELETE) [13], [8], [4]. However, Web Service-enabled WSNs still need a complete protocol stack definition for their direct integration in the Internet, proving that a Web-based system cam smoothly bridge information, objects and new services through WSNs.

XML has been acknowledged as the *de–facto* standard for data representation and exchange but his great flexibility comes at the prize of being very redundant; to alleviate this, many solutions (see [14] for a thorough review of XML compression techniques) are available: blind compressors, such as gzip, bzip2, DTDPPM [15] treat XML as plain text files; a second group of compressors (e.g., enhanced XMill [16], XMLPPM [17]) takes the XML document structure into account to achieve higher compression ratios. To the best of our knowledge, XBC and EXI have been the first working groups focusing on optimizing XML for constrained devices and the W3C [18] selected EXI as its standard.

In this paper, we illustrate some of the strengths of the Internet Engineering Task Force (IETF) approach. To this end, we detail the realization of simple, but powerful Web Services for IoT applications that use the Constrained Application Protocol (CoAP) [19], which is being defined in the Constrained RESTful Environment (CoRE) charter [20], and the eXtensible Markup Language (XML), which is combined with the Efficient XML Interchange (EXI) format [21], [22].

II. Constrained communication

WS&AN has been a hot research topic for nearly 10 years now and can be considered mature. This is testified, e.g., by

Fig. 1. Internet of Things in domestic environments: appliances and utilities can be monitored and controlled in near-realtime using smart embedded systems with IP connectivity.

the huge number of ZigBee devices being shipped, which has been doubling every year, hitting 20 millions in 2009 [23]. The main characteristics of the sensor devices in a WS&AN are: **low–power radio**, providing wireless communication capabilities, but very little bandwidth, **low–power CPU**, enabling substantial energy savings in the face of little computational capabilities, and **small footprint**, allowing easy installation, but posing design constraints.

These features enabled the development of very economic devices, easy to install and showing long lifetime on batteries, thus making WS&AN one of the principal actors in the IoT world. Fig. 1 shows how a home environment can be instrumented with fully–connected sensor nodes for smart metering and control of appliances.

While this figure shows a domestic environment, this concept can be extended to a larger number of scenarios, such as hospitals, offices, shopping malls, factories and even cities; adding the Internet dimension to these smart–environment will enable a whole set of new services and applications. The true IoT will be achieved when every interconnected device will be able to communicate using the same language(s).

Ideally, in the IoT each device will be represented as a resource providing its own description in terms of hardware capabilities and software interfaces as well as a description of the services that it provides. For instance, the refrigerator will provide information about its description, such as its main physical function, its retailer, its operating status, but will also provide access to smart–services such as "best before"–notifiers when products in it are passing that date or an "out-of-stock"–notifier, informing the user about which products are needed. The user, will thus be able to interact with the refrigerator in same way as with any website, by just connecting to the appropriate (IP) address and modifying parameters or activating services.

We stress that, this can only be achieved through a thorough standardization process and we think that the best candidate to drive this activity is the IETF, which is running many charters focused on constrained devices. The remainder of this paper is dedicated to the description of our implementations and performance evaluation of CoAP and EXI.

III. Constrained Application Protocol

CoAP [19] is currently being defined within the CoRE [20] working group of the IETF, which aims at providing a REST–based framework for resource–oriented applications optimized for constrained IP networks and devices, by designing a protocol set able to cope with limited packet sizes, low-energy devices and unreliable channels.

CoAP is based on the REST architectural style sharing the objectives and the intrinsic limitation listed above. It is designed for easy stateless mapping with HTTP, and for providing M2M interaction. HTTP compatibility is obtained by maintaining the same interaction model, using a subset of the HTTP methods.

Nodes supporting CoAP provide flexible services over any IP network using UDP, and they also provide a solid communication framework to connect sensor nodes to the Internet. Any HTTP client or server can interoperate with CoAP–Ready endpoints by simply installing a translation proxy between the two devices. This will not be a burden for the proxy, since these translation operations have been designed not to be time and computationally demanding. Also, CoAP features a transaction layer between the application protocol and UDP to provide basic reliability and session matching support[1].

We designed and developed a TinyOS CoAP implementation using the 6LoWPAN header-compression (HC) library from Harvan and Schoenwaelder (`6lowpan` [24]) which implements the first version of the HC (RFC 4944) [25]. `CoAPP` is a monolithic component providing client and server functionalities; it handles session data regardless of its type (either client or server), thus optimizing its memory usage. The actual implementation of this component can handle up to COAP_MAX_TRANSACTIONS transactions simultaneously, a value that can be chosen arbitrarily at build time by trading between memory occupation and flexibility.

The `CoAPClient` interface provides the `CoAPP` module with a TinyOS command to send any arbitrary request to a CoAP endpoint, and a TinyOS event to manage the response it gets back. Next, we show the TinyOS code of the interface:

```
interface CoAPClient {
  command coap_tid_t request (
    coap_absuri_t* absuri,
    coap_method_t method,
    coap_content_t* content,
    bool acked );

  event void response (
    coap_tid_t tid,
    coap_status_t status,
    coap_content_t* content ); }
```

The interface defines different custom data types to provide better readability and high-level operations. When a `request` command is issued the user must provide *i)* `absuri` describing endpoint host, port and URI of the requested resource,

[1]These functionalities are provided by the transport layer in the ISO/OSI stack.

ii) `method` specifying which method is used to access the requested resource, *iii)* `content` providing a pointer to the content to be attached to the request, if present, *iv)* `acked` to request a response message; the `request` command provides the user with the `coap_tid_t` internally assigned to the transaction. A `response` event is triggered when the related reply is received. This response contains *i)* a `tid` field identifying the transaction, *ii)* a `status` field containing the status code resulted after processing the request and *iii)* a pointer to the `content` piggybacked in the response.

The `CoAPServer` interface provides the `CoAPP` module with server capabilities: external components can use this interface to serve resources using a CoAP server. The `CoAPServer` and the the `CoAPClient` are complementary in the sense that commands issued using one interface trigger events managed by the other interface and viceversa.

```
interface CoAPServer {
  event void request (
    coap_rid_t rid,
    coap_absuri_t* uri,
    coap_method_t method,
    coap_content_t* content,
    bool toack );

  command error_t response (
    coap_rid_t rid,
    coap_status_t status,
    coap_content_t* content ); }
```

In order for a `request` to be properly processed, the following data is needed: *i)* a `rid` value internally assigned to univocally identify the request, *ii)* the `uri` of the requested resource, *iii)* `method` describing the access method, *iv)* a pointer to the `content`, if present, and *v)* a `toack` flag to signal if the client requested an ACK. The `response` command can be used by the serving module together with the following parameters, *i)* a `rid` to match the related request, *ii)* `status` value resulted from the processing of the request and *iii)* `content` pointer to data to be sent in the response.

The `CoAPServer` interface is characterized within the `CoAPP` module by a `port` parameter identifying on which UDP port the CoAP service has to be activated in the node.

The client/server architecture of the `CoAPP` module allows the implementation of lightweight Web services on constrained WS&AN nodes. Moreover, it makes it possible to implement M2M interactions, such as publish/subscribe, and to create multiple Web servers and services without burdening a constrained node system. As it will become evident in Section V from our experimental campaign, our design choices do not need heavy computational power, on the contrary the resulting CoAPP software proved to be fast and reliable in managing requests and responses.

IV. EXI COMPRESSOR

The Efficient XML Interchange (EXI) format is a compact XML representation, currently being standardized by the World Wide Web Consortium (W3C) [18]. It is designed to support high-performance XML applications for resource

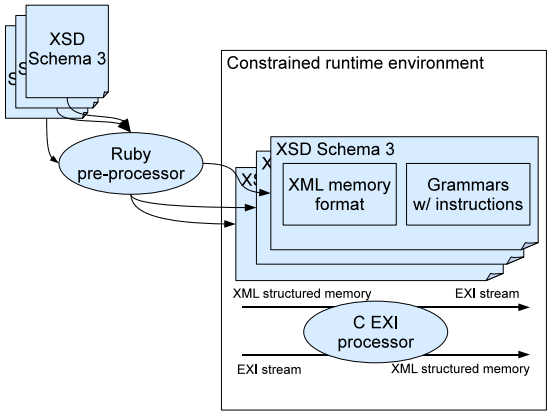

Fig. 2. Usage diagram for the EXI processor and pre-processor.

constrained environments, significantly reducing bandwidth requirements and improving encoding/decoding performance.

EXI compression exploits information about the document structure to internally generate small tags based upon the current XML schema, the current processing stage and the context. Also, tags data representation is optimized to be as compact as possible.

Although an efficient compression can be achieved from the XML schema, the standard defines other operating modes to produce a more-compact representation of the XML file using only partial or no XML schema information.

The encoded XML document results in an EXI stream, which represents the document in binary format where every data tag of the document is encoded using an event code. Event codes are binary tags that preserve their value only in their assigned position within the EXI stream.

Thus EXI implements event-based encoding: for efficient encoding, at any given point of an XML stream a set of grammars are used to understand which event is most likely to occur next. A set of grammars, representing the XML document structure, has to be produced before the actual EXI processing. The sequence of events describes the sequence of finite-state machines defined using each different grammar as transition function.

In an EXI stream every XML element is represented using a specific grammar; each grammar consists of a set of productions, defining the set of possible events in a specific state. EXI assigns an event–code (EC) to each production. The sequence of XML elements codified to ECs forms an EXI stream. When a new element of the EXI stream is parsed, a new grammar associated with the element is stacked upon the preceding one and the control is passed to a new automaton which is in charge of handling the new grammar, until the new element is completed and the control returns to the preceding routine.

We designed and implemented *libEXI*, an implementation of the EXI processor that has been specifically targeted for resource constrained MCUs (e.g., Texas Instrument MSP430). The design required to limit the number of features implemented. libEXI is a byte-aligned and schema-informed EXI

TABLE I
ROM/RAM UTILIZATION OF TINYOS COMPONENTS

Component	ROM	RAM
TinyOS core	1396	4
802.15.4 and ActiveMessage	9258	327
UDP/6LoWPAN	5804	1983
CoAP	2668	1801
RAI/RPI	1752	548
libEXI	7134	1016
Subscription	1580	522
Resources	12402	526
HW drivers	7338	160
CoAP web-services	3632	208
EXI handling	1432	158

processor, which encodes EXI streams using a preprocessed grammar-set (defining the XML schema in use) and a pre-processed C data structure set (representing a document compliant to the XML schema).

Hence, libEXI can translate EXI streams into a structured memory representation which can be stored and process by CPU-constrained devices. Bit-aligned encoding, even if very efficient, showed to be too complex to match our requirements.

As shown in Figure 2, our EXI library uses the results of a preprocessing phase: a Ruby pre-processor has to run on the XML schemas before libEXI can process EXI streams. This preprocessor extracts from any XML schema the set of grammars required to encode and decode EXI streams; in addition, it builds a set of C structures representing the XML document layers. The libEXI memory representation built by the pre-processor is an optimized translation of the XML document contents, needed for constrained devices to properly manage EXI streams.

The libEXI processor uses a grammar stack to encode/decode EXI streams. Grammars contain the list of events as well as the information of which grammar has to be stacked to handle the next part of the EXI stream, and which production will be the current grammar into when the control returns to it. Any new grammar piled in the grammar stack corresponds to a new execution of the encode/decode function call: in this way, there is a one-to-one mapping between the processor internal stack and the current grammars stack.

V. RESULTS

In this section we show some experimental results of the CoAP/EXI components that we implemented on telosb sensor nodes. In the evaluation campaign of these components, we have used our SENSEI [26] node implementation (see Section V of [8]) adding the new CoAP/EXI implementation to the SENSEI's protocol stack. Table I shows the RAM/ROM memory occupation of the various software components.

A. CoAP

As shown in Table I, the implementation of the CoAP component is smaller of that of our previous BinaryWS [8] implementation. Our previous implementation led to a higher memory (ROM) consumption as it accounted for a separate implementation of the interface toward each component on the nodes (resources such as leds, on-board sensors and actuators, etc.), also considering the specific requirements of their XML schemas and the hardware drivers needed for their physical access. The current component instead leverages uniform interfaces which are reused for all components, leading to a lightweight implementation of Web servers on sensor devices.

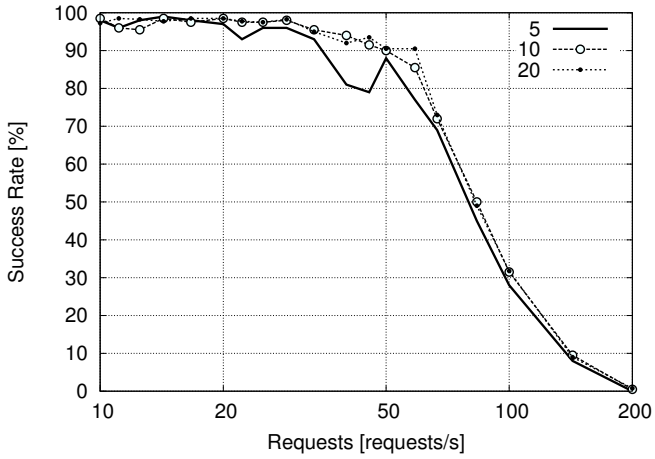

Fig. 3. CoAP success rate vs. requests rate.

As a first set of results, in order to prove that CoAP/EXI components do not negatively impact the performance of the nodes, we set up an experiment with up to 20 CoAP servers running on a single telosb node (*serving node*). A second telosb node (*client node*) was used to send requests to these servers at a rate that was kept constant during each experiment and varied across them so as to highlight the dependence on this parameters. The outcomes of this test are shown in Fig. 3 where we plot the CoAP request success probability (intended as the percentage of occurrences for which a request is successfully handled by the serving node) as a function of the request rate by the client node. The experiment has been run considering 5, 10 and 20 CoAP servers. Due to our efficient implementation of CoAP, the success probability only slightly depends on the number of servers running on the nodes. As expected, a very high (i.e., higher than 60 requests/second) request rate severely impacts the access performance but this is due to the inherent limitations of the currently used 6LoWPAN library [24]. Given that, we can conclude that our CoAP implementation scales well with the number of server instances without causing a major and noticeable decrease in the access performance.

B. EXI

In Table II we show the compression efficiency for EXI, also showing that of other compression schemes, i.e., Gzip, enhanced Xmill (Xwrt) [16] and XMLPPM [17]. The size of the Uncompressed XML document, which is taken as the reference document for the experiment, is expressed in bytes. For the compression schemes we show the size of the

TABLE II
XML COMPRESSION PERFORMANCE: SIZE OF THE COMPRESSED STREAM (BYTES) AND COMPRESSION RATIO (BETWEEN PARENTHESES)

Format	Schema-1	Schema-2	Schema-3
Uncompressed XML	591 bytes	242 bytes	229 bytes
Gzip	302 bytes (*0.51*)	206 bytes (*0.85*)	175 bytes (*0.76*)
enhanced Xmill (Xwrt) [16]	784 bytes (*1.33*)	431 bytes (*1.78*)	453 bytes (*1.97*)
XMLPPM [17]	262 bytes (*0.44*)	164 bytes (*0.68*)	128 bytes (*0.55*)
EXI w/o schema byte-aligned	298 bytes (*0.50*)	104 bytes (*0.43*)	99 bytes (*0.43*)
EXI w/o schema bit-aligned	237 bytes (*0.40*)	96 bytes (*0.40*)	83 bytes (*0.36*)
EXI w/ schema byte-aligned	58 bytes (*0.10*)	10 bytes (*0.04*)	41 bytes (*0.17*)
EXI w/ schema bit-aligned	27 bytes (*0.05*)	4 bytes (*0.02*)	26 bytes (*0.11*)

compressed documents (also in bytes) and their compression ratio (within parentheses in the table), defined as the ratio between the sizes of the compressed stream and that of the uncompressed XML. The XML document we picked for our experiments is described by simple schemas that are suitable for, e.g., environmental monitoring and binary actuation (i.e., the operations that we expect from a sensor node). As demonstrated by the experimental results in Table II, EXI compression is extremely efficient especially for schema-informed XML compression, leading to compressed files that are as small as just 4 bytes, thus achieving a compression efficiency of 50 times (the inverse of the compression ratio). We remark that this is particularly important for resource constrained sensor nodes as EXI dramatically reduces the amount of data that has to be transferred (most likely via radio transmission) through the network.

As a last remark, we note that the bit-aligned mode is the most convenient choice, however, its implementation is more complex on MCUs working in byte-aligned mode. Thus, implementors may want to use the byte-aligned mode even though it provides inferior results.

As a third set of experiments, our EXI implementation has been extensively tested on a regular desktop PC and compared against EXIficient [27], a well-known and freely available Java implementation of EXI. The design criteria of EXIficient are very different from those of libEXI: Java was chosen due to its portability and all EXI features were implemented. However, the resulting implementation is not optimized for energy constrained devices and, as we show shortly, its performance is not consistent across repeated applications to the same document.

The steady-state XML throughput (number of processed XML elements per second) has been measured for libEXI and EXIficient. The latter can output an EXI stream at the maximum rate of about 0.9 millions of XML elements per second, whereas libEXI outputs about 6 millions XML elements per second.

Next, we look at the *XML processing time*, which is the time taken to compress an input XML file (and is inversely proportional to the XML throughput). Notably, the processing time of EXIficient decreases across repeated applications to the same XML file, showing a (non-negligible) transient phase at the beginning of which its performance is much worse

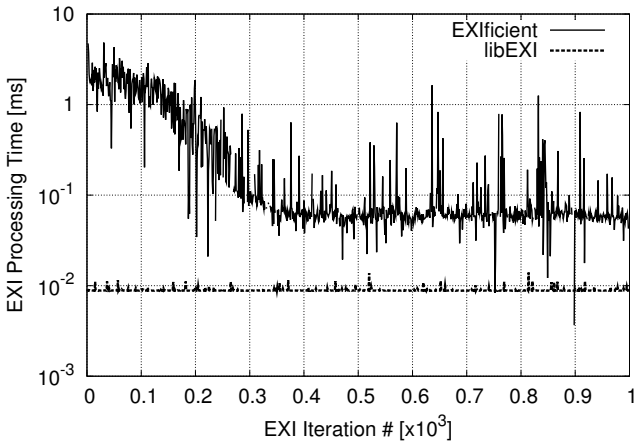

Fig. 4. EXI processing time against iteration number.

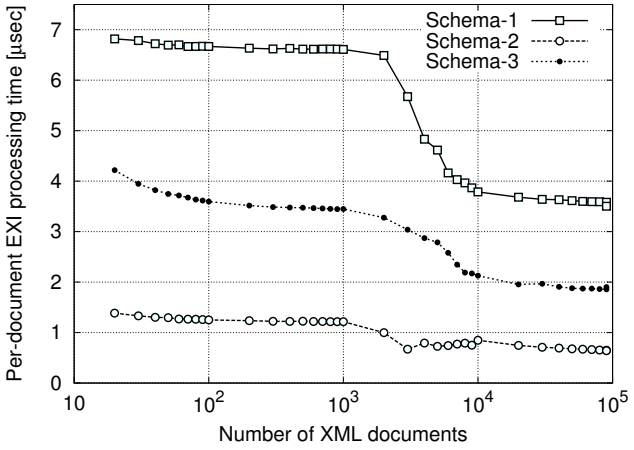

Fig. 5. libEXI per-document processing time against number of XML documents.

(up to 50 times) than that in steady-state. This is shown in Fig. 4, where we plot the average processing time for libEXI and EXIficient for a reference document containing 50 XML elements. As can be seen from the same figure, the processing time of libEXI is nearly constant across repeated compressions of the same XML file.

Finally, in Fig. 5 we shown the average libEXI processing

time for XML documents containing a single XML element. These small-sized documents are relevant to the IoT as they can represent queries to, e.g., acquire the readings of specific sensor nodes. Obtaining the compressed EXI stream of these documents is very fast (less than 10 micro-seconds), which is desirable for, e.g., a proxy connecting devices within the Internet domain with the IoT. In fact, one of the main functions of this proxy will be that of performing conversions between EXI and XML. EXI will be the preferred format for the constrained devices residing within the IoT, whereas XML will be used by the more powerful computers located within the Internet.

VI. Conclusions

This paper described and evaluated the performance of a Web service architecture for constrained devices. The original contributions of this work are: *i)* the use of standard (or under standardization) protocols to connect the constrained world of WS&ANs to the Internet, *ii)* a TinyOS implementation of this framework which is both small in terms of memory occupation and computationally lightweight, *iii)* an experimental performance evaluation campaign proving the feasibility of the framework and the efficiency of our implementation of it.

To these authors, the future of the Internet of Things relies much on the use of standard solutions for extending the Internet and the Web paradigms into the world of resource constrained sensing and actuating devices. Efficient implementations of CoAP as well as of the EXI compression scheme are just two of the blocks needed for the realization of this vision. We are currently working to build a complete communication stack realized using IETF standards, such as 6LoWPAN and RPL (Routing for Low Power and Lossy Networks) [28]. Further steps will be those of porting this architecture to other constrained devices, such as, e.g., RFID and PLC networks.

Acknowledgment

This work has been supported in part by the FP7 EU projects "SENSEI" G.A. no. 215923, http://www.ict-sensei.org, "SWAP" G.A. no. 251557, "IoT-A" G.A. no. 257521 and by the CaRiPaRo Foundation, Italy, within the WISE-WAI project, http://cariparo.dei.unipd.it.

References

[1] S. M. Amin and B. Wollenberg, "Toward a smart grid: power delivery for the 21st century," IEEE Power and Energy Magazine, vol. 3, no. 5, pp. 34–41, Sept-Oct 2005.

[2] F. Li, W. Qiao, H. Sun, H. Wan, J. Wang, Y. Xia, Z. Xu, and P. Zhang, "Smart Transmission Grid: Vision and Framework," IEEE Transactions on Smart Grid, vol. 1, no. 2, pp. 168–177, Sep. 2010.

[3] J. P. Conti, "The Internet of Things," Communications Engineer, vol. 4, no. 6, pp. 20–25, Dec-Jan 2006.

[4] N. Bressan, L. Bazzaco, N. Bui, P. Casari, L. Vangelista, and M. Zorzi, "The Deployment of a Smart Monitoring System Using Wireless Sensor and Actuator Networks," in IEEE SmartGridComm, Gaithersburg, WA, USA, Oct. 2010.

[5] S. Roy, V. Jandhyala, J. Smith, D. Wetherall, B. Otis, R. Chakraborty, M. Buettner, D. Yeager, Y.-C. Ko, and A. Sample, "RFID: From Supply Chains to Sensor Nets," Proceedings of the IEEE, vol. 98, no. 9, pp. 1583–1592, Sep. 2010.

[6] J. Fischer, "NFC in cell phones: The new paradigm for an interactive world," IEEE Communications Magazine, vol. 47, no. 6, pp. 22–28, Jun. 2009.

[7] Y.-J. Lin, H. Latchman, M. Lee, and S. Katar, "A power line communication network infrastructure for the smart home," IEEE Wireless Communications, vol. 9, no. 6, pp. 104–111, Dec. 2002.

[8] 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," in IEEE WoT, Mannheim, Germany, Mar-Apr 2010.

[9] Z. Shelby and C. Borman, 6LoWPAN: The Wireless Embedded Internet. Wiley, Nov. 2009.

[10] B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in Proceedings of ACM SenSys, Raleigh, NC, Nov. 2008.

[11] D. Yazar and A. Dunkels, "Efficient Application Integration in IP-Based Sensor Networks for Emerging Energy Management Systems," in Proceedings of ACM Buildsys, Berkeley, CA, US, Nov. 3 2009.

[12] L. Schor, P. Sommer, and R. Wattenhofer, "Towards a Zero-Configuration Wireless Sensor Network Architecture for Smart Buildings," in Proceedings of ACM Buildsys, Berkeley, CA, US, Nov. 3 2009.

[13] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim, "TinyREST - a protocol for integrating sensor networks into the internet," in Proceedings of REALWSN, Stockholm, Sweden, Jun. 2005.

[14] S. Sakr, "XML compression techniques: A survey and comparison," Journal of Computer and System Sciences, vol. 75, pp. 303–322, 2009.

[15] "DTDPPM Compressor." [Online]. Available: http://xmlppm.sourceforge.net/dtdppm/

[16] "Xwrt: enhanced XMill compressor." [Online]. Available: http://sourceforge.net/projects/xmill/

[17] J. Cheney, "Compressing XML with Multiplexed Hierarchical PPM Models." [Online]. Available: http://xmlppm.sourceforge.net/paper/paper.html

[18] World Wide Web Consortium (W3C), "XML Technology." [Online]. Available: http://www.w3.org/standards/xml/

[19] Z. Shelby, B. Frank, and D. Sturek, "Constrained Application Protocol (CoAP)," IETF Internet Draft draft-ietf-core-coap-02, 2010. [Online]. Available: http://datatracker.ietf.org/doc/draft-ietf-core-coap/

[20] C. Bormann and C. Jennings, "Constrained RESTful Environments (core)," IETF Working Group, 2010. [Online]. Available: http://datatracker.ietf.org/wg/core/

[21] J. Schneider and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0," W3C Working Draft, 2008. [Online]. Available: http://www.w3.org/TR/2008/WD-exi-20080919

[22] D. Peintner, H. Kosch, and J. Heuer, "Efficient XML Interchange for rich internet applications," in IEEE ICME, New York, NY, USA, Aug. 2009.

[23] T. Wolverton, "ZigBee radio chips could allow remote use of home electronics," Los Angeles Times, Apr. 2010. [Online]. Available: http://articles.latimes.com/2010/apr/01/business/la-fi-zigbee1-2010apr01

[24] M. Harvan and J. Schoenwaelder, "TinyOS Motes on the Internet: IPv6 over 802.15.4 (6lowpan)," PIK - Praxis der Informations - verarbeitung und Kommunikation, vol. 31, pp. 244–251, 2008.

[25] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," IETF Request For Comments, Sep. 2007.

[26] SENSEI FP7 Project, "Integrating the Physical with the Digital World of the Network of the Future." [Online]. Available: http://www.ict-sensei.org/

[27] "EXIficient: an Open Source Implementation of the W3C Efficient XML Interchange (EXI) Format Specification." [Online]. Available: http://exificient.sourceforge.net/

[28] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks," IETF Internet Draft draft-ietf-roll-rpl-12, 2010. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-roll-rpl/