HSML tutorial for CoAP

PWP course work

by Mikko Majanen

mikko.majanen@vtt.fi

Spring 2017

1. Introduction

This work introduces Hypertext Sensor Markup Language (HSML) [1] for Constrained Application Protocol (CoAP) [2]. CoAP is a web transfer protocol designed for machine-to-machine (M2M) applications running in constrained nodes and networks. CoAP can be easily converted to HTTP since they have many similarities such as usage of Representational State Transfer (REST) architecture with request/response interaction model, usage of URIs, Internet media types, etc. The main difference is that CoAP runs over UDP, while HTTP uses TCP. This enables CoAP to support multicasting, which cannot be done with TCP.

Until recently, CoAP has lacked a real hypermedia format. CoRE Link Format [3] defines the state transitions, i.e., the links between resources and their relations, but does not contain the data part. This is discussed more on the next section.

In this work, we concentrate on HSML. It is a data format targeted for both state transitions and data. It is to be noted that the specification work of HSML is still in progress. This work bases on the version 01 of the corresponding Internet Draft document.

The rest of the work is organized as follows: Section 2 shortly summarizes the current work for hypermedia formats for CoAP. Section 3 introduces basics of HSML based on [1] and Section 4 documents the HSML implementation based on the CoAPthon [4] library. Section 5 concludes the work with future work items.

2. Related work

The CoRE Link Format [3] extends the HTTP Link Header format [5] to describe the link format to be used in finding out the URIs (links) to the resources hosted by the server. The CoRE Link Format is carried in the packet payload, not in the header as HTTP Link Header. Each server has a special ”/.well-known/core” URI that serves as an entry point for requesting the list of resource links. [3] also defines some CoRE-specific Link Attributes, namely Resource Type ’rt’, Interface Description ’if’ and Maximum Size Estimate ’sz’ attributes. Resource Type defines an application-specific semantic type to the resource, for example ’outdoor-temperature’. The type could be also a URI to an ontology. Interface Description provides a name or URI to a specific interface definition that reveals how to interact with the target resource. For the different type of sensors, the Interface Description could be, e.g., ’sensor’. The ’sz’ attribute defines the maximum size of the resource

representation that is returned as a response to the GET operation. These attributes can be used in uri-queries to filter the returned resources.

As already told in the introduction, the CoRE Link Format can be used for the links, but it does not contain the data part of the resources. For the data part, for example, Sensor Measurement Lists (SenML) [6] could be used. HSML combines SenML and CoRE Link Format to manage both links and data.

3. HSML

HSML data model consists of collections containing links and items. An instance of a collection is a resource. Links are standard web links based on either CoRE Link Format or HTTP Link Header formats that can be extended with actions and monitors or any referencing extension documents. Items are data elements, and an instance of an item is a resource. Resources are identified by a URI. In this section, the basic features of the HSML are shortly summarized. For more details, see [1]. The example codes presented are also from [1].

3.1 Transfer model

HSML media type uses simple CRUD+O transfer model (Create, Retrieve, Update, Delete, Observe) for interacting with the representations.

CREATE creates an instance of a resource as specified by the request's payload. CREATE uses POST method in CoAP. The location of the created resource is returned in the response message.

RETRIEVE obtains a representation of the selected resource. It uses GET method in CoAP.

UPDATE replaces (part of) the representation of the selected resource. It uses PUT or PATCH in CoAP. PATCH was not part of the original CoAP standard; it has been specified recently in [7].

DELETE removes the representation of the selected resource. It uses DELETE method in CoAP.

CoAP has also an OBSERVE transfer operation. It obtains a sequence of representations of the selected resource. A new representations is sent immediately after each state change of the resource. Thus, it is like RETRIEVE automatically done after each state change. OBSERVE is implemented in CoAP by using an observe option in the GET message.

3.2 Collection

HSML Collection representation includes Base Elements, Link Elements, and Item Elements.

Base element describes the context under which to interpret values embedded in sebsequent items within the collection. The base identifier element (bi) may contain an absolute URI or an absolute path reference. The format of base elements are specified in [6].

An example base element specifying an absolute relative path /sensors/ is depicted below:

```
{
        ”bi”: ”/sensors/”
```

}

A link element is a hyperlink based on [3] and [8]. An example link specifying a link /sensors/temp (assuming above bi definition) with resource type some.sensor.temp is depicted below:

{

”href”: ”temp”,

”rt”: ”some.sensor.temp”

}

An item element in a collection is a data element that is referenced by a link in the collection. An example item element specifying the above mentioned /sensors/temp is depicted below:

{

”n”: ”temp”,

”v”: 27

}

Note that the name (n) must match with the link’s href!

There are two possibilities to store the items: either directly embedded in the collection, or as separate collections themselves. In this work, we focus on the former one. Thus, an example collection with embedded items is depicted below:

[

{

"bi": "/sensors/"

},

{

"href": "temp",

"rel": "item"

},

{

"href": "humid",

"rel": "item"

},

{

"n": "temp",

"v": 27

```
    },
    {
        "n": "humid",
        "v": 50
    }
]
```


As can be seen, collection format shows all elements: items, links, and link extensions. Other formats, link and item content formats, have only links or items, respectively. These formats can be requested by having an accept or content-type option in the request message, or by having an interface (”if”) URI query parameter in the request. Also other URI parameters (like resource type ”rt”) can be used to select items or links. The matching rules are specified in [3].

3.3 Group communication

HSML defines also group transfer operations, i.e., the transfer method targeted for a collection resource is routed to each resource in the collection that has a group (”grp”) relation in its link. Responses from the resources are aggregated and returned as a single response message. An example group collection is depicted below.

RETRIEVE /sensor-group/ accept=application/hsml.collection+json

Response payload:


```
[
    {
        ”bi”: ”/sensor-group/”
    },
    {
        ”anchor”: ”/sensor-group/”,
        ”rel”: [”self”, ”index”]
    },
    {
        ”href”: ”/sensors/temp”,
        ”rel”: ”grp”
    },
    {
```

```
        ”href”: ”/sensors/humid”,
        ”rel”: ”grp”
    }
]
```

Then, for example, updating both temperature and humidity sensors’ values to zero can be done by a single UPDATE request with the value in the payload:

UPDATE /sensor-group/ content-type=application/hsml.item+json

Payload:

```
[
    {
        ”v”: 0
    }
]
```

3.4 Link extensions

3.4.1 Actions

Actions can be indicated by a ”rel”: ”action” in a link. For example, the code below defines action of type ”st.on” on a link named ”switchcommand”, it is used by using create, i.e., POST method with a payload defined by the schema. The response will be plain text.

```
{
    "rel": "action",
    "type": "st.on",
    "href": "switchcommand",
    "method": "create",
    "accept": "text/plain",
    "schema": {"type": "string", "enum": ["on"]}
}
```

3.4.2 Monitors

Monitors are indicated by a link relation ”monitor” defined in [9]. For example in the code below, the monitor resource behind the link ”tank-level-events” observes the context URI and sends notifications if value is less than 20 or more than 80. The notifications are sent no more than once in 600 seconds but at least once every 3600 seconds.

```
{
        "rel": "monitor",
        "href": "tank-level-events",
        "content-type": "application/senml+json",
        "transfer-method": "create",
        "pmin": 600,
        "pmax": 3600,
        "nbul": 20, #bmx later in the draft!
        "nbll": 80 #bmn later in the draft!
}
```

4. HSML implementation

HSML was implemented on top of CoAPthon [4] library. You can download the HSML supported CoAPthon from https://github.com/mmajanen/CoAPthon/tree/hsml. Note: the HSML support is in the hsml branch, not in the master or other branches. The HSML related changes in the code are marked by comments including ”MiM”. Mostly, the changes are in the end of exampleresources.py, where the classes for the HSML collection (HSMLsensorResource) and basic sensor item (SensorItemResource) resources are defined. The example CoAP server was modified to have sensor collection resource with temperature and humidity sensor items. Also, the coapclient.py and helperclient.py were updated to handle the accept/content-type option in the requests.

4.1 CoAPthon installation and usage

Follow the installation instructions of CoAPthon for installing it. The coapserver.py has HSML collection resource /sensors/ and temperature and humid sensor items in the collection. Use the coapclient.py for testing them. The CoAPthon API documentation can be found under ~/docs/build/html/index.html after executing ”make docs” command.

The CoAP server is started by command ”coapserver.py”.

The CoAP client is started by command ”coapclient.py”. It needs command line arguments:

-o METHOD, where METHOD is one of the GET/POST/PUT/DELETE. DISCOVER and OBSERVE are also supported. DISCOVER is the same as GET for the .well-known/core resource and it returns the available resources on the server in the CoRE Link Format. OBSERVE is the same as GET with the observe option set on. Every time the observed resource changes its state, a notification is sent to the client.

-p PATH, where PATH is the requested URI, e.g., coap://127.0.0.1:5683/sensors/. PATH can also contain Uri-Query options. They are added in the end of the PATH as key=value pairs and separated by ’?’ from the PATH and by ’\&’ from each other. For example, coap://127.0.0.1:5683/sensors?href=temp\&rt=sensor.temp -A 22002.

-A CT, where CT is the accept / content-type option value, e.g., 22000 for hsml+json

-f PAYLOADFILE, where PAYLOADFILE is the file containing the payload for the request; the payload can be also given directly by using -P PAYLOAD. Example payloads are provided in POST?.txt and PUT?.txt files.

4.2 Examples

In the following subsections, several example commands for the client are presented.

4.2.1 RETRIEVE

coapclient.py -o GET -p coap://127.0.0.1:5683/sensors/ -A [22000/22001/22002]

returns the /sensors/ HSML collection, collection links, or collection items, respectively.

coapclient.py -o GET -p coap://127.0.0.1:5683/sensors?if=hsml.link

returns the links of the collection, i.e., interface uri-query parameter can be used instead of accept option. if=hsml.collection, hsml.link, hsml.item for the collection, links, and items, respectively.

coapclient.py -o GET -p coap://127.0.0.1:5683/sensors/temp

returns the temperature sensor value as plain text.

coapclient.py -o GET -p coap://127.0.0.1:5683/sensors?href=temp -A 22002

returns the temperature sensor resource as hsml.item, i.e., href uri-query can be used for resource selection.

coapclient.py -o GET -p coap://127.0.0.1:5683/sensors?rt=sensor.humid -A 22001

returns the humidity sensor resource as hsml.link format, i.e., resource type uri-query can be used for selecting links.

4.2.2 CREATE

coapclient.py -o POST -p coap://127.0.0.1:5683/sensors/ -f POST1.txt -A 22000

creates a new link and item into the /sensors/ collection; the new link and item are specified in the POST1.txt file and its contents are attached to the request's payload.

coapclient.py -o POST -p coap://127.0.0.1:5683/sensors/ -f POST2.txt -A 22001

creates new link into /sensors/ collection; only the link is specified in the request's payload.

coapclient.py -o POST -p coap://127.0.0.1:5683/sensors/ -f POST3.txt -A 22002

creates new link and item into /sensors/ collection; only the item is specified in the request's payload.

4.2.3 UPDATE

coapclient.py -o PUT -p coap://127.0.0.1:5683/sensors?href=temp -f PUT1.txt -A 22001

updates the temperature sensor's resource type as specified in the request's payload.

coapclient.py -o PUT -p coap://127.0.0.1:5683/sensors?href=temp -f PUT2.txt -A 22002

updates the temperature sensor's value as specified in the request's payload.

coapclient.py -o PUT -p coap://127.0.0.1:5683/sensors/ -f PUT2.txt -A 22002

updates the resource item specified in the payload to a value specified in the payload.

coapclient.py -o PUT -p coap://127.0.0.1:5683/sensors/temp -P 123

updates the temperature sensor's value to 123.

4.2.4 DELETE

coapclient.py -o DELETE -p coap://127.0.0.1:5683/sensors/temp

deletes the temperature sensor resource.

coapclient.py -o DELETE -p coap://127.0.0.1:5683/sensors/ -A 22000

deletes the whole collection and resources in it.

coapclient.py -o DELETE -p coap://127.0.0.1:5683/sensors?href=humid -A 22001

deletes the humidity sensor resource.

coapclient.py -o DELETE -p coap://127.0.0.1:5683/sensors?href=temp -A 22002

deletes the temperature sensor resource.

5. Conclusions and future work

This work presented how HSML hypermedia data type can be used with CoAP-based communication. The implementation was based on CoAPthon library and it can be freely downloaded from https://github.com/mmajanen/CoAPthon/tree/hsml. Some HSML features were left as future work items:

- group communication

- actions

-monitors.

Other missing enhancements include, e.g., connection to the data base for persistent data storage.

6. References

[1] M. Koster, ”Media Types for Hypertext Sensor Markup”, draft-koster-t2trg-hsml-01, Internet-Draft, work in progress, March 13, 2017.

[2] Z. Shelby, K. Hartke, C. Bormann, ”The Constrained Application Protocol (CoAP)”, RFC 7252, June 2014.

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

[4] G.Tanganelli, C. Vallati, E.Mingozzi, "CoAPthon: Easy Development of CoAP-based IoT Applications with Python", IEEE World Forum on Internet of Things (WF-IoT 2015).

[5] M. Nottingham, ”Web Linking”, RFC 5988, October 2010.

[6] C. Jennings, Z. Shelby, J. Arkko, A. Keranen, C. Bormann, ”Media Types for Sensor Measurement Lists (SenML)”, draft-ietf-core-senml-08, work in progress, May 23, 2017.

[7] P. Van der Stok, C. Bormann, A. Sehgal, ”PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)”, RFC 8132, April 2017.

[8] K. Li, A. Rahman, C. Bormann, ”Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR”, draft-ietf-core-links-json-08, work in progress, April 27, 2017.

[9] A.B. Roach, ”A SIP Event Package for Subscribing to Changes to an HTTP Resource”, RFC 5989, October 2010.