Linked Data (in Low-Resource) Platforms: A Mapping for Constrained Application Protocol

Giuseppe Loseto, Saverio Ieva, Filippo Gramegna, Michele Ruta(✉), Floriano Scioscia, and Eugenio Di Sciascio

Politecnico di Bari, via E. Orabona 4, 70125 Bari, Italy
{giuseppe.loseto,saverio.ieva,filippo.gramegna,
michele.ruta,floriano.scioscia,eugenio.disciascio}@poliba.it

Abstract. This paper proposes a mapping of the Linked Data Platform (LDP) specification for Constrained Application Protocol (CoAP). Main motivation stems from the fact that LDP W3C Recommendation presents resource management primitives for HTTP only. A general translation of LDP-HTTP requests and responses is provided, as well as a framework for HTTP-to-CoAP proxying. Experiments have been carried out using the LDP W3C Test Suite.

Keywords: Linked Data Platform · CoAP · Semantic web of things

Resource type: Software
Permanent URL: http://dx.doi.org/10.5281/zenodo.50701

1 Introduction and Motivation

The World Wide Web Consortium (W3C) has standardized the Linked Data (LD) management on the Web with the Linked Data Platform (LDP) specification [8]. Unfortunately, this effort leaves out the so-called Web of Things (WoT) where HTTP is replaced by simpler protocols, *e.g.*, CoAP (Constrained Application Protocol) [12], suitable for resource-constrained scenarios. CoAP adopts a loosely coupled client/server model, based on stateless operations on *resources* [2] identified by URIs (Uniform Resource Identifiers). Clients access them via asynchronous request/response interactions through HTTP-derived methods mapping the Read, Create, Update and Delete operations of data management. Section 3.12 of Linked Data Platform Use Cases and Requirements [1] reports on a possible one-to-one translation of HTTP primitives toward CoAP, nevertheless the proposed solution appears quite limited. The given mapping [3] only considers basic HTTP interactions: `options`, `head` and `patch` methods are not allowed and various MIME content-format types are missing.

Main motivation of this resource is to enable the extension of the Linked Data Platform standard to Web of Things contexts. A specific variant of the HTTP-CoAP mapping is proposed, preserving LDP features and capabilities:

P. Groth et al. (Eds.): ISWC 2016, Part II, LNCS 9982, pp. 131–139, 2016.
DOI: 10.1007/978-3-319-46547-0_14

Table 1. Current LDP implementations

Name	Status	Last Version	License	Language	Supported LDP Resources
RWW.IO	Pending	1.2 (Nov 2014)	MIT	PHP	RS, BC
Apache Marmotta	Full release	3.3.0 (Dec 2014)	APL 2.0	Java	RS, NR, BC
Bygle	In progress	Feb 2015	APL 2.0	Java	RS, BC
Eclipse Lyo	Completed	2.1.0 (Mar 2015)	EPL 1.0	Java	RS, NR, BC, DC
LDP.js	Completed	Apr 2015	APL 2.0	JavaScript	RS, BC, DC
Glutton	In progress	Apr 2015	GPLv3	Python	RS, BC
Carbon LDP	In progress	0.5.7 (Oct 2015)	BSD	JavaScript	RS, NR, BC, DC, IC
LDP4j	In progress	0.2.0 (Dec 2015)	APL 2.0	Java	RS, BC, DC, IC
RWW Play	In progress	2.3.6 (Dec 2015)	APL 2.0	Scala	RS, NR, BC
Fedora	Full release	4.5.0 (Jan 2016)	APL 2.0	Java	RS, NR, BC, DC, IC
Callimachus	Full release	1.5.0 (Mar 2016)	APL 2.0	Java	RS, NR, IC
Gold	In progress	1.0.1 (Apr 2016)	MIT	Go	RS, BC
OpenLink Virtuoso	Full release	7.2.5 (Apr 2016)	GPLv2	C/C++	RS, BC
ldnode	In progress	0.2.31 (Apr 2016)	MIT	JavaScript	RS, BC

the envisioned HTTP-CoAP proxy makes objects networks first-class Linked Data providers on the Web. Novel features are also giving added value to the strongest peculiarities of CoAP with respect to HTTP, *e.g.*, resource discovery via CoRE Link Format. The proposed solution is released as open source. Performance tests evidence LDP-CoAP supports all types of LDP resources keeping computational performances comparable with other frameworks. Results of the W3C LDP conformance test suite show the proposal does not completely cover LDP specification yet.

2 Coping with Lightweight Linked Data Platform

The LDP W3C Recommendation provides standard rules for accessing and managing Linked Data on the Web *LDP servers*. Basically, it defines seven types of LDP *Resources* as well as patterns of HTTP methods and headers for CRUD (Create, Read, Update, Delete) operations[1]. W3C LDP implementations web page (http://www.w3.org/wiki/LDP_Implementations) lists several software tools: Table 1 reports the most relevant ones along with main properties and supported resource types, in order of release date. All solutions are based on the HTTP protocol, with no current support for WoT standards such as CoAP.

The W3C suggests explicit use cases [1] aiming to integrate LDP in resource-constrained devices and networks with specific reference to CoAP [12], a compact protocol conceived for machine-to-machine (M2M) communication. Some CoAP options are derived from HTTP header fields (*e.g.*, content type, headers and proxy support), while some other ones have no counterpart in HTTP. So an HTTP-CoAP mapping is needed to exploit all LDP features with CoAP. An early mapping proposal was defined in [3], but it only worked with basic HTTP interactions. The HTTP-CoAP mapping for LDP envisioned in [7] and outlined here,

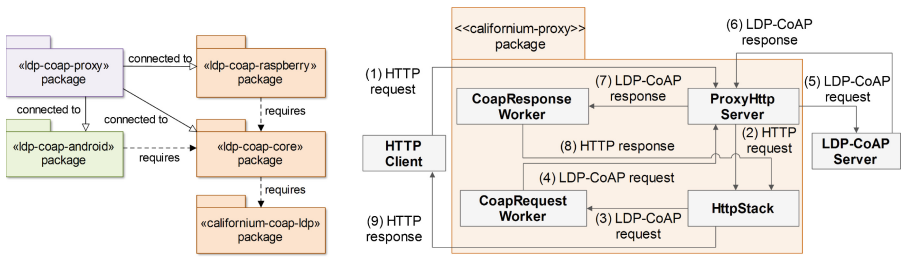
[1] Due to space constraints, details of LDP specification are not recalled here; basic knowledge of LDP is assumed, while the reader is referred to [8] for details.

Table 2. HTTP-CoAP mapping of preference headers

HTTP Header	LDP-CoAP
`Prefer: return=representation; include="pref"`	`ldp-incl=pref` Core Link Format attribute
`Prefer:return=representation; omit="pref"`	`ldp-omit=pref` Core Link Format attribute
`Preference-Applied: return=representation`	*pref* returned using `location-query` CoAP option

enables a direct CoAP-to-CoAP interaction. *HTTP methods* mapping is applied for each CoAP method (if present). `HEAD` and `OPTIONS`, undefined in CoAP, are mapped to existing `GET` and `PUT` methods, by adding the new Core Link Format attribute `ldp`. There is full backward compatibility with the standard protocol, while extending the basic CoAP functionalities. W.r.t. the early proposal [7], additional features have been also defined to support: (i) `PATCH` method; (ii) *RDF Patch* format [10] along with `application/rdf-patch` content-format media type; (iii) LDP *Prefer headers* of request/reply messages (Table 2).

LDP-CoAP mapping was implemented in a Java-based framework providing the basic components required to publish Linked Data on the WoT according to LDP-CoAP specification. It consists of several modules, as shown in Fig. 1a.



(a) Main modules (b) LDP HTTP-CoAP proxy server

Fig. 1. LDP-CoAP framework architecture

ldp-coap-core: includes the implementation of all LDP-CoAP resources and a basic LDP-CoAP server handling CoAP-based communication and RDF data management. The main Java package `coap.ldp` was partitioned in the following sub-packages each providing a specific functionality.

– `coap.ldp.server`: the reference `CoAPLDPServer` implementation. It extends the `CoAPServer` provided by *californium-core-ldp* module (described below) and exposes methods to create and manage LDP resources. The package also includes the `CoAPLDPTestSuiteServer`, used for experiments described in Sect. 3.

– `coap.ldp.resources`: according to the LDP resource hierarchy [8], several Java classes were developed extending the `CoAPLDPResource` base class providing common methods and attributes. For each resource class, a specific data handler can be implemented to retrieve whatever kind of data (*e.g.*, observation from a sensor) and update the RDF repository with user-defined periodicity. Handlers can be defined starting from the `LDPDataHandler` abstract class. In this way, developers can build specific applications implementing the whole business logic and data management procedures within the `handleData` method of the handler, without any other modification of the source code. `CoAPLDPResourceManager` implements read-write operations on the RDF data storage exploiting an *OpenRDF Sesame* (http://rdf4j.org) in-memory RDF repository.
– `coap.ldp.handler`: two simple handlers were defined as usage examples to expose real-time system CPU load and RAM usage ratio as LDPRDFResource. Data are collected through the operating system interfaces of Java 7 (or later).
– `coap.ldp.exception`: a `CoAPLDPException` class was defined to catch errors due to incorrect usage of LDP methods, headers or attributes. Its subclasses represent typical problems (*e.g.*, *content format* or *precondition failed*).
– `rdf.vocabulary`: contains RDF ontologies mapped as Java classes to simplify creation and querying of RDF triple. As an example, *SSN-XG* ontology [4] was mapped through the *Sesame Vocabulary Builder* (http://github.com/tkurz/sesame-vocab-builder) tool and included here.

The following libraries are required to correctly compile *ldp-coap-core*: *JSON-java* (http://github.com/stleary/JSON-java) to format data in JSON; *jsonld-java* (http://github.com/jsonld-java) to support the `json-ld` specification [6]; *Apache Marmotta RDF Patch Util* (http://marmotta.apache.org/sesame.html) to update RDF statements of a Sesame repository according to the `rdf-patch` [10] format.

californium-core-ldp: a modified version of the *Californium* CoAP framework [5], extended to support LDP features. Main modifications include: (i) novel content-format media types added to `MediaTypeRegistry` class; (ii) additional response codes introduced within `CoAP` main class.

ldp-coap-proxy: a modified version of *californium-proxy* implementing the mapping rules defined before and translating LDP-HTTP request to the corresponding LDP-CoAP ones. As shown in Fig. 1b, LDP-CoAP mapping procedures take advantage of the classes in this module. In particular, *ProxyHttpServer* is responsible for processing a request –coming from a generic HTTP client– through its *HttpStack* member class where the mapping occurs. *HttpStack* transforms an HTTP request into a compatible LDP-CoAP one and for each CoAP request it starts two threads, *CoapRequestWorker* and *CoapResponseWorker*, synchronized according to the producer-consumer pattern. The *CoapRequestWorker* thread produces the LDP-CoAP translated request for the *ProxyHttpServer* class instance which forwards that request to the proper LDP-CoAP server. The *CoapResponseWorker* is responsible for consuming and translating the LDP-CoAP response coming from the *ProxyHttpServer* into the HTTP response which is returned to the client.

In addition to the basic framework, the following two packages were developed to build LDP-CoAP applications on embedded and resource-constrained devices.

ldp-coap-raspberry: *ldp-coap-core* was tested on a Raspberry Pi (http://www.raspberrypi.org) board. W.r.t. other LDP implementations, LDP-CoAP is very lightweight and simple to run on low-resource environments like Raspberry Pi, having a minimum number of dependencies and low system requirements in terms of memory and processing capabilites. As a reference example, two handlers were implemented to publish CPU temperature and free RAM as LDP resources. Data are retrieved using the *Pi4J* (http://pi4j.com) library.

ldp-coap-android: a simple project exploiting *ldp-coap-core* on Android devices. It runs unmodified on all platforms supporting modules compiled with Java SE runtime environment, version 7 or later, so it can be directly used as a library also by Android applications. Android OS provides a uniform interface (the Android sensor framework, http://developer.android.com/guide/topics/sensors/sensors_overview.html) to access sensor data. Therefore, a single handler (named `GenericSensorHandler`) was implemented to manage both hardware and software-based device sensors. The project includes a basic activity starting a LDP-CoAP server exposing data from interface sensors modeled as LDP resources. Source code is available on [9], including *Javadoc* documentation; quick usage examples are on the project website http://sisinflab.poliba.it/swottools/ldp-coap. All modules were developed as Eclipse (http://eclipse.org) projects using Apache Maven (http://maven.apache.org) to manage dependencies. Only *ldp-coap-android* is a project for Android Studio (http://developer.android.com/tools/studio/index.html), the Google official IDE for app development. In this case, all dependencies can be defined through a Gradle (http://gradle.org) configuration file.

A few validation examples are reported here, in order to clarify the proposal. Full examples are on the LDP-CoAP project website.

Ex. 1 – Basic HTTP GET request on an LDP resource. HTTP-CoAP mapping is shown in Fig. 2. As described in [7], a single CoAP `GET` request cannot produce all the needed headers. So the original HTTP request (Fig. 2a) is translated to three LDP-CoAP packets: a `GET` message (Fig. 2d), a CoAP discovery message (Fig. 2c), and an `OPTIONS` message (Fig. 2b). In particular, since `Allow`, `Accept-Post` and `Accept-Patch` response headers are not defined in CoAP, their values are set in the LDP-CoAP `OPTIONS` response body in JSON syntax and then mapped to the corresponding HTTP headers. As per the CoRE Link Format specification [11], the CoAP discovery request maps the HTTP `Link` header with the resource type (`rt`) retrieved via the /.well-known/core reserved resource path.

Ex. 2 – Create a new LDP resource through an HTTP POST request. In this case, the HTTP request (Fig. 3a) is translated to a single CoAP POST message, as in Fig. 3b (see [7] for details).

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

HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#BasicContainer> rel="type",
<http://www.w3.org/ns/ldp#Resource> rel="type"
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json
Accept-Patch: application/rdf-patch
Content-Length: 250
ETag: W/'123456789'
...RDF payload...
```

(a) HTTP GET

```
GET coap://example.org/alice?ldp=options

2.05 Content
Content-Format (ct): application/json
{
"Allow": ["OPTIONS", "HEAD", "GET",
"POST", "PUT", "PATCH"],
"Accept-Post": [ "text/turtle",
"application/ld+json"],
"Accept-Patch": "application/rdf-patch"
 }
```

(b) CoAP OPTIONS

```
GET coap://example.org/.well-known/core?title=alice

2.05 Content
Content-Format (ct): application/link-format
</alice>
rt="http://www.w3.org/ns/ldp#BasicContainer
http://www.w3.org/ns/ldp#Resource";
ct=4; title="alice"
```

(c) CoAP Discovery

```
GET coap://example.org/alice/
Accept: text/turtle

2.05 Content
Content-Format (ct): text/turtle
ETag: W/'123456789'
...RDF payload...
```

(d) CoAP GET

Fig. 2. HTTP-CoAP mapping for an LDP GET request/response

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

HTTP/1.1 201 Created
Location: http://example.org/alice/foaf
Link: <http://www.w3.org/ns/ldp#Resource> rel='type'
Content-Length: 0
```

(a) HTTP POST

```
POST coap://example.org/alice?title=foaf
Content-Format (ct): text/turtle
...RDF payload...

2.01 Created
Location-Path:
coap://example.org/alice/foaf
```

(b) CoAP POST

Fig. 3. HTTP-CoAP mapping for an LDP POST request/response

3 Experiments

The W3C LDP Test Suite (http://w3c.github.io/ldp-testsuite/) is used to evaluate the functionality of the proposed framework and to compare it with existing solutions. The suite consists of 236 tests which query an LDP server by means of HTTP messages; only for LDP-CoAP requests were sent to the server through an LDP-CoAP proxy as in Fig. 1a. Obtained results are grouped by supported LDP resources (RDF Sources, Non-RDF Sources and Basic, Direct, Indirect Containers – see [8] for definitions) and compliance levels (MUST, SHOULD, MAY). For each resource/level pair, Table 3 compares the score of LDP-CoAP with the highest value obtained by other LDP tools. Full LDP-CoAP results are on the project website. Overall, LDP-CoAP presents good scores, when considering 17 manual tests were skipped in this first experimental campaign and only automated ones were executed.

In addition to LDP-CoAP 7 tools were evaluated: *Virtuoso*, *LDP.js*, *Apache Marmotta*, *LDP4j*, *RWW.IO*, *Fedora4* and *Eclipse Lyo*. They were selected according to the features listed in Table 1: current status, completeness, open license, last update and supported resources (in particular RDF Source and Basic

Table 3. Comparison of implementation conformance tests

Feature	MUST		SHOULD		MAY	
	LDP-CoAP	Highest Val.	LDP-CoAP	Highest Val.	LDP-CoAP	Highest Val.
LDP-RS	91.7 % (22/24)	100 % [a,b,c,d,e]	71.4 % (5/7)	100 % [a,b,c,d]	100 % (1/1)	100 % [all]
LDP-BC	86.5 % (32/37)	100 % [b,c,d,e]	88.2 % (15/17)	100 % [b,c]	100 % (4/4)	100 % [b,c,e,f]
LDP-DC	88.1 % (37/42)	100 % [b,d,e]	89.5 % (17/19)	100 % [b]	100 % (4/4)	100 % [b,d,f]
LDP-IC	84.6 % (33/39)	97.4 % [a]	88.2 % (15/17)	88.2 % [d]	100 % (4/4)	100 % [f]
LDP-NR	80.0 % (12/15)	100 % [a,b,c]	100 % (1/1)	100 % [a,b,c,f,i]	66.7 % (4/6)	100 % [b,c,f]

(a) Callimachus, (b) Eclipse Lyo, (c) Apache Marmotta, (d) LDP4j, (e) LDP.js, (f) Fedora4, (g) ldphp, (h) Virtuoso, (i) rww-play

Container). *Gold* was tested and discarded due to the limited compatibility with LDP specification. Only supported resources were taken into account to retrieve processing time. Each test was repeated three times on the same PC and (only for tests passed by all tools) the average value was reported in Fig. 4. Fedora4 and LDP-CoAP support all LDP resources. Eclipse Lyo and LDP4j manage four resources groups, whereas remaining frameworks only operate on RDF Sources and Basic Containers. LDP-CoAP has good processing times, as results are comparable with the other implementations even while involving the HTTP-CoAP proxy. Only for non-RDF Source tests performance is slightly worse.

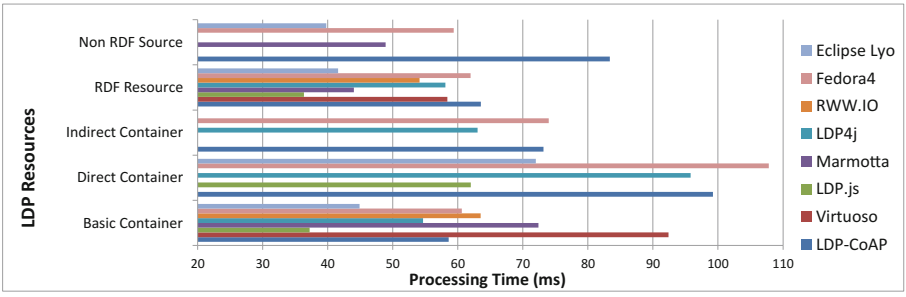


Fig. 4. Comparison of processing time for tested LDP implementations

To evaluate the feasibility of exploiting LDP in mobile and pervasive computing scenarios, LDP-CoAP performance was tested on three different Java-compatible platforms: a PC[2], an Android smartphone (LG Google E960 *Nexus 4*, specifications at http://www.lg.com/us/cell-phones/lg-LGE960-nexus-4) and a Raspberry Pi 1 Model B+ board (http://www.raspberrypi.org/products/model-b-plus/) All requests were originated from a PC client running both the LDP Test Suite and the LDP HTTP-CoAP proxy, connected through a local IEEE 802.11 network to one of the three LDP-CoAP servers for each test. The overall processing time, shown in Fig. 5, is defined as the time elapsed from

[2] With Intel Core i7 CPU 3770K at 3.50 GHz (4 cores/8 threads), 12 GB DDR3-SDRAM (1333 MHz), 2 TB SATA (7200 RPM) HD, 64-bit Microsoft Windows 7 Professional and 64-bit Java 8 SE Runtime Environment (build 1.8.0_65-b17).

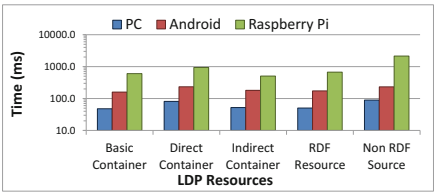


Fig. 5. Device time comparison

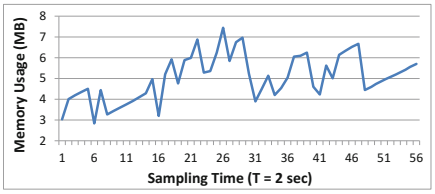


Fig. 6. Memory use on Raspberry Pi

sending the request until receiving a response by the client, including communication and HTTP-CoAP message translation times. Values on Android are roughly 3 times higher than on PC, whereas performance on Raspberry are an order of magnitude higher with respect to PC. However, average response times are under 1 second both on Android and Raspberry (except for LDP-NR responses on Raspberry). Memory usage was also measured every 2 s during the execution of the test suite for the three platforms. Memory allocation peak of the LDP-CoAP server was about 44.7 MB on PC, 18.3 MB on Android and 7.4 MB on Raspberry. Stricter memory constraints on smartphones and embedded devices imposes to have as much free memory as possible at any time. Consequently, on these platforms Java virtual machines perform more frequent and aggressive garbage collection (see Fig. 6). The garbage collector was invoked many times, corresponding to the falling edges in the chart. This behavior reduces memory usage, but on the other hand it causes the processing time gap found on the different platforms.

4 Future Directions

This paper presented an LDP-CoAP mapping and framework for managing Linked Data in the Web of Things. Performance tests evidence LDP-CoAP supports all types of LDP resources and its computational performances are comparable with those of other frameworks. Future revisions will extend compliance as much as possible; progress will be measured through test suite adopted here. Planned developments also include: evolving the forks of Californium core and proxy modules to merge them with the original codebase eventually; adding the capability to manage RDF resources on persistent storage in addition to in-memory ones; porting LDP-CoAP server to more languages (*e.g.*, C/C++, Python, Go) and computing platforms (*e.g.*, Arduino).

References

1. Battle, S., Speicher, S.: Linked data platform use cases and requirements. W3C Working Group Note, W3C, March 2014. http://www.w3.org/TR/ldp-ucr/
2. Bormann, C., Castellani, A., Shelby, Z.: CoAP: an application protocol for billions of tiny internet nodes. IEEE Internet Comput. **16**(2), 62–67 (2012)

3. Castellani, A., Loreto, S., Rahman, A., Fossati, T., Dijk, E.: Guidelines for HTTP-CoAP Mapping Implementations. Internet-Draft 07, IETF Secretariat, July 2015
4. Compton, M., Barnaghi, P., Bermudez, L., Garcia-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., et al.: The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. Web Semantics: Science, Services and Agents on the World Wide Web **17**, (2012)
5. Kovatsch, M., Lanter, M., Shelby, Z.: Californium: Scalable cloud services for the Internet of Things with CoAP. In: International Conference on the Internet of Things, 2014, pp. 1–6. IEEE (2014)
6. Lanthaler, M., Sporny, M., Kellogg, G.: JSON-LD 1.0. W3C Recommendation, W3C, January 2014. http://www.w3.org/TR/json-ld/
7. Loseto, G., Ieva, S., Gramegna, F., Ruta, M., Scioscia, F., Di Sciascio, E.: Linking the web of things: LDP-CoAP mapping. In: Shakshuki, E., (ed.) 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)/Affiliated Workshops. Procedia Computer Science, vol. 83, pp. 1182–1187. Elsevier, May 2016
8. Malhotra, A., Arwe, J., Speicher, S.: Linked Data Platform 1.0. W3C Recommendation, W3C, February 2015. http://www.w3.org/TR/ldp/
9. Ruta, M., Scioscia, F., Loseto, G., Ieva, S., Gramegna, F., Sciascio, E.D.: LDP-CoAP: Linked Data Platform for the Constrained Application Protocol (v1.0) (2016). http://dx.doi.org/10.5281/zenodo.50701
10. Seaborne, A., Vesse, R.: RDF Patch Describing Changes to an RDF Dataset. Unofficial Draft, August 2014. https://afs.github.io/rdf-patch/
11. Shelby, Z.: Constrained RESTful Environments (CoRE) Link Format. RFC 6690, August 2012. http://www.ietf.org/rfc/rfc6690.txt
12. Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252, June 2014. http://www.ietf.org/rfc/rfc7252.txt