CITY IoT

Vipuvoimaa EU:lta 2014–2020

European Union
European Regional Development Fund

NGSI-LD

Otto Hylli

May 26, 2020

Contents

6Aika Uudenmaan liitto Nylands förbund OULU TAMPERE OULUN YLIOPISTO OAMK Tampereen yliopisto Tampere University

1 Introduction

In the CityIoT project we used open source components from the FIWARE project in building our IoT platform. FIWARE uses the NGSI-v2 context entity based data model and context broker API. A newer version of this API, called NGSI-LD, has been specified and implementation work is on going. The first version of the API specification was published last year (2019). This report explores this new specification: explaining what it is, how it compares to the current NGSI-v2 data model API and what benefits it may offer. This report assumes that the reader is familiar with FIWARE and the NGSI-v2 specification at least on a general level. The purpose of this report is not to be an indepth analysis of NGSI-LD, rather its purpose is to give an general overview and give the reader some ideas why they might or might not want to continue to investigate NGSI-LD more on their own.

Specification work for NGSI-LD happens under the European Telecommunication Standards Institute (ETSI). ETSI is a not-for-profit organization for developing standards for ICT systems and services. More specifically the NGSI-LD specification and other related material has been created by ETSI's Cross Cutting Context Information Management (CIM) industry specification group (ISG). On the group's web site the CIM group's purpose and goal is described with the following:

> "From the digitizing of industrial processes to creating smart services for citizens, it is essential to accurately record data together with its context information and to transfer these without misinterpretation to other systems. Single-purpose solutions work well within a known context but are not suitable for multi-system interoperability.
>
> Our mission is to make it easier for end-users, city databases, Internet of Things and 3rd-party applications to exchange information. Cross-cutting Context Information Management is the exchange of information, with proper formal definitions, between vertical applications, so that these applications get the original meaning.
>
> Our aim is to enable interoperable software implementations for Context Information Management. It is about bridging the gap between abstract standards and concrete implementations, especially for use cases related to Smart Cities, but also to be extended later to Smart Agrifood and Smart Manufacturing."

NGSI-LD combines concepts and ideas from the existing NGSI-v2 specification with the concept of linked data. Linked data is structured data that is distributed and linked together. It can be processed by machines and navigated through via links. Thus first this report shortly introduces the concepts behind linked data and the JSON-LD format that can be used to represent it. This information is needed to understand NGSI-LD. Then the report describes the NGSI-LD data model and API and compares them to the NGSI-v2. A quick

overview of possible NGSI-LD system architectures and current NGSI-LD implementations is also given. The next section contains some discussion about possible NGSI-LD benefits and issues. After that information about further reading is provided. Finally the last section contains conclusions. The sections about linked data, the NGSI-LD data model and API go into some technical details about their subject with the help of various examples. To help a reader who is not interested in those details and who wants just a general overview, these sections have been divided into an overview and example subsections. The example sections then can be skipped easily.

2 Linked data and JSON-LD

2.1 Overview

On the lowest level linked data consists of subject, property and value (or subject, predicate and object) triples. These triples then state information about resources or things. A single resource can be a subject in one triple and a property or a value in another triple. Values can also be value literals such as a text string or a number. The data thus forms a graph of information.

Linked data should also follow these four principles suggested by Tim Berners-Lee:

1. Use URIs to name (identify) things.
2. Use HTTP URIs so that these things can be looked up (interpreted, "dereferenced").
3. Provide useful information about what a name identifies when it's looked up, using open standards such as RDF, SPARQL, etc.
4. Refer to other things using their HTTP URI-based names when publishing data on the Web.

JSON-LD then is an expansion of the JSON format for representing the linked data graph or part of it as a tree-like structure. Most notable difference compared to regular JSON is that key names in JSON-LD are URIs. To keep the JSON still readable the used URIs can be mapped to short, human friendly terms. This mapping is done in a JSON-LD specific @context attribute. The use of URIs is meant for avoiding ambiguity when data from multiple sources is processed and combined.

2.2 Example

For example if the CityIoT project would like to release information about its publications in a machine readable form, this report could be described with a plain JSON like the following.

```
{
    "title": "NGSI-LD",
    "summary": "This report shows how the new NGSI-LD specification...",
    "author": "Otto Hylli"
}
```

This JSON uses keys invented for just this purpose so for some external system to utilize this information it would have to be told the meaning of each key. For example Google uses JSON-LD and other linked data formats in enriching its search results. An online store for instance can add a JSON-LD based describption of a product to its page for that product. Google can then show relevant information about the product such as price right on the search results page. So lets imagine that Google would show information about this report if it just could get it in the correct format. So if the project would like to use JSON-LD for this purpose, then it has to also use some existing vocabulary definitions that Google understands. The schema.org web site is one source for such vocabularies. One of the things it defines terms for is Report which is suitable for our case. The JSON-LD describing this report could look like the following:

```
{
    "@context": {
        "schema": "http://schema.org/",
        "projectReport": "schema:Report",
        "title": "schema:name",
        "summary": "schema:abstract",
        "author": { "@id": "schema:author", "@type": "@id" }
    },
    "@id": "https://cityiot.fi/reports/ngsi-ld",
    "@type": "projectReport",
    "title": "NGSI-LD",
    "summary": "This report shows how the new NGSI-LD specification...",
    "author": "https://cityiot.fi/people/ottohylli"
}
```

First of all it still has the same keys than the plain JSON. But now they are mapped to full URIs inside the @context object. So for example summary actually means `http://schema.org/abstract`. The idea being that we can still use whatever terms we want but now they are mapped to a common vocabulary that Google understands. Thus Google can for example find and show the abstract of our report. The value of the author property has changed. The Report type defines that value of the author property should be of the type Person. So now the value is an URL that should point to a JSON-LD description of the person who authored the report. Google then could also retrieve the contents of that URL and show information about the report author as well. Alternatively the Person object could just have been directly the value of the author attribute.

In addition to @context the example also has two other JSON-LD specific keys. @id is used to define the URI of the object described and @type gives its type which in this case is the schema.org Report type. The context also defines that the value of the author property is an URI indicated by defining its type to be an id. If we would be content to use the schema.org short terms instead of our own such as abstract instead of summary, we could just have referred to schema.org's own context definition by its URL with our own context attribute.

Since the example is linked data, it can also be presented as subject, property and value triples. This is shown in the table below. URIs are inside angle brackets and the following prefixes for URIs are used:

- schema: http://schema.org/
- cityiot: https://cityiot.fi/
- rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#[1]

Subject	Property	Value
<cityiot:reports/ngsi-ld>	<rdf:type>	<schema:Report>
<cityiot:reports/ngsi-ld>	<schema:name>	NGSI-LD
<cityiot:reports/ngsi-ld>	<schema:abstract>	This report shows ...
<cityiot:reports/ngsi-ld>	<schema:author>	<cityiot:people/ottohylli>

3 NGSI-LD

The NGSI-LD specification defines both a data model and an API for an context broker that uses the model. This section describes both the model and the API. It also explores the notable differences compared to the old NGSI-v2 model and API. Possible NGSI-LD architectures and the current NGSI-LD implementations are also shortly introduced.

3.1 Data model

3.1.1 Overview

The NGSI-LD data model is purposefully similar to the NGSI-V2 model. The intention is to make the models compatible and to make migration easy. The data model still consists of entities that can represent physical or more abstract things from the real world. Entities have an id and type which defines what the entity represents. The type defines attributes the entity has. Attribute values can be simple such as text or umber, more complex data structures like an address or a relationship to another entity. Domain specific data models can then be defined by specifying the required entity types and their attributes. The basic difference of course in representing the data is that NGSI-LD uses JSON-LD instead of plain JSON.

[1] Part of the W3C standard RDF schema vocabulary

In NGSI-v2 there were no specific compulsory requirements for the format of the id or type. In NGSI-LD as per linked data principles both are URIs. NGSI-LD makes the difference between attributes, whose value is a relationship to another entity, and other attributes more explicit. It has two kinds of attributes: properties and relationships. This attribute type is stated always in the entity JSON-LD representation.

The most notable difference between NGSI-v2 and NGSI-ld is how metadata is handled. In NGSI-v2 metadata such as attribute measurement timestamp or accuracy goes under a metadata key of an attribute. However NGSI-Ld actually does not contain the concept of metadata directly. Instead properties and relationships can have properties or relationships of their own. These then can be used to represent the additional information that went under metadata in NGSI-v2. NGSI-LD defines also some common metadata like properties such as *observedAt* for measurement time and *unitCode* for measurement unit.

NGSI-LD has one notable additional feature compared to NGSI-v2. It includes temporal representation of entity attributes. This means that NGSI-LD can represent the changing values of an attribute over time.

3.1.2 Example

To further illustrate how the NGSI-LD data model works and to highlight the differences to the NGSI-v2 model, we will use the ThreePhaseACMeasurement data model as an example. It is used to represent electrical measurements from a system that uses three-phase alternating current. It was developed in the CityIoT project and accepted as an official FIWARE data model.

Listings 1 and 2 show the same ThreePhaseAcMeasurement entity represented in the NGSI-v2 and the NGSI-LD data models. The entity has of course the common id and type. Then it has attributes for the measurement name, the start date of measuring, the cumulative nergy used since the start date and the active power measured for each phase. It has also one relationship to a Device entity that represents the energy meter which made the measurement. For the energy and active power the entity has the information when they were measured. In addition to that for the active power measurement the entity contains the information that the measurement is an average from one second.

The first thing to note about the NGSI-LD version is its context definition in the end. It uses JSON-LD context information from two sources:

1. `https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld`: NGSI-LD core context which defines the terms that are part of the NGSI-LD data model such as Property and RelationShip
2. `https://fiware.github.io/data-models/context.jsonld`: The context of the FIWARE data models that defines the terms used in the FIWARE domain specific data models such as the entity types and their attributes.

One small but perhaps confusing thing the NGSI-LD context defines is that it maps the JSON-LD @id and @type to the simple id and type. So the id

Listing 1: Example NGSI-v2 entity.


```
{
  "id": "ThreePhaseAcMeasurement:
         LV3_Ventilation",
  "type": "ThreePhaseAcMeasurement",
  "dateEnergyMeteringStarted": {
    "type": "DateTime",
    "value": "2018-07-07
              T15:05:59.408Z"
  },
  "refDevice": {
    "type": "Relationship",
    "value": ["Device:eQL-EDF3GL
              -2006201705"]
  },
  "name": {
    "type": "Text",
    "value": "ventilation"
  },
  "totalActiveEnergyImport": {
    "metadata": {
      "timestamp": {
        "type": "DateTime",
        "value": "2019-01-24
                  T22:00:00.173Z"
      }
    },
    "type": "Number",
    "value": 150781.96448
  },
  "activePower": {
    "metadata": {
      "timestamp": {
        "type": "DateTime",
        "value": "2019-01-24
                  T22:00:00.173Z"
      },
      "measurementType": {
        "value": "average"
      },
      "measurementInterval": {
        "value": 1
      }
    },
    "type": "StructuredValue",
    "value": {
      "L1": 11996.416016,
      "L2": 9461.501953,
      "L3": 10242.351562
    }
  }
}
```


Listing 2: Example NGSI-LD entity.


```
{
    "id": "urn:ngsi-ld:ThreePhaseAc
           Measurement:LV3_Ventilation",
    "type": "ThreePhaseAcMeasurement",
    "dateEnergyMeteringStarted": {
        "type": "Property",
        "value": {
            "@type": "DateTime",
            "@value": "2018-07-07
                       T15:05:59.408Z"
        }
    },
    "refDevice": {
        "type": "Relationship",
        "object": [
            "urn:ngsi-ld:Device
            :eQL-EDF3GL-2006201705"
        ]
    },
    "name": {
        "type": "Property",
        "value": "ventilation"
    },
    "totalActiveEnergyImport": {
        "type": "Property",
        "value": 150781.96448,
        "observedAt": "2019-01-24
                       T22:00:00.173Z"
    },
    "activePower": {
        "type": "Property",
        "value": {
            "L1": 11996.416016,
            "L2": 9461.501953,
            "L3": 10242.351562
        },
        "observedAt": "2019-01-24
                       T22:00:00.173Z",
        "measurementType": {
            "type": "Property",
            "value": "average"
        },
        "measurementInterval": {
            "type": "Property",
            "value": 1
        }
    },
    "@context": [
        "https://fiware.github.io/data
        -models/context.jsonld",
        "https://uri.etsi.org/ngsi-ld
        /v1/ngsi-ld-core
        -context.jsonld"
    ]
}
```


and type in the listing 2 are the same @id and @type that were used in the JSON-LD example in the previous section. Apart from the @context the high level structure of both entity representations are the same since both have keys for the entity id, type and attributes. Though in the NGSI-LD version the context maps the key names to URIs whose purpose is to avoid ambiguity. For example name actually means https://uri.etsi.org/ngsi-ld/name. So if another data model uses the same key or even different key that maps to that URI it means that the keys represent the same thing. The id of the NGSI-LD version is also an URI or more precisely an URN that uses the NGSI-LD URN scheme urn:ngsi-ld. The URN also contains the entity type. The type under the type key is also the same in both versions but again in the NGSI-LD version the context maps it into an URI: https://uri.fiware.org/ns/data-models#ThreePhaseAcMeasurement.

The values under the attribute keys are different. Though for example name has both type and value the meaning of type is different. In NGSI-v2 type tells the datatype of the value such as Text in this case. However in NGSI-LD the type is more of a type for the attribute which in this case is Property. The data type of the value can also be indicated if required. This has been done with the value of *dateEnergyMeteringStarted* attribute. The type Relationship for the *refDevice* is actually the same but meaning is different since it is the alternative for the Property type. Note also that the related entity id goes under object not value which seems to be another way to conceptually separate Property and Relationship from each other.

As described previously metadata is handled differently by using properties / relationships of properties / relationships. In the example the commonly used *timestamp* metadata attribute is replaced with the *observedAt* property. Its definition is part of the NGSI-LD core context. This is the reason why it is represented in a more simple way compared to the others. In the NGSI-v2 version the *activePower* has two additional metadata attributes: *measurementType* and *measurementInterval*. In the NGSI-LD version these become properties of the *activePower* property.

As mentioned previously NGSI-LD also allows temporal representation of entity attributes. This is done simply by providing a list of property or relationship instances for an attribute value with each instance using the *observedAt* property to indicate time. Listing 3 shows a *ThreePhaseAcmeasurement* entity which has a temporal representation of the *totalActiveEnergyImport* attribute with two values measured half an hour apart.

Listing 3: Example of a temporal representation of an entity attribute.

```
{
    "id": "urn:ngsi-ld:ThreePhaseAcMeasurement:LV3_Ventilation",
    "type": "ThreePhaseAcMeasurement",
    "totalActiveEnergyImport": [
        {
            "type": "Property",
            "value": 150781.96448,
            "observedAt": "2019-01-24T22:00:00.173Z"
        },
        {
            "type": "Property",
            "value": 150800.76841,
            "observedAt": "2019-01-24T22:30:00.428Z"
        }
    ],
    "@context": [
        "https://fiware.github.io/data-models/context.jsonld",
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ]
}
```

3.2 API

3.2.1 Overview

The actual API in NGSI-v2 and NGSI-LD are quite similar allowing the same kind of operations. Entities can be created, deleted and updated in various ways. Both also offer similar ways for querying entities and a way to subscribe

to entity change notifications. Both also allow other services to register as context sources. The NGSI-LD API naturally has also additional features for working with the temporal representations of attributes that are missing from the NGSI-v2 model and API.

3.2.2 Example

The use of JSON-LD inthe NGSI-LD API also requires some specific attention. The context has to be always clear when performing API operations. When sending JSON-LD it can be simply included as a part of the JSON payload in the @context attribute. This way the context broker can know the exact term URIs used. This has to be also done when querying information. Either the request must use the full URIs of the terms used or alternatively a HTTP link header, that contains an URL pointing to a context definition, can be used. So for example all ThreePhaseAcMeasurement entities can be fetched with either of the following curl commands:

```
curl -X GET \
  http://example.com/ngsi-ld/v1/entities/?type=ThreePhaseAcMeasurement \
  -H 'Accept: application/ld+json' \
  -H 'Link: <https://schema.lab.fiware.org/ld/context>;\
     rel="http://www.w3.org/ns/json-ld#context";\
     type="application/ld+json"' \

curl -X GET \
  http://example.com/ngsi-ld/v1/entities/\
  ?type=https://uri.fiware.org/ns/data-models%23ThreePhaseAcMeasurement\
  -H 'Accept: application/ld+json' \
```

Both use the same entities API endpoint with a *type* query parameter. The first can simply use the term ThreePhaseAcMeasurement as the value for the type, because it uses the lijnk header that includes a URL that points to the FIWARE data models context. The second has to use the full URI of the *ThreePhaseAcMeasurement* entity type since there is no context. Both also request a JSON-LD response with the *accept* header. They could also request plain json in which case the response would also have a link header containing a context URL.

The new temporal API features allow for example queries for the state of the entity between a given time interval. For example to get *totalActive-PowerImported* for the *ThreePhaseAcMeasurement* entity with id urn:ngsi-ld: ThreePhaseAcMeasurement:LV3_Ventilation between 22:00 and 23:00 on 24th of January 2019 the following curl command could be used:

```
curl -X GET \
  http://example.com/ngsi-ld/v1/temporal\
  /entities/urn:ngsi-ld:ThreePhaseAcMeasurement:LV3_Ventilation\
  ?attrs=totalActiveEnergyImported&timerel=between\
```

```
&time=2019-01-24:22:00:00Z&endTime=2019-01-24:23:00:00Z \
-H 'Accept: application/ld+json' \
-H 'Link: <https://schema.lab.fiware.org/ld/context>;\
   rel="http://www.w3.org/ns/json-ld#context";\
   type="application/ld+json"' \
```

The response then would contain something like the contents of listing 3.

3.3 Architecture

Though implementation details or architecture of a NGSI-Ld system are not in the main focus of the NGSI-LD specification, general guidelines for possible architectures are discussed in the specification and other related documents. The specification distinguish es three kinds of architectures:

1. **Centralized**: There is one NGSI-LD context broker. Context producers add and update entities to it and context consumers query or subscribe to it.

2. **Distributed**: Context sources implement query and subscription APIs an register themselves to distribution registry. Context consumer queries or subscribes to distribution broker which queries the registry for suitable context sources. Then the broker queries or subscribes to the sources. Consumer can also use registry and sources directly.

3. **Federated**: Similar to distributed except registration happens on a more coarse domain and scope level instead of enumerating each entity the source knows. The source also is a full context broker responsible for its domain. For example one registration could be streetlights in Tampere and another streetlights in Oulu.

Both NGSI-v2 and NGSI-LD support the registration of context sources required by the latter two architectures. However only NGSI-LD supports subscriptions from the distributed or federated sources.

3.4 Implementations

There are at least three ongoing NGSI-LD implementation projects in various states of development: Orion-ld, scorpio and Djane. Orion-ld and Scorpio are incubated FIWARE generic enablers meaning that they might become full FIWARE components but currently they are not yet ready. All of them are open source.

Orion-ld is based on the original FIWARE Orion context broker and shares code with it. The goal is to integrate it to Orion when it is complete. Currently it does not implement the whole NGSI-LD specification. It also seems that at least the temporal part of the API will not be implemented. Orion-ld is implemented with C / C++ and it uses the MongoDB database. It consists

of a standalone executable and its developers consider it suitable for small or resource constrained deployments.

Scorpio uses a microservice architecture. It is implemented with Java and the Spring Cloud framework. It uses PostgreSQL as its database and Apache Kafka for communication between the microservices. Unlike Orion-ld Scorpio also implements the temporal part of the NGSI-LD API meaning that it can be used to store and query also the entity history. Though currently the implementation does not cover everything in the temporal API specification.

Djane is implemented with Node.js and uses the MongoDB database. It offers some authentication and security features which the other components seem to lack. Its completeness level is not immediately apparent from its documentation.

4 Discussion and open questions

Though NGSI-LD is based on linked data principles it and the FIWARE models do not quite follow all of them. Namely the principles of using HTTP URLs and providing useful information when things are looked up. The entities are not identified with HTTP URLs but instead the NGSI-LD URN scheme is used. Both the NGSI-LD and the FIWARE models use HTTP URLs to define the used terms. However these URLs do not provide any information when used. Of course these are early days for NGSI-LD so this issue can very well be fixed in the future.

An important thing to realize about NGSI-LD is that its possible benefits will not be clear in a simple example or use case. Though some possible advantages and disadvantages can be noticed from a simple example as well. The extra context and use of URIs can feel like just unnecessary complexity. The use of attribute type to specify just property and relationship can also feel unnecessary. On the other hand the use of properties or relationships of attributes, which themselves can have properties and relationships, can offer new interesting ways for modeling data. The inclusion of temporal API endpoints can make dealing with entity history more convenient. We have encountered some issues with the current FIWARE approach of using a separate history component which is kept up to date using the NGSI-v2 subscription system.

However the idea is that the full potential of NGSI-LD will only be apparent in a more complex case where there is a distributed or federated architecture with many stakeholders, multiple different data sources and consumers. The data should also concern multiple domains and applications using the data should combine data from these domains. This kind of situation is supposed to be main use case for NGSI-LD. The linked data based data model is meant to ensure that the different operators can understand each other and be especially sure that they use the same terminology. In addition in NGSI-LD the more explicit separation between properties and relationships should make navigating between entities easier for machines. The possibility to use other linked data sources or linked data and semantic web technologies and tools such as the

SPARQL query language are also mentioned as benefits of NGSI-LD.

However it is not so clear what is required from the client software side or more generally from the people using the data, to take full advantage of the possibilities offered by NGSI-LD. For instance it is possible to document the entity relationships in a machine readable form. This can be done by using the @graph JSON-LD attribute as shown in the example below.

```
"@graph": [
   {
      "@id": "fiware:refDevice",
      "@type": "https://uri.etsi.org/ngsi-ld/Relationship",
      "schema:domainIncludes": [{"@id": "fiware:ThreePhaseAcMeasurement"}],
      "schema:rangeIncludes": [{"@id": "fiware:Device"}],
      "rdfs:comment": "The device which made the measurement.",
      "rdfs:label": "refDevice"
    },
    ...
]
```

It tells that the *refDevice*[2] relationship is between a *ThreePhaseAcMeasurement* and *Device* entity. It also gives an human friendly explanation and a label for the relationship. Note that currently the FIWARE models are not specified in this way but according to a GitHub issue this may be the way in the future. One FIWARE tutorial uses this method and discusses its advantages on a higher level. Still it is not clear what kind of application logic could fully utilize this information. Human understanding of the data model and requirements for the use of the data are still needed.

5 Further reading

Since this is just an overview, this section will point out sources for more information. Of course the NGSI-LD specification is the primary source for detailed information but it can be a bit overwhelming. The official NGSI-LD primer or the NGSI-LD white paper can be good introductions. There is also an official document presenting example use cases for NGSI-LD. For more practical hands-on material there is FIWARE tutorials for linked data basics and relationships with linked data which both use Orion-ld.

6 Conclusions

The aim of this report was to give a general overview of the new NGSI-LD data model and API specification. On a general level the data model and API are very similar to the currently used NGSI-v2 specification. The main difference

[2]This attribute has been named according to the old NGSI-v2 guidelines. For a native NGSI-LD data model better name for this attribute could be measuredBy or isMeasuredBy

being that NGSI-LD is based on linked data principles and uses the JSON-LD format. The core idea in this approach is that used terms are full URIs to avoid ambiguity. However the full benefits of NGSI-LD should be only apparent in a complex case including multiple data sources and consumers. When evaluating NGSI-LD it has to be kept in mind that as of this writing NGSI-LD is a new specification. This means that related implementations, tools, documentation and data models are in early stages of development. All of these are required for NGSI-LD to succeed and thus it is not possible to give a full evaluation of NGSI-LD since for this kind of system the state of the related ecosystem is very important.