Mapping the NGSI-LD Context Model on Top of a SPARQL Event Processing Architecture: Implementation Guidelines

Fabio Viola*†✉, Francesco Antoniazzi†, Cristiano Aguzzi†, Carlos Kamienski‡, Luca Roffia†
*Centro Nazionale per la Ricerca e Sviluppo nelle Tecnologie Informatiche e Telematiche
Istituto Nazionale di Fisica Nucleare (INFN-CNAF), Bologna, Italy
fabio.viola@cnaf.infn.it
† University of Bologna, Bologna, Italy
{name.surname}@unibo.it
‡ Federal University of ABC (UFABC), Santo Andrè, Brazil
cak@ufabc.edu.br

***Abstract*—NGSI-LD is an open specification released by ETSI which proposes an information model and an API for an easy to use and standard management of context information. The NGSI-LD information model is framed within an ontology and adopts JSON-LD as serialization format for context information. This paper presents an approach to the implementation of the NGSI-LD specification over a SPARQL Event Processing Architecture. This work is being developed within the European-Brasilian H2020 SWAMP project focused on implementing an Internet of Things platform providing services for smart water management in agriculture.**

I. Introduction

SWAMP (Smart WAter Management Platform, http://swamp-project.org) is a EU funded research project exploiting Internet of Things (IoT) [1] technologies to solve issues related to water management. More specifically, SWAMP is aimed at reducing water wastage caused by leakages in distribution and irrigation systems or by inefficiencies of the irrigation methods. In fact, surface irrigation (the most common technique) wastes water by irrigating areas where no plants live. Localized irrigation is instead more efficient; nonetheless, farmers usually tend to use more water than needed to avoid under-irrigation. The SWAMP project investigates on how to employ technology to estimate the amount of water needed by the plants to efficiently use water. This project currently counts on the participation of industrial and academic partners from Italy, Finland, Spain and Brazil.

Among the technologies adopted in SWAMP there are FIWARE [2] and SEPA [3].

FIWARE (https://www.fiware.org/) is a EU-funded initiative born in the context of the FI-PPP programs (Future Internet Public Private Partnership) to support the growth of EU global competitiveness through an innovative infrastructure for cost-effective creation and delivery of services [4]. FIWARE provides a platform with a set of Generic Enablers for the development of smart applications [5]. FIWARE is currently being used in many real life application, among which we mention SmartPort [6] where it is employed to manage and monitor the seaport of Las Palmas de Gran Canaria. For what concerns the management of context information, FIWARE relies on the NGSI-LD open specification [7] r eleased by ETSI. NGSI-LD defines the context information model and the API to produce, consume and subscribe to context information.

SEPA (short for SPARQL Event Processing Architecture) [3], is an architecture supporting the development of dynamic linked data applications and services. It provides a publish/subscribe layer on top of standard SPARQL endpoints, granting the ability to subscribe to changes of a given context over a Linked Data network.

Both NGSI-LD and SEPA APIs aim at fulfilling t he requirements of Tim Berners-Lee's 5-star model [8]; according to this model, in fact, data should be visible, structured and described according to standards and its meaning is clarified by a common definition (i.e., an ontology). If on the one hand the NGSI-LD specification a ims a t s implifying t he management of context information (e.g., by limiting the query language to simple property-value matching), on the other hand SEPA provides full support to the SPARQL language for updating, retrieving and subscribing to context information. This paper proposes a merge of the two approaches to exploit the benefits of both. This is achieved through a set of guidelines for implementing an NGSI-LD interface over SEPA.

After an overview of the related works in Section II, the background knowledge needed to read the paper is presented (Section III). Section IV proposes an in-depth analysis of the design guidelines for the implementation of the NGSI-LD HTTP binding on top of a SEPA broker. Eventually, in Section V, conclusions are drawn.

II. Related Work

The Internet of Things is profundly changing a wide range of research areas, among which agriculture. IoT applications are usually considered as three-layered architectures composed by a sensing (or perception) layer, a networking layer and an application (or service) layer [9]. The new generation of

ISSN 2305-7254

agricultural applications based on the IoT paradigm must face several challenges involving all of these layers. An example is given by security that involves the sensing layer (i.e., the hardware and its acquisition methods), the networking layer (and the way information is transferred) and the application layer that must ensure read/write access only to the users with the right permission. A big challenge of the networking layer is represented by the quality of the transmission. Wireless communication helps to reduce the costs of wiring large plots of land. Nevertheless, communication may be affected by the environmental conditions (e.g., humidity, temperature) as well as obstacles between transmitters and receivers. A second challenge that cannot be neglected is that of interoperability, due to the proliferation of devices and protocols caused by the IoT [10], [11]. All of these challenges are well surveyed by Tzounis et al. [12].

Many IoT platforms are currently available and employed in agricultural applications. OpenIoT [13], for example, is the platform supporting Phenonet, an Australian wireless sensor network collecting information over a field of experimental crops with the aim to provide scientists and farmers with a platform for high resolution crop analysis in real world growing conditions [14]. Karim et al. in [15] proposed an application prototype for precision farming based on the IoT platform Ubidots. ThingSpeak has been used by Mondal and Rehena in [16], where an intelligent field monitoring system is proposed. Soil humidity and temperature are measured by their platform in order to automate irrigation. As regards the adoption of semantic technologies in the IoT for agriculture, the work by Yuan et al. [17] (in collaboration with ChinaMobile) deserves a mention. In their paper, a semantic framework to improve farmers' ability of decision making is proposed (e.g., when to fertilize).

III. Background

This Section introduces the two main technologies behind this Research work: FIWARE, and its specification for the information model (i.e., NGSI-LD) are detailed in Subsection III-A; SEPA and the underlying technologies borrowed from the Semantic Web are presented in Subsection III-B

A. FIWARE and NGSI-LD

FIWARE was born to create an open source platform that can be assembled together with other third-party components to speed up the development of IoT-Cloud solutions. The platform consists of a set of software agents called Generic Enablers (GE) that must be defined by GE Open Specifications. They serve as a public royalty-free blueprint that GE implementations (GEi) must follow to be considered as part of the platform. Furthermore, for every specification, FIWARE provides at least one open source reference implementation aiming at fostering the adoption of the related solution within the community.

FIWARE has been already employed in practical IoT applications. For example in [4] where the authors describe the process of building a remote e-health monitoring platform for caregivers providing practical software architecture insights. Moreover [18] reports the advantages of using FIWARE in the development of an IoT precision agriculture application in respect to another cloud solution.

The advantage of using FIWARE is that software architects can exploit a consolidated set of open-source solutions aimed at handling specific IoT problems. In fact, several GEs have been developed to address different needs (https://catalogue-server.fiware.org/), like storing time series, managing sensor networks, messaging, big data analysis etc. Among these, one of the most relevant is the Orion Context Broker.

Orion is aimed at addressing data interoperability issues among different IoT silos. This is achieved through a publish-subscribe mechanism decoupling data producers and consumers, and an information model based on linked data. The latter follows the NGSI-LD specification, now published as a recommendation in [19].

NGSI-LD defines three levels of data abstraction: the core meta-model, cross domain model and the domain specific model.

The core meta-model defines the atomic minimal information that can be published in Orion: entities, properties and relationships. In particular the semantic of these abstract components is very close to the Entity-Relationship model of relational databases. Everything in Orion is an entity that may have zero or more properties and may be linked to others with zero or more relationships. Properties and relationships may have, in turn, properties (e.g. the property *WaterQuality* may have a property *Accuracy*).

To contextualize data and to be compliant with the Linked Data world, every entity should specify a context binding the data representation to a vocabulary. Moreover, vocabularies can be published trough Orion and linked through URLs to entity instances.

On the other hand the cross domain model is more focused on concepts like time and space. Among others it contains the concept of geolocation, creation time and time intervals.

Finally the domain specific model should be more tighten to the actual application. For example, Fig. 1 reports a possible data model for a parking utility application.

B. SEPA and SPARQL 1.1 Subscribe Protocol

The idea of Semantic Web was born within the vision outlined in [20] and [21]. Various new concepts were introduced to support this Web revolution, intending to transform the web into something that simultaneously would fit the needs of human users, but yet would also be machine understandable.

Since then, a lot of research and work has been done. For instance, it's worth citing the semantic repositories hosted into DBpedia (https://wiki.dbpedia.org/) and WikiData (https://query.wikidata.org/), which contain millions of triples and link information on a plethora of topics. Special platforms have also been developed, like LOV (Linked Open Vocabularies, https://lov.linkeddata.es/dataset/lov) [22]. LOV contributes to foster the use of semantic ontologies [23], which are an

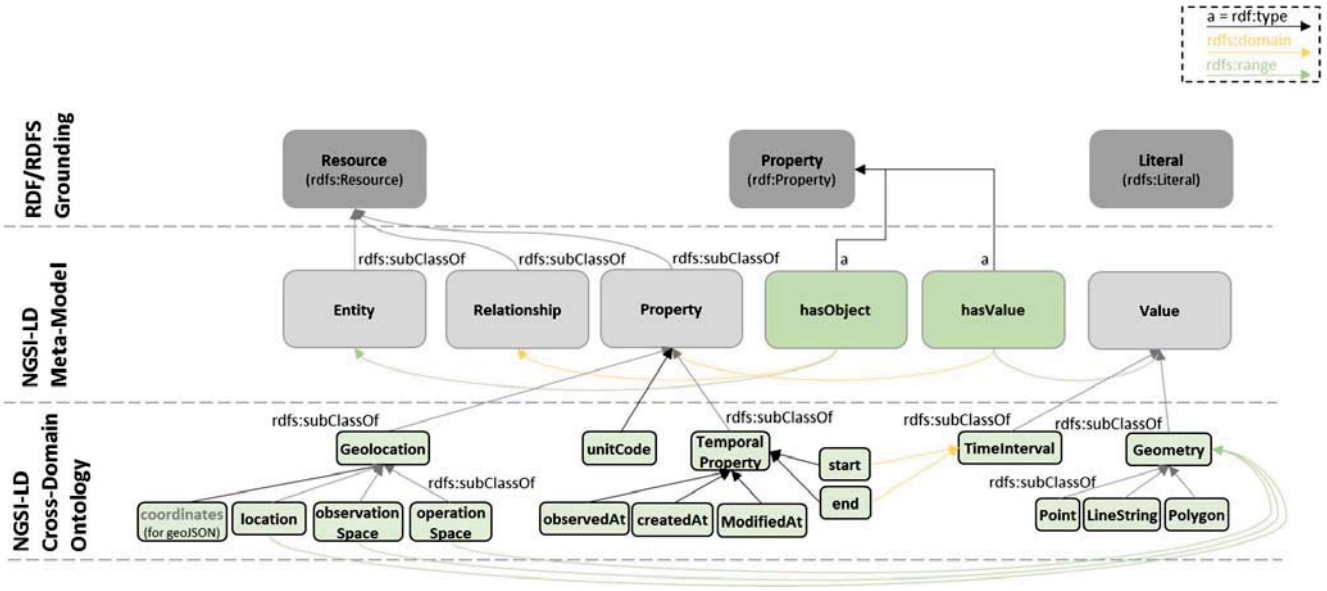

Fig. 1. The NGSI-LD information model (Fig. 4.2.3-1 in [19])

essential tool to grant interoperability at information level between services and systems over the time [24].

Interoperability at information level is the key concept on which SEPA, and all its predecessors of the Smart-M3 family [25], [26], [27], [28] pivot. The term "interoperability", as a matter of fact, includes a reference to the active and responsive behaviour of systems [29]: what if, instead of semantically codify only the internal data of the system, its whole activity and information flow was semantically defined?

SEPA implements an architecture to give an answer to this question. In particular, it is made of two sections addressing on one hand the storage of semantic data, the information flow on the other.

SEPA stores the semantic data in an underlying RDF graph contained in a SPARQL endpoint like Blazegraph (https://www.blazegraph.com/), Fuseki (https://jena.apache.org/documentation/fuseki2/) and Virtuoso (https://virtuoso.openlinksw.com/rdf/). At this level, interoperability between systems is reached by formatting the graph (also known as Knowledge Base) according to one or more ontologies, so that one can consistently query and use the data produced by the other. The graph represents the data-context of the application, and can be manipulated by the entities running: it is possible to add new pieces to the graph, and remove them. Or, even simpler, it is possible to check which are the triples in it. All those tasks can be performed through the SEPA, by posting SPARQL Updates and Queries according to the SPARQL 1.1 Secure Event Protocol (http://mml.arces.unibo.it/TR/sparql11-se-protocol.html).

The novelty of SEPA, however, is its subscription engine on top of the RDF endpoint. The setup of such engine enriches the knowledge base with a semantic publish-subscribe layer leveraging the graph's contents. The internal mechanisms of the subscription engine are out of the scope of this paper, that instead focuses on how SEPA fosters interactions through SPARQL 1.1 Subscribes. Subscriptions, in particular, work as follows:

1) First of all, a subgraph within the knowledge base has to be defined by writing the corresponding SPARQL pattern, as it is done for all queries;
2) Secondly, a communication channel is opened from the client towards the SEPA engine, by which the subscription is declared;
3) A first result is received containing the actual result of the query. The client can, therefore, synchronize its own view of the current context with the one contained in the RDF store;
4) Eventually, the publish-subscribe mechanism is started: from now on, the communication channel will be used to notify changes that match the chosen pattern in the knowledge base. This means that the client is going to be asynchronously warned that new items have been added and/or some items have been removed from the graph. Then, the usage of that information is up to the client's business logic.

As for February 2019, SEPA has been implemented, maintained and released in a Java version available on Github (https://github.com/arces-wot/SEPA). The development, however, is still ongoing targeting the study of new Subscription Evaluation and Triggering algorithms, to enhance its performances. This is a great challenge that is located at the core of the SEPA project, i.e., the detection as fast as possible of the subscriptions that have to be triggered given the received update request.

To use SEPA, according to the specifications of the SPARQL 1.1 SE Protocol previously cited, Java, JavaScript

and Python3 APIs have been developed, while C++ and C (for constrained and IoT devices like Arduino) are being scheduled for the future. All those APIs, nevertheless, follow the standards required by SEPA to communicate: HTTP(S) GETs for queries, POST for updates; Websocket(S) for subscriptions. A different SEPA implementation, realized using CoAP protocol, has also been studied in [30].

Fig. 2 depicts the relationship between SEPA and SEPA-based applications. Applications read/write information (i.e., access the KB) according to the rules expressed by one or more ontologies. The access is mediated by SEPA, that stands on top of the knowledge base and provides an interface for updates and subscriptions. Moreover, whenever a change in the KB is detected, SEPA generates the proper notifications for the related subscriptions.

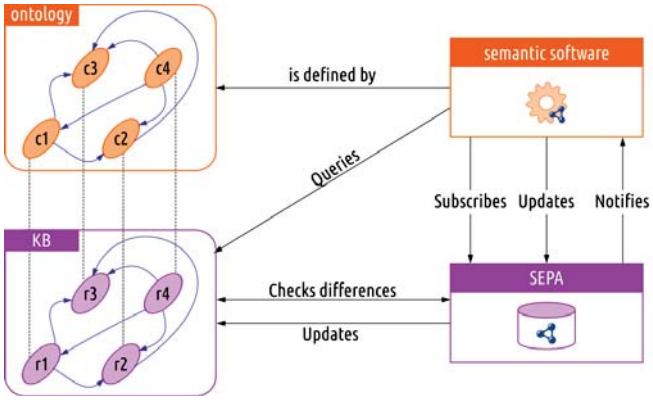

Fig. 2. SEPA and SEPA-based applications

IV. NGSI-LD HTTP BINDING API IMPLEMENTATION GUIDELINES

This section discusses on the implementation of the NGSI-LD API [7] over SEPA (see Section III-B). In particular, the implementation is focused on the HTTP binding of the NGSI-LD API (see Section 6 in [7]) which consists in 23 REST calls enumerated in Table I. For each REST call it is specified the HTTP verb (i.e., POST, GET, DELETE, PATCH) and the URI of the relative path (e.g., `/entities`) to be appended to the main URI (i.e., NGSI-LD recommends the following format for the URI: `http(s)://host/apiRoot/apiName/apiVersion`). For example, a new entity can be created by making use of API 1 which consists in a HTTP POST at an URI like `https://mml.arces.unibo.it/swamp/ngsi-ld/v1/entities`, having as body of the request the JSON-LD representation of the entity to be created.

The NGSI-LD API allows to manage context information and context sources (i.e., represented according to the NGSI-LD information model described in [7], Section 4.2). Within the NGSI-LD information model, the context is represented by entities characterized by properties and linked together through relationships. Table I summarizes some of the most relevant APIs (i.e., APIs for the batch creation/update/deletion of entities and attributes, including the temporal representation of an entity, are not listed) which allow to provide (APIs 1-6), consume (APIs 7-8) and subscribe (APIs 9-13) to context information. NGSI-LD assumes that context information is produced by context sources which can register (APIs 14-16) to a context broker and that can be discovered (APIs 17-18) by other clients. As for context information, also for context sources the API provides a notification mechanism (APIs 19-23).

The first step of the implementation would consist in mapping each API with the corresponding SEPA primitive (i.e., SPARQL 1.1 Update [31], SPARQL 1.1 Query [32], SPARQL 1.1 Subscribe [33]). The envisioned mapping is shown in Table I. For example, the creation of an entity (API 1) would correspond to a SPARQL 1.1 Update (i.e., a INSERT DATA primitive), while retrieving a particular entity (API 7) would be implemented through SPARQL 1.1 Queries (i.e., a recursive series of CONSTRUCTs primitives). In some cases an API would be mapped in one or more SPARQL 1.1 queryies followed by a SPARQL 1.1 Update. For example, deleting an entity (i.e., API 2) would be implemented by retrieving the corresponding RDF graph (i.e., query) and deleting all the triples included in the graph (i.e., update).

The modular architecture of a SEPA broker allows supporting new protocols by implementing new gates (see NGSI-LD gate in Fig. 3). In the case of the NGSI-LD HTTP

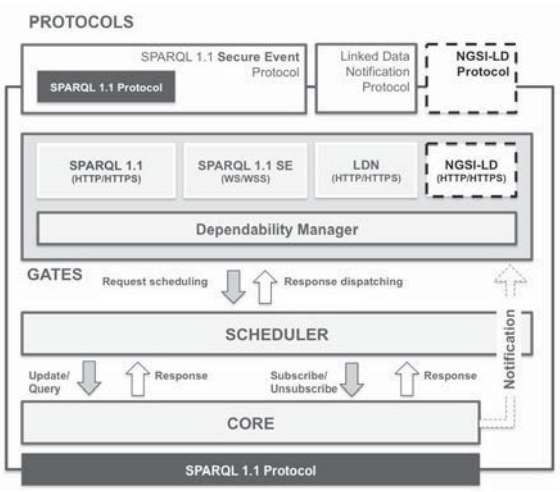

Fig. 3. Extension of the SEPA broker to implement the NGSI-LD protocol

binding, the corresponding SEPA gate is aimed at mapping NGSI-LD HTTP requests into SPARQL primitives and serialize/deserialize RDF in/from JSON-LD. Thanks to the publish-subscribe mechanism implemented by a SEPA broker, supporting the NGSI-LD subscriptions, both for context information and context sources, would be straightforward. Considering for example APIs 9 and 19, these would correspond to a SPARQL 1.1 Update (i.e., the storage of the subscription details into the underpinning SPARQL endpoint), followed

TABLE I. NGSI-LD HTTP BINDING API

Context Information			
Entitites Provisioning	Create[a]	POST	/entities
	Delete[a,b]	DELETE	/entities/*entityId*
Attributes Provisioning	Append[a]	POST	/entities/*entityId*/attrs
	Update[a]	PATCH	/entities/*entityId*/attrs
	Delete[a]	DELETE	/entities/*entityId*/attrs/*attrId*
	Partial update[a]	PATCH	/entities/*entityId*/attrs/*attrId*
Entitites Consumption	Retrieve[b]	GET	/entities/*entityId*
	Query[b]	GET	/entities
Subscription	Create[a,c]	POST	/subscriptions
	Query[b]	GET	/subscriptions
	Retrieve[b]	GET	/subscriptions/*subscriptionId*
	Update[a,c]	PATCH	/subscriptions/*subscriptionId*
	Delete[a,c]	DELETE	/subscriptions/*subscriptionId*
Context Source			
Registration	Register[a]	POST	/csource
	Update[a]	PATCH	/csource/*registrationId*
	Delete[a]	DELETE	/csource/*registrationId*
Discovery	Query[b]	GET	/csource
	Retrieve[b]	GET	/csource/*registrationId*
Subscription	Create[a,c]	POST	/csourceSubscriptions
	Query[b]	GET	/csourceSubscriptions
	Update[a,c]	PATCH	/csourceSubscriptions/*subscriptionId*
	Retrieve[b]	GET	/csourceSubscriptions/*subscriptionId*
	Delete[a,c]	DELETE	/csourceSubscriptions/*subscriptionId*

(a) Mapped as SPARQL 1.1 Update (b) Mapped as SPARQL 1.1 Query (c) Mapped as SPARQL 1.1 Subscribe

by a SPARQL 1.1 Subscribe (http://mml.arces.unibo.it/TR/sparql11-subscribe.html) (i.e., the activation of the new subscription). The subscription mechanism proposed by NGSI-LD assumes that notifications are sent to an endpoint whose URL is specified at subscription time. This would require the NGSI-LD gate to map the notifications directed to the NGSI-LD interface to be mapped into HTTP POST requests as specified in [7], Section 6.3.8. Presenting a complete implementation of the NGSI-LD protocol is out of the scope of this work. Instead, the following sections focus on two APIs that can be considered as representative of the overall implementation process. The first is the creation of a new entity (API 1), while the second is the retrieval of an entity given its identifier (API 7). The former provides the details on how to perform the conversion from a JSON-LD representation of an entity into a set of RDF triples, while the latter focuses on the opposite way, namely serializing a set of triples into a representation compliant with NGSI-LD.

The following sections consider as example the entity description in Listing 1 (inspired by the examples found in [7], Annex C).

```
{"@id": "urn:ngsi-ld:Vehicle:V123",
"@type": "Vehicle",
"speed": {
 "@type": "Property",
  value": 23,
 "accuracy": {
  "@type": "Property",
   value": 0.7 },
  "providedBy": {
   "@type": "Relationship",
   "object": "urn:ngsi-ld:Person:Bob"}},
"closeTo": {
 "@type": "Relationship",
 "object": "urn:ngsi-ld:Building:B1234"},
 "location": {
  "@type": "GeoProperty",
    value": {
   "@type": "Point",
   "coordinates": [-8,44]}},
"@context": [
"Property":"http://uri.etsi.org/ngsi-ld/Property",
"Relationship":"http://uri.etsi.org/ngsi-ld/Relationship",
"GeoProperty":"http://uri.etsi.org/ngsi-ld/GeoProperty",
"object":"http://uri.etsi.org/ngsi-ld/hasObject",
"value":"http://uri.etsi.org/ngsi-ld/hasValue",
"location":"http://uri.etsi.org/ngsi-ld/location",
"coordinates":"http://uri.etsi.org/ngsi-ld/coordinates",
"Point":"https://purl.org/geojson/vocab#Point",
"Vehicle":"http://uri.fiware.org/ns/datamodels/Vehicle",
"speed":"http://uri.fiware.org/ns/datamodels/speed",
"accuracy":"http://uri.fiware.org/ns/datamodels/accuracy",
"providedBy":"http://uri.fiware.org/ns/datamodels/
    providedBy",
"closeTo":"http://uri.fiware.org/ns/datamodels/closeTo"]}
```

Listing 1. Example of a JSON-LD representation of an NGSI-LD entity context information

A. Entity creation: from JSON-LD to RDF

According to the NGSI-LD API specification, the entity described in Listing 1 can be created by making an HTTP POST (e.g., to `https://mml.arces.unibo.it/swamp/ngsi-ld/v1/entities`) having as body the above mentioned JSON-LD. The corresponding RDF graph is shown in Fig. 4.

The N-TRIPLE [34] serialization can be obtained through the Java method in Listing 2 which uses the functionali-

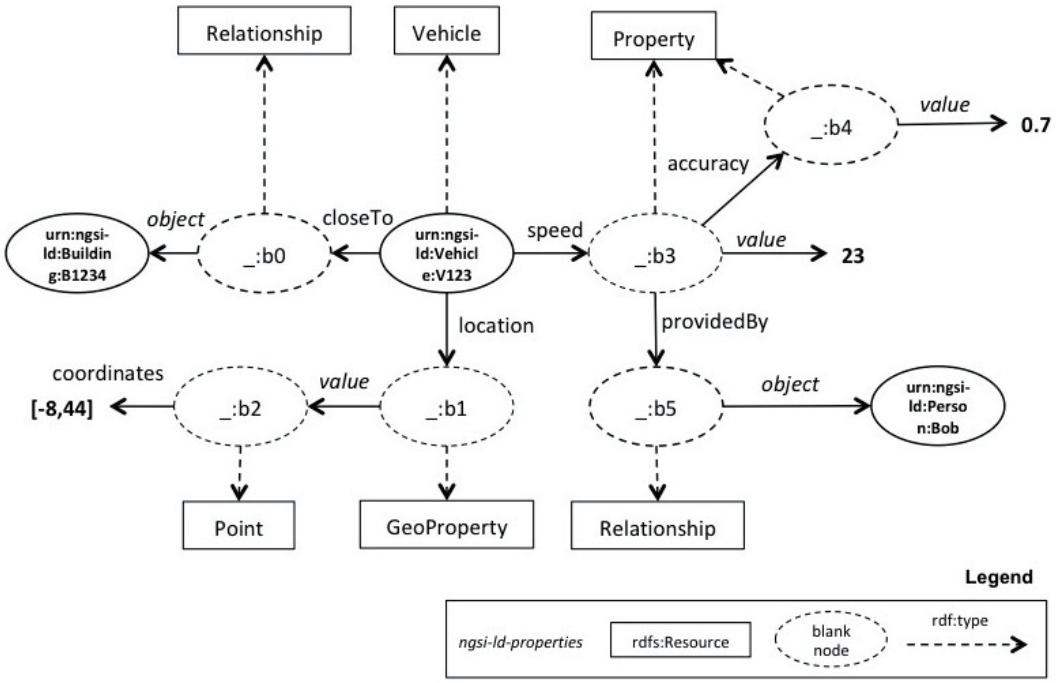

Fig. 4. The RDF graph corresponding to the NGSI-LD entity representation in Listing 1 can be obtained through a reification process

ties offered by the Apache Commons RDF libraries (http://commons.apache.org/proper/commons-rdf/).

```
import java.io.IOException;
import java.io.InputStream;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.rdf.api.Dataset;
import org.apache.commons.rdf.api.RDFSyntax;
import org.apache.commons.rdf.api.Triple;
import org.apache.commons.rdf.jsonldjava.JsonLdRDF;
import org.apache.commons.rdf.jsonldjava.experimental.
    JsonLdParser;

public static String jsonLd2NQuads(InputStream in, String
    baseIRI, int timeout) throws IllegalStateException,
    IllegalArgumentException, InterruptedException,
    ExecutionException, TimeoutException, IOException {

String nquads = "";

// Parse input stream as JSON-LD
JsonLdRDF ld = new JsonLdRDF();
Dataset ldDataset = ld.createDataset();
new JsonLdParser().base(baseIRI).source(in).contentType(
    RDFSyntax.JSONLD).target(ldDataset).parse().get(timeout
    ,TimeUnit.SECONDS);

// Serialize the results as NQUADS (NTRIPLES as default
    graph is used)
for (Triple triple : ldDataset.getGraph().iterate()) {
    nquads += triple.getSubject().ntriplesString() + "␣" +
        triple.getPredicate().ntriplesString() + "␣" +
        triple.getObject().ntriplesString() + "␣.\r\n";}
return nquads;
}
```

Listing 2. Java code to convert a JSON-LD into NTRIPLE format

The output produced is shown in Listing 3 and it would be embedded in the body of a SPARQL 1.1 INSERT DATA update.

```
<urn:ngsi-ld:Vehicle:V123> <http://www.w3.org/1999/02/22-
    rdf-syntax-ns#type> <http://uri.fiware.org/ns/
    datamodels/Vehicle> .
<urn:ngsi-ld:Vehicle:V123> <http://missing/closeTo> _:b0 .
<urn:ngsi-ld:Vehicle:V123> <http://uri.etsi.org/ngsi-ld/
    location> _:b1 .
<urn:ngsi-ld:Vehicle:V123> <http://uri.fiware.org/ns/
    datamodels/speed> _:b3 .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Relationship> .
_:b0 <http://uri.etsi.org/ngsi-ld/hasObject> <urn:ngsi-ld:
    Building:B1234> .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/GeoProperty> .
_:b1 <http://uri.etsi.org/ngsi-ld/hasValue> _:b2 .
_:b2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    https://purl.org/geojson/vocab#Point> .
_:b2 <http://uri.etsi.org/ngsi-ld/coordinates> "-8"^^<http
    ://www.w3.org/2001/XMLSchema#integer> .
_:b2 <http://uri.etsi.org/ngsi-ld/coordinates> "44"^^<http
    ://www.w3.org/2001/XMLSchema#integer> .
_:b3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Property> .
_:b3 <http://missing/accuracy> _:b4 .
_:b3 <http://missing/providedBy> _:b5 .
_:b3 <http://uri.etsi.org/ngsi-ld/hasValue> "23"^^<http://
    www.w3.org/2001/XMLSchema#integer> .
_:b4 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Property> .
_:b4 <http://uri.etsi.org/ngsi-ld/hasValue> "7.0E-1"^^<http
    ://www.w3.org/2001/XMLSchema#double> .
_:b5 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Relationship> .
_:b5 <http://uri.etsi.org/ngsi-ld/hasObject> <urn:ngsi-ld:
    Person:Bob> .
```

Listing 3. List of triples produced as deserialization of the JSON-LD in Listing 1

B. Entity retrieval: from RDF to JSON-LD

According to the NGSI-LD API (see API 7 in Table I), the context information related to an entity (e.g., the entity identified by `urn:ngsi-ld:Vehicle:V123`) can be retrieved by making an HTTP POST at the entity URI (e.g., `https://mml.arces.unibo.it/swamp/ngsi-ld/v1/entities/urn:ngsi-ld:Vehicle:V123`). As the NGSI-LD data model assumes that properties and relationships can be themselves characterized by other properties and relationships (see for example the *accuracy* property and the *providedBy* relationship related to the *speed* property in Fig. 4), retrieving the RDF graph related to an entity is a incremental process which starts from the entity identifier (e.g., `urn:ngsi-ld:Vehicle:V123`). A possible solution is based on extending the SPARQL 1.1 Query template shown in Listing 4 (i.e., `$entityId` would be replaced by the current entity identifier) until the following condition is met: *for each blank node which is object of a triple there must be at least one triple in which that blank node is subject.*

```
CONSTRUCT {?entity ?p0 ?o0 . ?o0 ?p1 ?o1 ...} WHERE {{
 VALUES ?entity {$entityId} ?entity ?p0 ?o0}
UNION {
 VALUES ?entity {$entityId} ?entity ?p0 ?o0 . ?o0 ?p1 ?o1}
...
}
```

Listing 4. SPARQL query to construct the triples representing an entity identified by $entityId

With reference to the entity described in Listing 1, the corresponding RDF graph (see Fig. 4) can be constructed by extending the query template two times as shown in Listing 5.

```
***********************
*** First iteration ***
***********************
CONSTRUCT {?entity ?p0 ?o0}
WHERE {{
 VALUES ?entity {<urn:ngsi-ld:Vehicle:V123>}
 ?entity ?p0 ?o0}
}
----------------------------
Results (first triples set)
----------------------------
<urn:ngsi-ld:Vehicle:V123> <http://www.w3.org/1999/02/22-
    rdf-syntax-ns#type> <http://uri.fiware.org/ns/
    datamodels/Vehicle> .
<urn:ngsi-ld:Vehicle:V123> <http://missing/closeTo> _:b0 .
<urn:ngsi-ld:Vehicle:V123> <http://uri.etsi.org/ngsi-ld/
    location> _:b1 .
<urn:ngsi-ld:Vehicle:V123> <http://uri.fiware.org/ns/
    datamodels/speed> _:b3

************************
*** Second iteration ***
************************
CONSTRUCT {?entity ?p0 ?o0 . ?o0 ?p1 ?o1}
WHERE {{
 VALUES ?entity {<urn:ngsi-ld:Vehicle:V123>}
 ?entity ?p0 ?o0}
UNION {
 VALUES ?entity {<urn:ngsi-ld:Vehicle:V123>}
 ?entity ?p0 ?o0 . ?o0 ?p1 ?o1}
}
-------------------------------------------------
Results (triples to be added to the triples set)
-------------------------------------------------
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Relationship> .
_:b0 <http://uri.etsi.org/ngsi-ld/hasObject> <urn:ngsi-ld:
    Building:B1234> .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/GeoProperty> .
_:b1 <http://uri.etsi.org/ngsi-ld/hasValue> _:b2 .
_:b3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Property> .
_:b3 <http://missing/accuracy> _:b4 .
_:b3 <http://missing/providedBy> _:b5 .
_:b3 <http://uri.etsi.org/ngsi-ld/hasValue> "23"^^<http://
    www.w3.org/2001/XMLSchema#integer>

************************
*** Third iteration ***
************************
CONSTRUCT {?entity ?p0 ?o0 . ?o0 ?p1 ?o1 . ?o1 ?p2 ?o2}
WHERE {{
 VALUES ?entity {<urn:ngsi-ld:Vehicle:V123>}
 ?entity ?p0 ?o0}
UNION {
 VALUES ?entity {<urn:ngsi-ld:Vehicle:V123>}
 ?entity ?p0 ?o0 . ?o0 ?p1 ?o1}
UNION {
 VALUES ?entity {<urn:ngsi-ld:Vehicle:V123>}
 ?entity ?p0 ?o0 . ?o0 ?p1 ?o1 . ?o1 ?p2 ?o2}
}
-------------------------------------------------
Results (triples to be added to the triples set)
-------------------------------------------------
_:b2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    https://purl.org/geojson/vocab#Point> .
_:b2 <http://uri.etsi.org/ngsi-ld/coordinates> "-8"^^<http
    ://www.w3.org/2001/XMLSchema#integer> .
_:b2 <http://uri.etsi.org/ngsi-ld/coordinates> "44"^^<http
    ://www.w3.org/2001/XMLSchema#integer> .
_:b4 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Property> .
_:b4 <http://uri.etsi.org/ngsi-ld/hasValue> "7.0E-1"^^<http
    ://www.w3.org/2001/XMLSchema#double> .
_:b5 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://uri.etsi.org/ngsi-ld/Relationship> .
_:b5 <http://uri.etsi.org/ngsi-ld/hasObject> <urn:ngsi-ld:
    Person:Bob>
```

Listing 5. Building the RDF graph corresponding to an entity

After the first iteration, three triples have as objects blank nodes which are not subjects of any triple (i.e., `_:b0`, `_:b1`, `_:b3`). Because of that, the query has to be extended to include also the triples which have as subject the previously mentioned blank nodes. The query is extended by adding the following triple pattern: `?o0 ?p1 ?o1`. Also after the second iteration there are blank nodes (e.g., `_:b2`, `_:b4`, `_:b5`) which are present as objects of some triples but not as subjects of any triple. A third iteration is required (i.e., the triple pattern `?o1 ?p2 ?o2` is added). As shown in Listing 4, after the third iteration all the resulting triples do not include any blank node. The iterative process can end and the result is the union of all the triples resulting by each iteration. This would correspond to the set of triples listed in Listing 3.

The next step consists in representing this set of triples according to the JSON-LD format used by NGSI-LD (i.e., the result should be equal to Listing 1). This can be achieved thanks to JSON-LD 1.1 Framing [35]. The Java code and the frame used to obtain the JSON-LD in Listing 1 are respectively shown in Listing 6 and Listing 7.

```
import java.io.IOException;
import java.io.InputStream;

import com.github.jsonldjava.core.JsonLdOptions;
import com.github.jsonldjava.core.JsonLdProcessor;
import com.github.jsonldjava.utils.JsonUtils;
```

```
public static String nQuads2JsonLd(String nquads,
     InputStream frame) throws IOException {

// Options
JsonLdOptions opts = new JsonLdOptions();
opts.setUseNativeTypes(true);
opts.setPruneBlankNodeIdentifiers(true);
opts.setOmitGraph(true);

// Parse NQUADS input string
Object jsonObject = JsonLdProcessor.fromRDF(nquads, opts);

// Parse JSON-LD frame from input stream
final Object frameObj = JsonUtils.fromInputStream(frame);

// Framing algorithm
Object framed = JsonLdProcessor.frame(jsonObject, frameObj,
      opts);

return JsonUtils.toPrettyString(framed);
}
```

Listing 6. Java code to serialize RDF triples into a JSON-LD

```
{"@type" : "Vehicle",
"@context" : {
"accuracy" : "http://missing/accuracy",
"providedBy" : "http://missing/providedBy",
"closeTo" : "http://missing/closeTo",
"object" : "http://uri.etsi.org/ngsi-ld/hasObject",
"Relationship" : "http://uri.etsi.org/ngsi-ld/Relationship
     ",
"GeoProperty" : "http://uri.etsi.org/ngsi-ld/GeoProperty",
"value" : "http://uri.etsi.org/ngsi-ld/hasValue",
"Point" : "https://purl.org/geojson/vocab#Point",
"coordinates" : "http://uri.etsi.org/ngsi-ld/coordinates",
"Property" : "http://uri.etsi.org/ngsi-ld/Property",
"Vehicle" : "http://uri.fiware.org/ns/datamodels/Vehicle",
"location" : "http://uri.etsi.org/ngsi-ld/location",
"speed" : "http://uri.fiware.org/ns/datamodels/speed"}}
```

Listing 7. JSON-LD frame used to serialize RDF triples into an NGSI-LD compliant JSON-LD

It should be noticed that the frame depends on the type of entity (e.g., Vehicle) and so it should be created on the fly, request by request. In particular, the entity type can be always extracted from the results of the first iteration of the proposed approach (e.g., the triple `urn:ngsi-ld:Vehicle:V123 rdf:type http://uri.fiware.org/ns/datamodels/Vehicle` in Listing 5).

V. Conclusions and Future Work

In this paper we presented the fundamental blocks required to implement the NGSI-LD API on top of the SPARQL Event Processing Architecture (SEPA). The motivation of the presented work comes from the direct experience derived from the ongoing European-Brazilian H2020 SWAMP project. SWAMP aims to provide smart services focused on water saving in agriculture through the development of an Internet of Things platform. The platform is being built on top of FIWARE which adopts NGSI-LD for the management of context information, but also includes SEPA as technology supporting dynamic Linked Data services and applications. Implementing NGSI-LD on top of SEPA would enable a SEPA broker to become a generic enabler of FIWARE, enhancing the discovery and reasoning capabilities of FIWARE thanks to the fully support of SPARQL, and creating a bridge between FIWARE and the Linked Data cloud.

Acknowledgements

The work is being developed and experimented within the H2020-EUB-2017 SWAMP project, n. 777112, funded by the European Commission under: H2020-EU.2.1.1. - INDUSTRIAL LEADERSHIP - Leadership in enabling and industrial technologies - Information and Communication Technologies (ICT).

References

[1] K. Ashton *et al.*, "That internet of things thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] C. Kamienski, J.-P. Soininen, M. Taumberger, R. Dantas, A. Toscano, T. Salmon Cinotti, R. Filev Maia, and A. Torre Neto, "Smart water management platform: Iot-based precision irrigation for agriculture," *Sensors*, vol. 19, no. 2, 2019.

[3] L. Roffia, P. Azzoni, C. Aguzzi, F. Viola, F. Antoniazzi, and T. S. Cinotti, "Dynamic linked data: A sparql event processing architecture," *Future Internet*, vol. 10, p. 36, 2018.

[4] A. García Vázquez, P. Soria-Rodriguez, P. Bisson, D. Gidoin, S. Trabelsi, and G. Serme, "Fi-ware security: Future internet security core," in *Towards a Service-Based Internet* (W. Abramowicz, I. M. Llorente, M. Surridge, A. Zisman, and J. Vayssière, eds.), (Berlin, Heidelberg), pp. 144–152, Springer Berlin Heidelberg, 2011.

[5] F. Ramparany, F. G. Marquez, J. Soriano, and T. Elsaleh, "Handling smart environment devices, data and services at the semantic level with the fi-ware core platform," in *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 14–20, IEEE, 2014.

[6] P. Fernández, J. M. Santana, S. Ortega, A. Trujillo, J. P. Suárez, C. Domínguez, J. Santana, and A. Sánchez, "Smartport: A platform for sensor data monitoring in a seaport based on fiware," in *Sensors*, 2016.

[7] I. S. G. C. Information, "ETSI GS CIM 009 V1.1.1 (2019-01), Context Information Management (CIM); NGSI-LD API," tech. rep., ETSI Management (ISG-CIM), 2019.

[8] J. Höchtl and P. Reichstädter, "Linked open data - a means for public sector information management," in *Electronic Government and the Information Systems Perspective* (K. N. Andersen, E. Francesconi, Å. Grönlund, and T. M. van Engers, eds.), (Berlin, Heidelberg), pp. 330–343, Springer Berlin Heidelberg, 2011.

[9] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[10] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi, "From today's intranet of things to a future internet of things: a wireless-and mobility-related view," *IEEE Wireless communications*, vol. 17, no. 6, 2010.

[11] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *Systems Engineering Symposium (ISSE), 2017 IEEE International*, pp. 1–7, IEEE, 2017.

[12] A. Tzounis, N. Katsoulas, T. Bartzanas, and C. Kittas, "Internet of things in agriculture, recent advances and future challenges," *Biosystems Engineering*, vol. 164, pp. 31–48, 2017.

[13] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P. Calbimonte, M. Riahi, K. Aberer, P. P. Jayaraman, A. Zaslavsky, I. P. Žarko, *et al.*, "Openiot: Open source internet-of-things in the cloud," in *Interoperability and open-source solutions for the internet of things*, pp. 13–25, Springer, 2015.

[14] P. P. Jayaraman, D. Palmer, A. Zaslavsky, A. Salehi, and D. Georgakopoulos, "Addressing information processing needs of digital agriculture with openiot platform," in *Interoperability and Open-Source Solutions for the Internet of Things* (I. Podnar Žarko, K. Pripužić, and M. Serrano, eds.), (Cham), pp. 137–152, Springer International Publishing, 2015.

[15] F. Karim, F. Karim, and A. Frihida, "Monitoring system using web of things in precision agriculture," *Procedia Computer Science*, vol. 110, pp. 402–409, 2017.

[16] M. AshifuddinMondal and Z. Rehena, "Iot based intelligent agriculture field monitoring system," in *2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 625–629, IEEE, 2018.

[17] Y. Yuan, W. Zeng, and Z. Zhang, "A semantic technology supported precision agriculture system: A case study for citrus fertilizing," in *Knowledge Science, Engineering and Management* (M. Wang, ed.), (Berlin, Heidelberg), pp. 104–111, Springer Berlin Heidelberg, 2013.

[18] J. Lpez-Riquelme, N. Pavn-Pulido, H. Navarro-Helln, F. Soto-Valles, and R. Torres-Snchez, "A software architecture based on fiware cloud for precision agriculture," *Agricultural Water Management*, vol. 183, pp. 123 – 135, 2017. Special Issue: Advances on ICTs for Water Management in Agriculture.

[19] *ETSI GS CIM 004 V1.1.1 (2018-04)Context Information Management (CIM);Application Programming Interface (API)*.

[20] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[21] N. Shadbolt, T. Berners-Lee, and W. Hall, "The semantic web revisited," *IEEE intelligent systems*, vol. 21, no. 3, pp. 96–101, 2006.

[22] P.-Y. Vandenbussche, G. A. Atemezing, M. Poveda-Villalón, and B. Vatant, "Linked open vocabularies (lov): a gateway to reusable semantic vocabularies on the web," *Semantic Web*, vol. 8, no. 3, pp. 437–452, 2017.

[23] A. Maedche and S. Staab, "Ontology learning for the semantic web," *IEEE Intelligent systems*, vol. 16, no. 2, pp. 72–79, 2001.

[24] N. F. Noy, "Semantic integration: a survey of ontology-based approaches," *ACM Sigmod Record*, vol. 33, no. 4, pp. 65–70, 2004.

[25] A. D'Elia, F. Viola, L. Roffia, P. Azzoni, and T. S. Cinotti, "Enabling interoperability in the internet of things: A osgi semantic information broker implementation," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 13, no. 1, pp. 147–167, 2017.

[26] F. Morandi, L. Roffia, A. D'Elia, F. Vergari, and T. S. Cinotti, "Redsib: a smart-m3 semantic information broker implementation," in *Open Innovations Association (FRUCT), 2012 12th Conference of*, pp. 1–13, IEEE, 2012.

[27] F. Viola, A. D'Elia, L. Roffia, and T. Salmon Cinotti, "A modular lightweight implementation of the smart-m3 semantic information broker," in *Proceedings of the 18th Conference of Open Innovations Association FRUCT*, pp. 370–377, FRUCT Oy, 2016.

[28] I. V. Galov, A. A. Lomov, and D. G. Korzun, "Design of semantic information broker for localized computing environments in the internet of things," in *2015 17th Conference of Open Innovations Association (FRUCT)*, pp. 36–43, April 2015.

[29] S. Heiler, "Semantic interoperability," *ACM Computing Surveys (CSUR)*, vol. 27, no. 2, pp. 271–273, 1995.

[30] F. Viola, L. Turchet, F. Antoniazzi, and G. Fazekas, "C minor: a semantic publish/subscribe broker for the internet of musical things," in *2018 23rd Conference of Open Innovations Association (FRUCT)*, pp. 405–415, IEEE, 2018.

[31] P. Gearon, A. Passant, and A. Polleres, "SPARQL 1.1 Update." https://www.w3.org/TR/sparql11-update/, 2013.

[32] S. Harris and A. Seaborne, "Sparql 1.1 query language." https://www.w3.org/TR/sparql11-query/, 2013.

[33] C. Aguzzi, F. Antoniazzi, F. Viola, and L. Roffia, "SPARQL 1.1 Subscribe Language, Unofficial Draft," 2018. [Online; accessed 21-November-2018].

[34] D. Beckett, "RDF 1.1 N-Triples, A line-based syntax for an RDF graph, W3C Recommendation." https://www.w3.org/TR/n-triples/, 2014.

[35] "JSON-LD 1.1 Framing, W3C JSON-LD 1.1 Framing, An Extension to the Application Programming Interface for the JSON-LD Syntax, W3C Editor's Draft," January 2019. [Online; accessed 14-February-2019].