Towards an End-to-End Web Experience in the Internet of Things

A Position Paper for the W3C Workshop on the Web of Things

Matthias Kovatsch
Institute for Pervasive Computing
Department of Computer Science
ETH Zurich, Switzerland
Email: kovatsch@inf.ethz.ch

***Abstract*—The idea of the Web of Things (WoT) is to adopt the ubiquitous protocols and well-known patterns of the World Wide Web to build an Internet of Things (IoT) that is interoperable, easy to program, and intuitive to use. The WoT architecture must follow the end-to-end principle to get out of the current fragmentation of IoT devices, and hence must eliminate application-level gateways that simply hide vertical silos. For such an end-to-end Web experience, we also need to adopt solutions for resource-constrained devices, as they are expected to constitute the majority of the IoT. The Constrained Application Protocol (CoAP) is a promising candidate, since it implements the REST architectural style, while satisfying the constraints of low-power nodes and networks. We have shown in previous work that CoAP enables Web-like programming models for resource-constrained devices and that users prefer the Web-like interaction for networked embedded systems. While CoAP is already an open Internet standard, it is not a part of the Web ecosystem yet. We propose to push further and make it part of the set of contemporary specifications implemented by Web browsers and provide standardized APIs to make CoAP-based devices first-class citizens of the Web.**

I. Introduction

The Web of Things (WoT) initiative aims at out-of-the-box interoperability and reusability of available systems within the Internet of Things (IoT) [3]. It advocates the use the ubiquitous Hypertext Transfer Protocol (HTTP) to connect sensor and actuator devices directly to the Web, which already provides a rich ecosystem of services. The adoption of well-known patterns from the Web further eases the development process and allows for intuitive interaction. With the help of WoT platforms such as *ThingSpeak*, the *COMPOSE API* [12], or *WoTKit* [1], even end-users can create their own IoT applications in the form of physical Web mashups.

Yet not all devices are powerful enough to support HTTP and WebSocket at a level that is required by the applications. Since Moore's Law will primarily help to minimize unit cost, power consumption, and device dimensions, it is expected that the majority of IoT devices will be resource-constrained. Constrained nodes and networks require expert knowledge and optimized protocols that are usually custom-tailored to the application. Thus, proprietary solutions and vertical silos have been predominant in this domain, which is now fragmented with a plethora of communication and service technologies.

II. Closing the Gap

Based on the success of embedded low-power IP stacks and the standardization of 6LoWPAN [5], the IETF chartered a working group to design a new Web protocol to close the gap between resource-constrained IoT devices and the powerful Web ecosystem: the *Constrained Application Protocol* (CoAP) [15]. It uses a compact binary format and runs over UDP (or DTLS when security is enabled), which also enables multicast communication. The well-known methods `GET`, `PUT`, `POST`, and `DELETE` as well as response codes that are defined in accordance to the HTTP specification provide RESTful interaction. CoAP resources are addressable by URIs, and common Internet Media Types represent resource state. Yet CoAP offers features that go beyond what HTTP provides and hence make it a better fit for the IoT:

1) Resources are *observable*, that is, extra responses continuously push state changes to all registered clients [4].
2) The extension to a request/multiple-response pattern also enables RESTful *group communication* where multiple servers respond to a request that is sent to an IP multicast address [13].
3) CoAP includes a machine-to-machine *discovery mechanism* to find matching resources based on Web Linking [11], [14]. It uses either multicast or resource directories [16] where devices register on start-up.
4) *Application-layer fragmentation* allows blockwise on-the-fly processing of messages that would otherwise exceed the maximum transmission unit (MTU) of 1280 bytes or the potentially even smaller buffers of highly resource-constrained devices [2].
5) Finally, CoAP can also support *alternative transports* that are not IP-based such as Short Message Service (SMS) or Unstructured Supplementary Service Data (USSD), while maintaining interoperability at the application layer [17].

In our work, we further close the gap between recourse-constrained devices and the Web by providing adequate programming models and tools. In the following, we give a brief overview over selected projects and summarize our experience with the WoT approach.

A. Programming Models

With *Erbium (Er)* [8], we provide the default CoAP implementation for Contiki, a leading embedded operating system for research, prototyping, and first IoT products. While a small memory footprint is one of the main design goals, we decided to trade efficiency for usability and provide an intuitive API for developers. All platform and networking details are hidden behind a resource handler abstraction similar to REST frameworks in unconstrained environments:

```
RESOURCE(res_leds,
    "title=\"LEDs\"", // Link-Format
    res_get_handler,
    NULL, // 4.05 Method not allowed
    res_put_handler,
    NULL); // 4.05 Method not allowed

static void
res_get_handler(void* request, void* response, uint8_t *
     buffer, uint16_t preferred_size, int32_t *offset)
{
  // GET handler implementation
}

static void
res_put_handler(void* request, void* response, uint8_t *
     buffer, uint16_t preferred_size, int32_t *offset)
{
  // PUT handler implementation
}
```

Since version er-coap-18, each resource can be implemented in its own C module and then reused by binding it to a URI path in the server configuration. Erbium (Er) focuses on the server side to implement the *thin server architecture* for the IoT. The CoAP server is supposed to be a mere wrapper for the hardware by providing a RESTful API to its (usually fixed) functionality. This way, application development can be separated from the embedded domain by moving the application logic to the cloud [10].

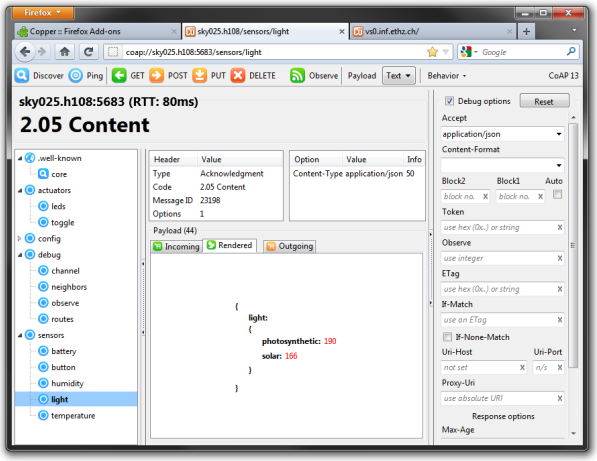
In our *Actinium (Ac)* project [9], we show Web-like scripting can be directly combined with resource-constrained devices. For this, we provide a runtime container that enables RESTful installation, configuration, and execution of JavaScript apps. These apps are modeled as REST resources themselves to easily provide results for other apps. To communicate with IoT devices, we define the *CoapRequest object API* similar to the XmlHttpRequest:

```
var req = new CoapRequest();
// define the callback for notifications
req.onprogress = function() {
// unlike XHR,  only contains payload of last message
    update(this.responseText);
  };
// request is DONE, i.e., the observe relationship ended
req.onload = function() {
    app.dump("Observing terminated"); // to console
  };
req.open("GET", "coap://mote1.example.com/sensors/pir",
         true /*asynchronous*/, true /*confirmable*/);
req.setRequestHeader("Observe", 0);
req.send(); // non-blocking
```

This allows for the creation of Web mashups that directly include IoT devices. Application-level gateways that have to be updated with every new device functionality become obsolete, which reduces complexity and establishes the end-to-end principle for the WoT.

B. User Interaction

For a true Web experience with resource-constrained IoT devices, we implemented CoAP support in the Web browser. With our *Copper (Cu)* add-on for Mozilla Firefox [6], users can simply enter a `coap` URI into the address bar to interact with IoT devices. Many Web patterns such as browsing, bookmarking, and linking can be used with CoAP resources as well. The add-on provides a user interface similar to other REST clients, since IoT representations usually do not provide visualization information (cf. HTML):



A user study with 48 developers from industry and academia shows that 77% prefer a Web browser integration over a standalone CoAP client for interaction [7]. Our browser add-on had a leading market share of 41%, followed by the libcoap console client with 17%. Copper (Cu) is a preferred debugging and testing tool for IoT developers, just like the Web browser is for Web developers. What most participants were missing is scripting support for our Web-browser-based client.

III. Conclusions

Our projects show, how resource-constrained IoT devices can directly become part of the Web of Things using the Constrained Application Protocol. CoAP is an open Internet standard (Proposed Standard since July 2013) and has become part of several service-level specifications such as OMA Lightweight M2M, the IPSO Profile, or OneM2M. While it already plays a significant role in the emerging IoT, it is not sufficiently aligned with the WoT initiative yet, though. Yet interoperability at the application layer and usability for both developers and users as provided by Web technology is key to the success of the IoT.

CoAP must also become an integral part of the technology used in the World Wide Web. In particular, we see native support by Web browsers as a key step in the realization of the Web of Things idea. With the protocol extensions used for real-time communication (WebRTC), fundamental implementations such as DTLS are already part of Web browsers. By providing adequate APIs such as our proposed CoapRequest object—or preferably a unifying object API for REST requests—Web applications can directly interact with IoT devices and provide intuitive user interfaces.

REFERENCES

[1] M. Blackstock and R. Lea. IoT Mashups with the WoTKit. In *Proc. IoT*, Wuxi, China, 2012.

[2] C. Bormann and Z. Shelby. Blockwise transfers in CoAP. draft-ietf-core-block-14, 2013.

[3] D. Guinard, V. Trifa, and E. Wilde. A Resource Oriented Architecture for the Web of Things. In *Proc. IoT*, Tokyo, Japan, 2010.

[4] K. Hartke. Observing Resources in CoAP. draft-ietf-core-observe-12, 2014.

[5] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, 2011.

[6] M. Kovatsch. Demo Abstract: Human–CoAP Interaction with Copper. In *Proc. DCOSS*, Barcelona, Spain, 2011.

[7] M. Kovatsch. Coap for the web of things: From tiny resource-constrained devices to the web browser. In *Proc. WoT*, Zurich, Switzerland, 2013.

[8] M. Kovatsch, S. Duquennoy, and A. Dunkels. A Low-Power CoAP for Contiki. In *Proc. MASS*, Valencia, Spain, 2011.

[9] M. Kovatsch, M. Lanter, and S. Duquennoy. Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications. In *Proc. IoT*, Wuxi, China, 2012.

[10] M. Kovatsch, S. Mayer, and B. Ostermaier. Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things. In *Proc. IMIS*, Palermo, Italy, 2012.

[11] M. Nottingham. Web Linking. RFC 5988, 2010.

[12] J. L. Pérez, A. Villalba, D. Carrera, I. Larizgoitia, and V. Trifa. The compose api for the internet of things. In *Proc. WWW Companion*, Seoul, Korea, 2014.

[13] A. Rahman and E. Dijk. Group Communication for CoAP. draft-ietf-core-groupcomm-18, 2013.

[14] Z. Shelby. Constrained RESTful Environments (CoRE) Link Format. RFC 6690, 2012.

[15] Z. Shelby, K. Hartke, and C. Bormann. Constrained Application Protocol (CoAP). draft-ietf-core-coap-18, 2013.

[16] Z. Shelby, S. Krco, and C. Borman. CoRE Resource Directory. draft-ietf-core-resource-directory-01, 2013.

[17] B. Silverajan and T. Savolainen. CoAP Communication with Alternative Transports. draft-silverajan-core-coap-alternative-transports-04, 2014.