The 6th International Symposium on Internet of Ubiquitous and Pervasive Things (IUPT 2016)

Linking the Web of Things: LDP-CoAP mapping

Giuseppe Loseto, Saverio Ieva, Filippo Grammegna, Michele Ruta*, Floriano Scioscia, Eugenio Di Sciascio

Politecnico di Bari, via E. Orabona 4, Bari (I-70123), Italy

Abstract

The Linked Data Platform (LDP) W3C Recommendation defined resource management primitives for HTTP only, pushing into the background not-negligible use cases related to Web of Things (WoT) scenarios where HTTP-based communication and infrastructures are unfeasible. This paper proposes a mapping of the LDP specification for Constrained Application Protocol (CoAP) in order to publish Linked Data on the WoT. A general translation of LDP-HTTP requests and responses is provided, as well as a fully comprehensive framework for HTTP-to-CoAP proxying. The theoretical work is corroborated by an experimental campaign using the W3C Test Suite for LDP.

1. Motivation

The World Wide Web Consortium (W3C) has recently released the Linked Data Platform (LDP) specification. It aims to provide a reference format for exposing and managing LD resources on the Web. Particularly, clear and direct guidelines are now given for resource classification made according to resource type. Although this standardization effort improves previous RDF graphs management based on SPARQL 1.1 Graph Store HTTP protocol (https://www.w3.org/TR/sparql11-http-rdf-update/) and basically fixes multiple issues, it leaves out Web of Things (WoT) scenarios where alternative lightweight application protocols surrogate HTTP. This is, for example, the case of CoAP (Constrained Application Protocol)2, a level 7 standard designed for Machine-to-Machine (M2M) communication of constrained devices in the Internet of Things. Following the REST (RePresentational State Transfer) architectural style, CoAP adopts a loosely coupled client/server model, based on stateless operations on resource representations3. Each resource is unambiguously identified by a URI (Uniform Resource Identifier). Clients access resources via asynchronous request/response interactions over a datagram-oriented transport like UDP, using HTTP-derived methods essentially mapping the Read, Create, Update and Delete operations of data management.

Hence, main motivation of the paper stems from the need of extending and enriching the standardization of Linked Data Platforms also to Web of Things use cases. It should be said that the W3C indicated possible solutions for

* Corresponding author. Tel.: +39-339-635-4949; fax: +39-080-596-3410
E-mail address: michele.ruta@poliba.it
resource management in the WoT (see sec. 3.12 of Linked Data Platform Use Cases and Requirements⁴), but their scope is quite limited. In fact, an approach based on a one-to-one HTTP-CoAP translation was followed⁵. Unfortunately, such a mapping only worked with basic HTTP interactions, where several methods and headers were not used and/or some other ones were particularly simplistic. As an example the methods options, head and patch were not allowed as well as several MIME types (content-format) were missing, so that not negligible constraints on resources introduced by LDP could not be considered. Hence, by trivially applying the one-to-one mapping suggested by the W3C for coping with WoT scenarios, significant LDP functionality would be lost.

This paper proposes a novel and specific variant of the HTTP-CoAP mapping able to preserve all the LDP features and capabilities in the Web of Things with a full support of the W3C specification. It also adds modern features giving value to the strongest potentialities of CoAP (e.g., resource discovery based on CoRE Link Format), which are completely absent in HTTP.

The remainder of the paper is organized as follows. The next section presents some background about LDP and CoAP in order to make the work self-consistent, Section 3 introduces the proposed LDP-CoAP mapping, while Section 4 addresses early experiments made to validate and corroborate the proposal. Finally, Section 5 closes the paper.

2. Background

2.1. Linked Data Platform

LDP W3C Recommendation states Linked Data Platform defines a set of rules for HTTP operations on web resources, some based on RDF, to provide an architecture for read-write Linked Data on the web. In particular, the specification describes the use of HTTP methods and headers for accessing and managing resources from LDP servers following the Linked Data approach⁶. Basically, seven types of Linked Data Platform Resources (LDPRs) are defined, conforming to simple patterns and conventions.

- **LDP Resource (LDPR):** a HTTP resource whose status complies with the basic LDP guidelines;
- **LDP RDF Source (LDP-RS):** a LDPR whose status corresponds to an RDF graph and can be fully represented in a RDF syntax. In particular LDP allows text/turtle⁷ and application/ld+json⁸ serializations;
- **LDP Non-RDF Source (LDP-NR):** a LDPR not represented in RDF, i.e., a binary or text document without useful RDF annotation. LDP servers can also generate metadata about LDP-NR resources, e.g., creation date or owner;
- **LDP Container (LDPC):** a LDP-RS as collection of LDP resources. Three types of LDPC are defined, namely Basic, Direct and Indirect;
- **LDP Basic Container (LDP-BC):** is a LDPC defining a simple link to its resources through the ldp:contains predicate, as shown in Figure 1(a);
- **LDP Direct Container (LDP-DC):** is a LDPC increasing the flexibility of a LDP-BC with the membership feature. A LDP-DC contains membership triples, according to the pattern in Figure 1(b), specifying the membership resource and the member relation.
- **LDP Indirect Container (LDP-IC):** is a LDPC similar to a LDP-DC, also capable of having member resources with different URIs, as shown in Figure 1(c), unrelated to the main container URI. These resources are specified using the insertedContentRelation LDP property in the body of LDP requests.

LDP specification also defines required and optional HTTP methods for LDP servers:
- **GET:** retrieves the description associated to the selected LDP resource;
- **POST:** creates a LDP resource in a LDP container;
- **PUT:** creates or updates a LDP resource in a LDP container;
- **DELETE:** removes a LDP resource on a LDP server;
- **HEAD:** retrieves the same HTTP headers as GET responses without body content;
- **OPTIONS:** lists the operations allowed on a LDP resource by means of specific HTTP response headers;
- **PATCH:** allows LDP clients to update a resource description exploiting the Linked Data Patch Format⁹.

The W3C LDP Implementations reference page (http://www.w3.org/wiki/LDP_Implementations) enumerates several frameworks proposed in the last years. The most relevant ones are summarized in Table 1, showing both main features
and supported resources, as reported in the LDP implementation report\(^\text{10}\). All implementations are based on the HTTP protocol. No support is currently given to WoT standards, such as CoAP.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Last Version</th>
<th>License</th>
<th>Language</th>
<th>Supported LDP Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Marmotta</td>
<td>Full release</td>
<td>May 2014</td>
<td>APL 2.0</td>
<td>Java</td>
<td>RS, NR, BC</td>
</tr>
<tr>
<td>Eclipse Lyo</td>
<td>Completed</td>
<td>Aug 2014</td>
<td>EPL 1.0</td>
<td>Java</td>
<td>RS, NR, BC, DC</td>
</tr>
<tr>
<td>rww.io</td>
<td>Pending</td>
<td>Nov 2014</td>
<td>MIT</td>
<td>PHP</td>
<td>RS, BC</td>
</tr>
<tr>
<td>LDP.js</td>
<td>Completed</td>
<td>Apr 2015</td>
<td>APL 2.0</td>
<td>JavaScript</td>
<td>RS, BC</td>
</tr>
<tr>
<td>Fedora 4.4</td>
<td>Full release</td>
<td>Oct 2015</td>
<td>APL 2.0</td>
<td>Java</td>
<td>RS, NR, BC, DC, IC</td>
</tr>
<tr>
<td>Carbon LDP</td>
<td>In progress</td>
<td>Oct 2015</td>
<td>BSD</td>
<td>JavaScript</td>
<td>RS, NR, BC, DC, IC</td>
</tr>
<tr>
<td>LDP4j</td>
<td>In progress</td>
<td>Dec 2015</td>
<td>APL 2.0</td>
<td>Java</td>
<td>RS, BC, DC, IC</td>
</tr>
<tr>
<td>rww-play</td>
<td>In progress</td>
<td>Dec 2015</td>
<td>APL 2.0</td>
<td>Scala</td>
<td>RS, NR, BC</td>
</tr>
<tr>
<td>OpenLink Virtuoso</td>
<td>Full release</td>
<td>Dec 2015</td>
<td>GPLv2</td>
<td>C/C++</td>
<td>RS, BC</td>
</tr>
<tr>
<td>gold</td>
<td>In progress</td>
<td>Jan 2016</td>
<td>MIT</td>
<td>Go</td>
<td>RS, BC</td>
</tr>
<tr>
<td>Callimachus</td>
<td>Full release</td>
<td>Jan 2016</td>
<td>APL 2.0</td>
<td>Java</td>
<td>RS, NR, IC</td>
</tr>
</tbody>
</table>

Table 1: Current LDP implementations

2.2. Constrained Application Protocol

A CoAP message is composed of: (i) a 32-bit header, containing the request method code (or response status); (ii) an optional token value, used to associate replies to requests, (iii) a sequence of option fields (containing information such as resource URI and payload media type), (iv) the payload data. The CoRE Link Format specification\(^\text{11}\) is adopted for resource discovery. A client accesses the reserved URI /\.well-known/core on the server via GET to retrieve available resource entry points. Further GET requests will include URI-query options to retrieve only resources with given attributes. Standardized query attributes include resource type (rt), interface usage (if), content-type (ct), and MIME (Multipurpose Internet Mail Extension) type for a resource. Further non-reserved attributes can be freely used. CoAP also supports proxying, enabling Web applications to transparently access the resources hosted in devices based on CoAP. A CoAP proxy can be explicitly selected by clients (forward-proxy) or can act as the origin server for the target resource (reverse-proxy). Particularly, HTTP-CoAP cross-protocol proxying allows HTTP clients to access resources on CoAP servers; a HTTP request containing a Request-URI with coap or coaps scheme is forwarded to the specified CoAP resource.

3. LDP-CoAP mapping

A novel HTTP to CoAP mapping taught for LDP is presented here, improving the proposal in\(^\text{5}\). LDP-HTTP request methods and headers have been properly translated to the corresponding LDP-CoAP ones and LDP-CoAP
responses are then mapped back to LDP-HTTP. The proposed mapping enables direct CoAP-to-CoAP interaction among devices supporting LDP-CoAP.

Basically, the proposed LDP-CoAP association is obtained applying the following rules:

**HTTP methods** (shown in Table 2) are translated to the corresponding CoAP methods (if present). PATCH, HEAD and OPTIONS, not defined in CoAP, are mapped to existing methods, adding the new Core Link Format attribute `ldp`. This solution extends the basic CoAP functionalities while maintaining a full compatibility with the standard protocol.

**HTTP status codes** are mapped with available CoAP codes as described in Table 2. Codes for bad requests and errors are translated as defined in the proposal.

**HTTP Headers** of request/response messages are translated as in Table 3.

Finally, novel **content-format** media types are introduced in CoAP: `text/turtle` and `application/ld+json`.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Mandatory</th>
<th>Supported in CoAP</th>
<th>LDP-CoAP</th>
<th>HTTP SC</th>
<th>CoAP SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>YES</td>
<td>YES</td>
<td>GET</td>
<td>200 OK</td>
<td>2.05 Content</td>
</tr>
<tr>
<td>POST</td>
<td>NO (optional)</td>
<td>YES</td>
<td>POST</td>
<td>201 Created</td>
<td>2.01 Created</td>
</tr>
<tr>
<td>PUT</td>
<td>NO (optional)</td>
<td>YES</td>
<td>PUT</td>
<td>204 No Content</td>
<td>2.04 Changed</td>
</tr>
<tr>
<td>DELETE</td>
<td>NO (optional)</td>
<td>NO</td>
<td>DELETE</td>
<td>204 No Content</td>
<td>2.02 Deleted</td>
</tr>
<tr>
<td>PATCH</td>
<td>NO (optional)</td>
<td>NO</td>
<td>PUT <code>?ldp=patch</code></td>
<td>204 No Content</td>
<td>2.04 Changed</td>
</tr>
<tr>
<td>HEAD</td>
<td>YES</td>
<td>NO</td>
<td>GET <code>?ldp=head</code></td>
<td>204 No Content</td>
<td>2.03 Valid</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>YES</td>
<td>NO</td>
<td>GET <code>?ldp=options</code></td>
<td>204 No Content</td>
<td>2.05 Content</td>
</tr>
</tbody>
</table>

Table 2: HTTP-CoAP Methods and Status Codes (SCs) Mapping

<table>
<thead>
<tr>
<th>HTTP Header</th>
<th>LDP-CoAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Type</td>
<td>Content-Format <code>(ct)</code> CoAP option</td>
</tr>
<tr>
<td>Link (rel=&quot;type&quot;)</td>
<td>Resource-Type <code>(rt)</code> Core Link Format attribute, available through a CoAP discovery request</td>
</tr>
<tr>
<td>Allow</td>
<td>Not defined in CoAP, available in JSON format as body content of a LDP-CoAP Options request</td>
</tr>
<tr>
<td>Accept-Post</td>
<td></td>
</tr>
<tr>
<td>Accept-Patch</td>
<td></td>
</tr>
<tr>
<td>Slug</td>
<td>title Core Link Format attribute</td>
</tr>
<tr>
<td>Location</td>
<td>location-path CoAP option</td>
</tr>
</tbody>
</table>

Table 3: HTTP-CoAP Headers Mapping

In order to make everything clearer, in what follows some reference examples of HTTP-CoAP translation are given. Note that in some cases an HTTP request cannot be translated into a single CoAP request, but more CoAP messages are needed.

**Example 1. Basic HTTP GET request on an LDP resource**

```
GET /alice/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

The HTTP response is shown in Figure 2. In this case, a single CoAP GET request is not able to produce all the required headers, because some of them are not defined in the response format of CoAP. So the original HTTP request is translated to the following three LDP-CoAP requests:

- a GET message to map Content-Type `(ct)`, ETag (if present) and RDF content of the LDP resource;
- a CoAP discovery message to retrieve the `rt` attribute indicating the LDP type of each resource. It maps the HTTP `Link` response header;
- an OPTIONS message (described later) to map the `Allow`, `Accept-Post` and `Accept-Patch` response headers.

**Example 2. Create a new LDP resource through a HTTP POST request**

```
POST /alice/ HTTP/1.1
Host: example.org
Slug: foaf
Content-Type: text/turtle
```

In this case, the request is translated to a single CoAP POST message with URL:

```
coap://example.org/alice?title=foaf&rt=ldp:Resource ct=text/turtle <payload>
```
As defined in Table 3, title and rt query parameters are obtained from the Slug and Link HTTP header fields, respectively. If the Link header is not defined, ldp:Resource is used as default value of rt. The HTTP response will contain the Location HTTP header corresponding to the Location-Path CoAP response option.

**Example 3. HTTP OPTIONS request on a LDP resource**

An OPTIONS request is used to obtain useful information about a resource, e.g., the list of available methods. HTTP OPTIONS response is shown in Figure 3. Also in this case, multiple LDP-CoAP requests are combined to obtain all the headers produced by the HTTP reply:
- **Allow**, Accept-Post and Accept-Patch response headers are not defined in CoAP so their values are set in the LDP-CoAP OPTIONS response body in JSON syntax and then mapped to the corresponding HTTP headers;
- a CoAP discovery request is used to obtain the resource type (rt) then mapped to the HTTP Link response header.

Further examples for each method are on our LDP-CoAP Web page (http://sisinflab.poliba.it/swottools/ldp-coap).

**4. Validation**

The validation framework consists of four elements, shown in Figure 4:
- **LDP-CoAP Server**, a CoAP server exposing resources complying with LDP-CoAP;
- **CoAP Client**, making requests to the LDP-CoAP server through CoAP;
- **HTTP Client**, querying through HTTP messages a web server which exposes LDP resources. It does not communicate directly with a LDP-CoAP server;
- **LDP-CoAP Proxy**, an HTTP-to-CoAP device used to connect CoAP devices to HTTP-based networks. It is responsible for: (i) processing requests from HTTP clients; (ii) mapping HTTP requests to compatible LDP-CoAP ones via the mapping rules described in Section 3; (iii) forwarding requests to the LDP-CoAP server; (iv) translating the LDP-CoAP responses to HTTP responses to be returned to the client.

Californium Java library\(^\text{12}\) was used to implement the LDP-CoAP framework. In particular, the LDP-CoAP proxy was based on the california-proxy package, whilst the LDP-CoAP server exploited californium-core and OpenRDF Sesame 2.8.6 library (http://rdf4j.org/) for RDF data processing and storage.
Functionality of the proposed framework was evaluated through the W3C LDP Test Suite (http://w3c.github.io/ldp-testsuite/). By default, the suite directly queries a LDP server by means of HTTP messages; therefore for LDP-CoAP tests requests were sent to the server through a LDP-CoAP proxy as in Figure 4. The suite consists of 236 tests referred to rules and restrictions of the LDP W3C specification. Table 4 reports on the obtained results grouped by supported LDP resources: RDF Sources, Non-RDF Sources and Basic, Direct, Indirect Containers. For each test category, the specification requirements are divided in three compliance levels: MUST, SHOULD, and MAY. Due to the lack of space, extensive reports with full details are not reported here, but can be found on the project Web page. Currently unsatisfied test cases are related to the following unsupported features: PUT-to-create and PATCH methods, paging and sorting, preference HTTP headers.

<table>
<thead>
<tr>
<th>Feature</th>
<th>MUST</th>
<th>SHOULD</th>
<th>MAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Container</td>
<td>32/37 (86.5%)</td>
<td>12/17 (70.6%)</td>
<td>3/4 (75.0%)</td>
</tr>
<tr>
<td>Direct Container</td>
<td>37/42 (88.1%)</td>
<td>13/19 (68.4%)</td>
<td>3/4 (75.0%)</td>
</tr>
<tr>
<td>Indirect Container</td>
<td>33/39 (84.6%)</td>
<td>12/17 (70.6%)</td>
<td>3/4 (75.0%)</td>
</tr>
<tr>
<td>Non-RDF Source</td>
<td>12/15 (80.0%)</td>
<td>1/1 (100.0%)</td>
<td>4/6 (66.7%)</td>
</tr>
<tr>
<td>RDF Source</td>
<td>22/24 (91.7%)</td>
<td>5/7 (71.4%)</td>
<td>1/1 (100.0%)</td>
</tr>
</tbody>
</table>

Table 4: LDP-CoAP Test Suite results summary

5. Conclusion and Future Work

This paper introduced a CoAP mapping of the Linked Data Platform specification for publishing Linked Data on the Web of Things. The LDP W3C Recommendation, which defined resource management primitives only for HTTP, stated the need for this work. The proposal includes a translation of LDP-HTTP requests and responses, as well as a framework for HTTP-to-CoAP proxying.

Future work will include a performance evaluation of the proposed framework to assess the impact of LDP support in resource-constrained devices. Furthermore, mapping the currently unsupported LDP features is under way, in order to increase the compatibility with the specification: running the test suite again on future revisions will measure progress in this area. Finally, a complete scenario will be defined to expose both real-time data and sensor observations (e.g., weather data) according to LDP-CoAP specifications.

References