Actinium (Ac) and the Thin Server Architecture

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

***Abstract*—Our paper *Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications* presents an architecture to make programming of Internet of Things applications significantly easier. Traditional programming models, stemming from networked embedded systems and wireless sensor network research, require developers to be knowledgeable in various technical domains, from low-power networking, over embedded operating systems, to distributed algorithms. With Actinium, we bring Web-like scripting to wireless sensor and actuator nodes and comparable resource-constrained devices. Applications are split into two components: *Thin servers* that provide the hardware functionality of IoT devices through a low-level RESTful interface and *scripted apps* that implement the application logic and run in the Cloud. Using our Actinium (Ac) app-server, Erbium (Er) REST engine for Contiki, Californium (Cf) CoAP framework, and Copper (Cu) CoAP user-agent, we demonstrate how to create IoT application by simply mashing up devices, modular apps, and other RESTful Web services. All these building blocks are also publicly available.**

I. Introduction

The Internet of Things reflects the vision of interconnecting the virtual and the physical world. For this, physical artifacts endowed with sensing, actuation, computing, and communication capabilities become the link between computer networks and the real world. With the rising presence of light-weight TCP/IP suites [2], [5], these ‘things’ are becoming directly accessible through the Internet. Creating IoT applications with such systems remains, however, unnecessarily difficult. Developers have to cope with different operating systems, deal with platform-dependent issues, and be aware of low-power networking.

To this end, the Web of Things (WoT) initiative proposes to use simple, well-defined RESTful interfaces [11] to access devices. Currently available WoT solutions usually require application-level gateways that put the RESTful interfaces on top of various low-power technologies such as Bluetooth or ZigBee [3]. We push this idea one step further by moving the Web servers directly onto low-end devices with only about 10kB of RAM and 100kB of ROM [7]. For this, we use CoAP [10] which has two key advantages over HTTP: First, the UDP-based protocol makes it feasible to run a Web server directly on devices with a performance comparable to native WSN protocols. Second, CoAP supports native push notifications, which are a central element for real-time updates.

Our software design is based on the separation of device firmware and application-specific logic, which we call the *Thin Server Architecture* [8]. Devices only expose their basic

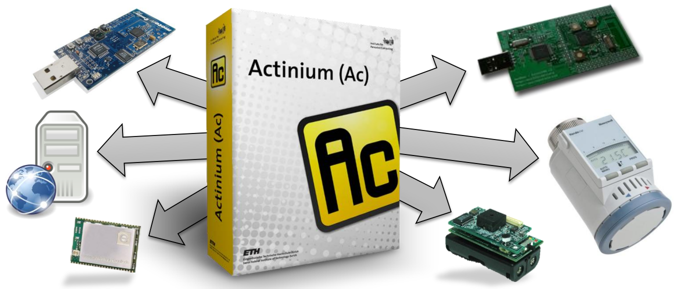


Fig. 1. Actinium mashes up resources directly from IoT devices as well as other Ac scripts or remote Web services. Our novel runtime container fully complies with the REST architectural style and even performs dynamic installation, updates, monitoring, and removal of scripted applications through RESTful interaction.

features through RESTful interfaces so that the firmware only handles sensing and actuation. The application logic runs in the Cloud, i.e., a local app-server or a remote service provider, both connected through IP. This has two key benefits: Application development is decoupled from the embedded domain and the infrastructure becomes usable by multiple concurrent applications.

II. Actinium (Ac)

Actinium is our novel RESTful runtime container for scripted IoT applications. It takes the experience from scripting mashups in the Web browser and transfers it to a standalone app-server for machine-to-machine scenarios. Each running script is modelled as a resource and the dynamic installation, update, and removal of scripts is done through a RESTful API. The internal scripting API adds the possibility for scripts to also export internal data through such an interface. Thus, scripts can not only mash up devices, but also other scripts. More complex applications can be built by reusing other modules. This example shows how to provide two sub-resources:

```
var threshold = 0;
// a sub-resource "/config"
var sub1 = new AppResource("config");
app.root.add(sub1);
// that accepts PUT requests
sub1.onput = function(request) {
// to configure the threshold
    threshold = request.payloadText;
  };
// a sub-resource "/occupancy"
var sub2 = new AppResource("occupancy");
app.root.add(sub2);
sub2.onget = function(request) {
```

```
// that returns true or false depending on a given value
    request.respond(2.05, value > threshold ?
      "true" : "false");
  };
```

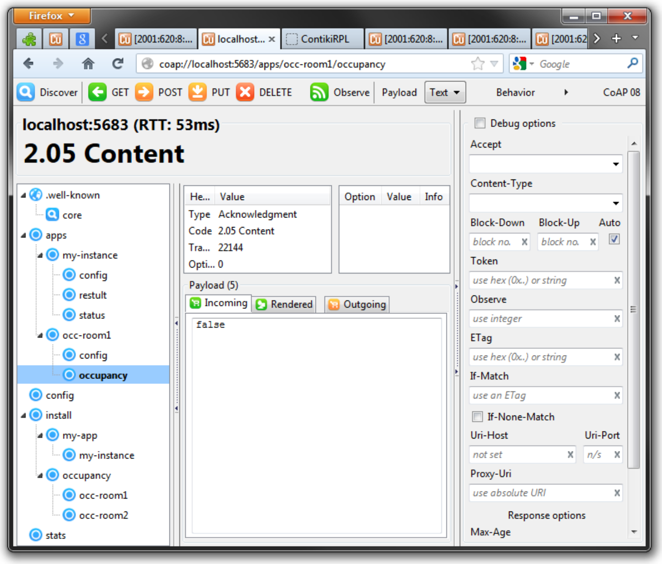
To directly access devices running CoAP, Actinium offers the *CoAPRequest object API*, which behaves similar to AJAX's XMLHttpRequest object. Yet, we also cover CoAP's new concepts such as observing [4]:

```
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
```

III. Other Building Blocks

Apps hosted by Actinium build on thin servers, which provide access to device functionality. Those are implemented with the Erbium REST Engine for Contiki [7]. All device resources are discoverable through `/.well-known/core`, which provides a listing in the CoRE Link Format [9]. Eventful resources such as those exposing the button are observable and large resources support blockwise transfers [1].

To install new scripts, change the configuration, and also test apps as well as devices, we use our Copper (Cu) browser add-on for Firefox [6] (Figure 2). It allows for direct interaction with networked embedded devices. It is a tool for configuration and assessment of correct system behavior like the Web browser is for classic RESTful Web services.



coap://localhost:5683/apps/occ-room1/occupancy

localhost:5683 (RTT: 53ms)

2.05 Content

He...	Value
Type	Acknowledgment
Code	2.05 Content
Tra...	22144
Opti...	0

Option	Value	Info

Payload (5)

```
false
```

Debug options
Accept:
Content-Type:
Block-Down: block no. Block-Up: block no. Auto [✓]
Token: use hex (0x..) or string
Observe: use integer
ETag: use hex (0x..) or string
If-Match: use an ETag
If-None-Match
Uri-Host: not set Uri-Port: n/s
Proxy-Uri: use absolute URI
Max-Age

Fig. 2. The tree on the left shows the resources of a running Actinium app-server. The current URI is the occupancy sub-resource from the example.

Finally, our Californium (Cf) CoAP framework [8] provides a CoAP-HTTP cross proxy that can connect normal RESTful Web services directly with devices.

IV. Demonstration Setup

The purpose of our demonstration is to show how embedded Web technology eases the creation of IoT applications. For this, we have the following setup (cf. Figure 3):

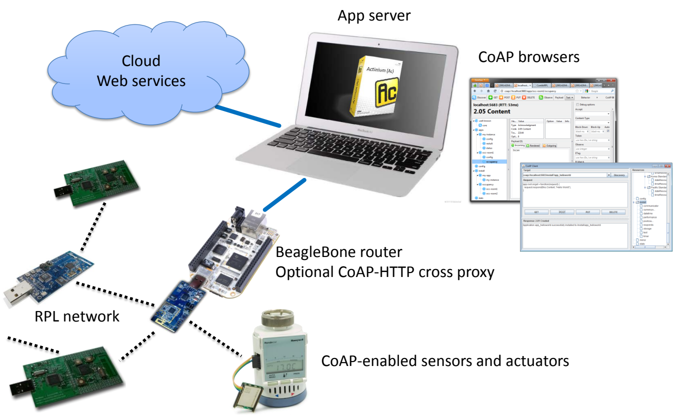


Fig. 3. Our demonstrator includes an Actinium app-server, several thin servers running on wireless sensor motes, an embedded 6LoWPAN border router, and tools for user interaction and testing.

The central component is Actinium, which runs on a laptop. We demonstrate how applications can be installed, configured, executed, and removed through the RESTful interface. As the latter is based on CoAP, we also demonstrate Copper and the Californium GUI client to interact with CoAP resources. To show the concept of IoT device mashups and our CoAPRequest object API, we provide multiple wireless sensor nodes running Erbium. The wireless sensor nodes form a RPL network and are connected through an embedded 6LoWPAN border router implemented on a BeagleBone.

References

[1] C. Bormann and Z. Shelby. Blockwise transfers in CoAP. draft-ietf-core-block-09, 2012.
[2] A. Dunkels. Full TCP/IP for 8-bit Architectures. In *Proc. MobiSys*, San Francisco, CA, USA, 2003.
[3] D. Guinard. *A Web of Things Application Architecture - Integrating the Real-World into the Web*. PhD th., ETH Zurich, 2011.
[4] K. Hartke. Observing Resources in CoAP. draft-ietf-core-observe-06, 2012.
[5] J. Hui and D. Culler. IP is Dead, Long Live IP for Wireless Sensor Networks. In *Proc. SenSys*, Raleigh, NC, USA, 2008.
[6] M. Kovatsch. Demo Abstract: Human–CoAP Interaction with Copper. In *Proc. DCOSS*, Barcelona, Spain, 2011.
[7] M. Kovatsch, S. Duquennoy, and A. Dunkels. A Low-Power CoAP for Contiki. In *Proc. MASS*, Valencia, Spain, 2011.
[8] 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.
[9] Z. Shelby. Constrained RESTful Environments (CoRE) Link Format. RFC6690, 2012.
[10] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP). draft-ietf-core-coap-11, 2012.
[11] E. Wilde. Putting Things to REST. Technical Report 2007-015, School of Information, UC Berkeley, Berkeley, CA, USA, 2007.