Cross-Fertilizing Data through Web of Things APIs with JSON-LD

Wenbin Li and Gilles Privat

Orange Labs, Grenoble, France
gilles.privat@orange.com, liwb1216@gmail.com

Abstract. Internet of Things (IoT) data are mostly cloistered in closed IoT infrastructures and vertically integrated applications, failing to leverage the potential interlinking of corresponding APIs. We propose a data model based on JSON-LD, in which semantics are added to Web of Things (WoT) APIs, enhancing their interoperability and evolvability by the composition of nested contexts factoring out shared generic categories. According to this model, we present the blueprint of a framework to automatically and iteratively enrich and interconnect WoT APIs, through a process of resource crawl, syntax extraction, semantic annotation and context generation. Based on our propositions, we demonstrate the idea of cross-fertilizing data with a scenario involving three separate IoT infrastructures, showing how their data are linked and interoperated.

Keywords: Internet of Things, Web of Things, semantic interoperability, REST

1 Introduction

Many existing Internet of Things (IoT) infrastructures are little more than dedicated stovepipes that connect one set of sensor devices to a few applications, or watertight silos in exclusive mastery of a single operator, stakeholder (as for e.g. metering infrastructures) or manufacturer (as for the newish breed of "connected devices"). The provision of IoT services through APIs relies on such infrastructures, which vary widely in scope, scale, genericity, and the levels of abstraction of the data they provide access to. We refer the IoT APIs as Web of Things (WoT) [1] APIs if they adhere to the principles of the REST [2] architectural style, especially by using dereferenceable URIs instead of arbitrary identifiers and providing explicit interlinking between fine-grain resources such as devices, physical entities beyond devices (e.g., room or car) and the states and attributes of these.

A key challenge of WoT API development is the semantic interoperability of data exposed by APIs, which refers to the capability of not only matching data, but also automatically interpreting them. This challenge comes from the heterogeneity of IoT 'things" (from electronic devices to physical objects), the variety of their data models, the scarcity of explicit data descriptions, and the limited accessibility of data locked inside IoT infrastructures.

In response to this, we propose the idea to cross-fertilize data through WoT APIs exposed by existing infrastructures. By *cross-fertilize*, we mean "extract and propagate semantics to connect, exchange and explore data via semantic links, thus providing enhanced data understanding and uncovering additional knowledge". We first propose an RDF-based data model with JSON-LD [3], in which semantics are divided into three parts as JSON-LD contexts to support data interoperability, evolvability and semantics reuse. We then introduce the blueprint of a framework that makes it possible to extract and iteratively propagates semantics based on this model from IoT APIs. At last we present how data are cross-fertilized through a WoT-RDF infrastructure connecting data, ontologies and IoT infrastructures.

To illustrate our contribution, we present a smart building configuration containing a room and a corridor on the same floor managed by different IoT infrastructures. The room contains a presence sensor and an electrical door lock using FIWARE [4] and also contains a thermostat using Netatmo® [5], while the corridor contains a presence sensor using openHAB [6]. Thus three distinct and non-interoperable IoT infrastructures coexist on the same floor. In addition, there is a mismatch between these infrastructures since FIWARE does, as other high-level IoT infrastructures, describe physical entities at a level of abstraction above devices (a room in this example), while openHAB and vendor-provided APIs such as Netatmo® are lower-level infrastructures that give access to data only at the level of connected devices.

2 JSON-LD based Data Model

Following Linked Data principles [7], we propose a RDF data model based on JSON-LD. A JSON-LD description consists of at least three main parts [3]: 1) *@context*: a context description; 2) *@id*: an identifier; 3) *A JSON description*. We use JSON-LD rather than other RDF serialization formats such as RDF/XML for three main reasons:

Separation of lexical/syntactic and semantic levels. JSON-LD contexts [3] can either be defined within the same JSON-LD document or by referencing URIs containing context definitions. By referring URIs, the syntax and semantics in a JSON-LD description are separated into two documents to ensure the evolvability of model: whenever a semantic update is required, we simply update the context definition in a different URI without making any change in the JSON-LD description.

Context composition. JSON-LD supports the composition of several contexts for one document. By context composition, we are able to specify different abstraction levels for semantics included in a JSON-LD document, and promote the reuse of semantic definitions. Higher abstraction levels link data from different domains, while lower abstraction levels link data from the same subdomain. The complete JSON-LD context results from the composition of contexts in different levels.

Compatibility with JSON. JSON-LD is totally compatible with plain JSON, and the compatibility allows directly updating JSON-LD descriptions to IoT APIs based on JSON without making any further change in APIs side.

To take full advantage of JSON-LD, we specify three levels of abstraction for JSON-LD contexts. In particular, the value of the "@context" is defined as a URI of a document containing the context definition by composing contexts from different abstraction levels. In the following, we introduce each abstraction level.

Generic IoT Context: Generic IoT context defines common concepts (e.g., thing vs. device, location, time, name, attribute) shared by all domains of IoT. A JSON-LD document refers to only one such context.
Objective: Generic IoT context is used to match and link data from different domains through the common concepts used, if any.
Reference Ontologies: Generic IoT context adopts standard domain-independent ontologies such as OneM2M ontology [8]. All IoT data adopt the same references in Generic IoT context.
Domain-specific Context: Domain-specific context defines vocabularies (e.g., SoilHumiditySensor) related to the subdomains of IoT (e.g, Smart Agriculture, Smart Cities and Smart Home). A JSON-LD context can combine one or more domain-specific contexts (e.g. smart energy and smart buildings) since possible it corresponds to devices and data from different subdomains.
Objective: Domain-specific context is used to match and link data from the same subdomain.
Reference Ontologies: Domain-specific context references standard domain-dependent ontologies such as SAREF [9] for smart home domain, or CityGML [10] for smart city domain. All data of the same domain adopt the same references in domain-specific context.
Vendor/Technology-specific Context: Vendor/Technology-specific context provides references for specific terms related to device manufacturers (e.g., Philips® and Netatmo®), protocols (e.g., ZigBee and Z-Wave), or technologies (e.g., CoAP). A JSON-LD context can combine one or more vendor//technology specific contexts since it possible contains data from different vendors/technologies.
Objective: Vendor/Technology specific context aims at mapping vendor/technology specific terms with generic IoT and domain-specific ontologies to link data.
Reference Ontologies: Vendor/Technology specific context firstly maps vendor/technology specific vocabularies with generic IoT or domain-specific reference ontologies; if mapping relation cannot be found, ontologies (e.g., Z-Wave Ontology [11]) related to specific manufactures, protocols or technologies are used; if no semantic reference exists for certain concept, ontologies are created by domain experts.

For illustration purpose, listing. 1 presents a simplified example of Netatmo® thermostat data from our scenario, in which the URI in the example represents our WoT-RDF infrastructure introduced in section 3. A detailed description of our scenario with all corresponding entities is provided in [12]. The left column presents the JSON-LD description based on our data model and three URIs for contexts, and the right column illustrates the three genericity levels of JSON-LD contexts used. By just adding two lines (i.e., @context and @id) to the basic JSON description provided by the Netatmo® API, we transform it into semantically meaningful JSON-LD. The complete context for the building, shared by all entities, is composed from three contexts, i.e.,

Netatmo, smart home, and generic IoT; when the context requires to be updated to take into account other concepts, we simply modify or add contexts at the required level without changing the building context itself.

URI: http://lab.wot-rdf.org/jsonld/thermostat1
{
"@context":"http://lab.wot-rdf.org/jsonld/context/building",
"@id": "http://lab.wot-rdf.org/jsonld/thermostat1",
"status": "ok",
"body": {
"temperature": 21.7,
"unit": "Celsius"},
"module_name": "Inside",
"rf_status": 161}

URI: http://lab.wot-rdf.org/jsonld/context/building
{"@context": [
"http://lab.wot-rdf.org/jsonld/GenericIoTContext",
"http://lab.wot-rdf.org/jsonld/SmartHomeContext",
"http://lab.wot-rdf.org/jsonld/NetatmoContext"]}

URI: http://lab.wot-rdf.org/jsonld/GenericIoTContext
{"@context": {
"m2m": "http://www.onem2m.org/ontology/Base_Ontology/"
"status": "m2m:hasOperationState",
"body": "m2m:hasOutput",
"unit": "m2m:concerns" }}

URI: http://lab.wot-rdf.org/jsonld/SmartHomeContext
{"@context": {
"saref": "http://ontology.tno.nl/saref#",
"biopax": "http://www.biopax.org/release/biopax-level3.owl#",
"temperature": "biopax:temperature" }}

URI: "http://lab.wot-rdf.org/jsonld/NetatmoContext
{"@context": {
"schema": "http://schema.org/",
"m2m": http://www.onem2m.org/ontology/Base_Ontology/,
"module_name": "m2m:isPartOf",
"rf_status": "netatmo:radioStatus" }}

Listing 1. Thermostat example

3 Semantics Extraction and Propagation Framework

We present the framework of semantics extraction and propagation, which is designed to generate semantic data based on the previous model from WoT APIs. The framework is illustrated in Fig. 1.

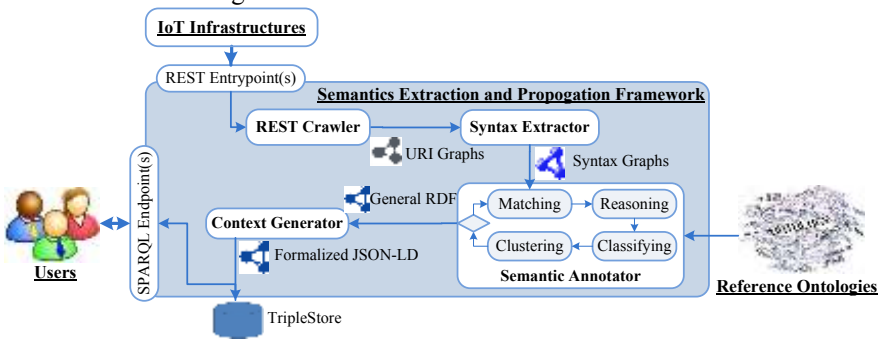

Fig. 1. Semantics extraction and propagation framework

Generally, starting with the input of REST entry points from different IoT infrastructures, the framework consists of four modules i.e., REST Crawler, Syntax Extractor, Semantic Annotator and Context Generator to generate JSON-LD documents. Finally RDF triples represented in JSON-LD are stored in a triplestore that provides a SPARQL endpoint for user and application queries.

3.1 REST Crawler

Most of IoT infrastructures expose REST APIs for user queries. In order to fully explore the APIs, a REST crawler is expected to automatically discover REST resources by following links from REST entry point(s). According to the design principle HATEOAS of REST, REST resources descriptions must have well-defined ways in which they expose links to related resources [2]. A number of formats define relations between resources to support HATEOAS in REST design, and their discovery strategies are introduced in [13].

In our framework, the REST crawler applies a recursive process of identifying the format of resources, extracting relationships from resource descriptions and generating a RDF graph connecting different resources identified by URIs.

In our scenario, all three distinct IoT infrastructures use REST as their architectural style. Fig. 2 presents the entry points and URI graphs constructed by REST crawler.

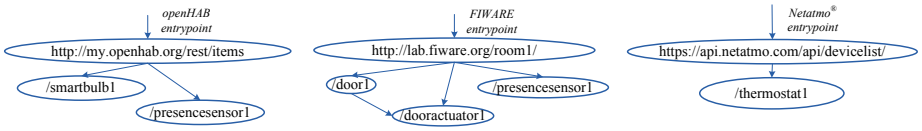

Fig. 2. URI graphs from three IoT infrastructures

3.2 Syntax Extractor

Syntax extractor extracts information from discovered resource descriptions and creates from these a stub RDF graph for future semantic annotation.

More than one way transforms hierarchical serialization formats into graphs, and here we present a recursive method to generate a RDF graph from JSON-based resource description. Other serialization formats are firstly transformed into JSON and then are processed in our framework. JSON is built on two base structures, either *a collection of key/value pairs* or *an array*. The extraction process is introduced as follows: 1) A first subject is generated by use of the JSON-based resource URI; 2) for a collection of key/value pairs, the keys are used to generate predicates in RDF graph while the values are regarded as the objects. In case that the value of a key *k* is another JSON object *obj* instead of a simple data type, an anonymous node *anode* is created as the object of the key *k*, and *anode* is the subject of the key/value pair of *obj*; 3) for an array of values, the predicate is regarded as "rdf: predicate", while the value elements in array are RDF objects. 4) An elimination algorithm introduced in [14] is applied to reduce the redundancy of the RDF graph.

3.3 Semantic Annotator

In order to provide semantics for WoT APIs, semantic annotator updates the stub RDF graphs from syntax extractor by associating RDF elements i.e., nodes and arcs, with three levels of context reference ontologies. The semantic annotator proceeds with an iterative process of the following four steps.

Keywords Matching. The descriptions of RDF elements are matched against ontologies elements i.e., classes and properties. We adopt the semantic matching algorithm introduced in [15] because of its performance on heterogeneities and inconsistence matching. The ontology alignment algorithm in [16] is applied to deal with multiple matching between one graph element and ontology elements.
Semantic Reasoning. This step firstly infers the classes and properties in RDF graphs based on ontology property's domain and range: for a RDF statement, if the predicate between two RDF nodes is identified, the classes of the subject and object can be inferred based on the domain and range of the predicate; equivalently for the contrary case. Secondly, this step infers RDF graph elements following semantic rules which come from the axioms in ontology definitions.
Classifying. This step analyzes graph elements' connections with other elements and use pre-trained classifier to refine RDF element. Here we adopt the ontology classification algorithm introduced in [17] to create the classifier based on Maximum Entropy Markov Model due to the optimum performance for class induction.
Interlinking. This step discovers RDF graph elements that represent the same concept by use of clustering algorithms to refine RDF graph elements within the same cluster. Here we use the approach presented in [18] to calculate the distance between RDF graph elements and carry out interlinking step.

Each internal step runs for sequent. At the end of one cycle, the generated RDF graph is sent to the first step to start a new cycle, because certain graph elements deduced by latter steps can possibly be used by previous steps to identify graph elements. The RDF graph automatically and incrementally propagates through such iteration. The iteration stops when the graph has not changed from the previous iteration.

Fig. 3 presents the simplified output graph of semantic annotator, while a detailed graph is presented in [12]. The ontologies used are OneM2M ontology [8] and DogOnt [19]. Through SPARQL endpoints, we are able to get additional information such as the presence state of the whole floor.

Fig. 3. Output RDF graph

3.4 Context Generator

In order to transform RDF graphs into JSON-LD documents, context generator transforms the result from semantic annotator to formalized JSON-LD documents with three abstraction levels of contexts as defined before.

Since tools exist to transform RDF between different serialization formats [20], here we only present the transformation process from RDF semantics into three levels of JSON-LD context. The generation process is a top-down matching process as follows: for a RDF graph element with semantic label, if a matching exists between the semantic label and the reference ontologies of generic IoT context, this label is stored in the generic IoT context part; otherwise, context generator checks if a matching exists between the semantic label and the reference ontologies of domain-specific context. If yes, this label is stored in the domain-specific context part; otherwise, it is stored in the part of vendor/technology specific context. This process repeats until all semantics in a RDF graph is transformed as JSON-LD context.

After generation, IoT data from different sources are connected by the generic IoT context, and IoT data from the same subdomain are also connected through the domain-specific context.

Fig. 4 summarizes the idea of cross-fertilizing IoT data based on our propositions. Semantics extraction and propagation framework connects WoT-RDF infrastructure and IoT infrastructures by constructing RDF graphs from IoT REST interfaces and generating JSON-LD descriptions based on our data model. WoT-RDF infrastructure provides two views for the linked IoT data: general RDF graphs for SPARQL query, and JSON-LD documents to connect IoT infrastructures. Three parts of ontologies respectively provides semantics in three abstraction levels.

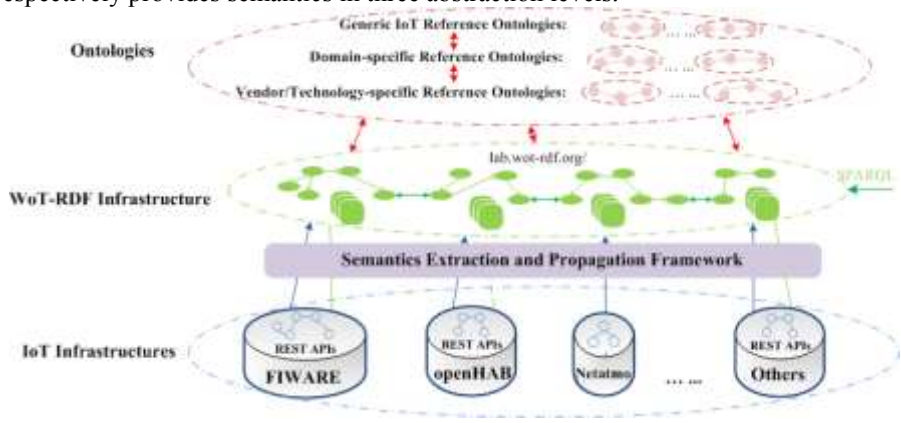

Fig. 4. Cross-fertilizing IoT data with JSON-LD

Regarding queries, users are able to not only perform local queries through REST APIs of individual IoT infrastructures, but also query through SPARQL endpoints in WoT-RDF infrastructure to get global information between IoT infrastructures and deduce explicit knowledge. Moreover, JSON-LD documents are also stored in WoT-RDF infrastructure and are used to update IoT infrastructure-side descriptions (for open IoT infrastructure such as FIWARE). By such updates users are able to locally perform queries via individual infrastructure APIs to get additional semantics and interconnected information.

4 Conclusion

JSON-LD is a promising evolutionary solution to cross-fertilize data through existing WoT APIs, maintaining full backward compatibility if needed. A framework has been presented to generate, by iterative propagation, specified JSON-LD data from WoT interfaces, and a WoT-RDF "superstructure" has been introduced to interoperate existing WoT infrastructures from the semantic level. This idea has been proposed as a solution for further opening and "semanticizing" the APIs of FIWARE, which is itself a high-level open infrastructure, but it can be taken up, as we have shown as a minimal pragmatic interoperability solution between existing platforms or infrastructures, taken as they are, without the need to subsume, replace or even alter them.

References

1. Guinard, D.D., Trifa, V.M.: Building the Web of Things Book | Web of Things. Manning Publications (2015).
2. Richardson, L., Ruby, S.: RESTful web services. O'Reilly, Farnham (2007).
3. JSON-LD 1.0, https://www.w3.org/TR/json-ld/
4. Internet of Things (IoT) Services Enablement Architecture, https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Internet_of_Things_%28IoT%29_Services_Enablement_Architecture
5. Netatmo Developers, https://dev.netatmo.com/doc
6. openHAB, http://www.openhab.org/
7. Bizer, C., Berners-Lee, T.H.T.: *Linked data - the story so far*. Int. J. Semant. Web Inf. Syst. (IJSWIS) 5(3), 1-22 (2009)
8. OneM2M Base Ontology, http://www.onem2m.org/ontology/Base_Ontology
9. SAREF Ontology Documentation, http://ontology.tno.nl/saref/
10. CityGML, http://www.citygml.org
11. Z-Wave ontology - Smart Appliances Project, https://sites.google.com/site/smartappliancesproject/ontologies/z-wave-ontology
12. Interoperability with JSON-LD, https://github.com/WoTRDF/InteroperabilityWithJSON-LD
13. Mayer, S., Guinard, D.: An Extensible Discovery Service for Smart Things. In: Proceedings of the Second International Workshop on Web of Things. No. 7. ACM (2011).
14. Pichler, R., Polleres, A., Skritek, S., Woltran, S.: Redundancy Elimination on RDF Graphs in the Presence of Rules, Constraints, and Queries. In: Web Reasoning and Rule Systems. pp. 133–148. Springer (2010).
15. Khan, S., Safyan, M.: Semantic Matching in Hierarchical Ontologies. Journal of King Saud University - Computer and Information Sciences. 26, 247–257 (2014).
16. Cheatham, M., Hitzler, P.: String similarity metrics for ontology alignment. In: The Semantic Web–ISWC 2013. pp. 294–309. Springer (2013).
17. Gandon, F., Cabrio, E., Stankovic, M., Zimmermann, A. eds: Semantic Web Evaluation Challenges. Springer International Publishing, Cham (2015).
18. Ferrara, A., Genta, L., Montanelli, S.: Linked data classification: a feature-based approach. In: Proceedings of the Joint EDBT/ICDT 2013 Workshops. pp. 75-82. ACM (2013).
19. DogOnt, http://elite.polito.it/ontologies/dogont/dogont.html
20. JSON-LD implementation for Java, https://github.com/jsonld-java/jsonld-java