Graph-based Semantic Evolution for Context Information Management Platforms

Wenbin Li
Easy Global Market
Sophia Antipolis, France
wenbin.li@eglobalmark.com

Gilles Privat
Orange Labs
Grenoble, France
gilles.privat@orange.com

José Manuel Cantera
FIWARE Foundation
Berlin, Germany
josemanuel.cantera@fiware.org

Martin Bauer
NEC Laboratories Europe
Heidelberg, Germany
martin.bauer@neclab.eu

Franck Le Gall
Easy Global Market
Sophia Antipolis, France
franck.le-gall@eglobalmark.com

***Abstract*—**Context Information Management (CIM) platforms have tended to rely on mostly hierarchical information models with weak semantics and predefined APIs, falling short of the requirements for interoperation with other platforms and flexibility of data utilization and federation. In order to improve cross-domain federation capabilities for existing CIM platforms, we propose an evolution framework by combining the "property graph" information model with RDF-based semantic modeling. We describe this evolution for what concerns: context information modeling, context information representation, query patterns and architecture. For each of these aspects, we introduce the problem space, the evolution strategy and provide examples based on existing popular CIM platforms. This evolution framework is designed to meet the requirement of backward compatibility to existing CIM platforms, while bringing graph-based semantic evolution based on the property graph model for CIM interoperability and cross-domain federation. This framework is currently evolved by the ETSI Industry Specification Group on "cross-cutting Context Information Management" (ETSI ISG CIM) for standardization.

***Index Terms*—Context Information Management, Internet of Things, Web of Things, Information Model, Property Graphs, Graph-oriented databases, RDF, JSON-LD, Semantic Query**

I. Introduction

The ongoing deployment of mostly closed, fragmented and proprietary IoT platforms has made the federation of information and high-level queries cutting across such IoT platforms challenging. An intermediate generation of Context Information Management (CIM) platforms [1] have appeared to meet the challenges by combining IoT platform functions with efficient APIs to support cross-domain information federation and flexible queries. The main objectives of a CIM platform are to provide common information models to federate data from different sources and to support efficient and flexible queries with user-friendly APIs. Operating upon lower-level IoT platforms, these CIM platforms have the potential to start an architectural evolution from siloed-centralized to distributed and federated IoT architectures.

Nevertheless, most of the existing CIM platforms such as FIWARE in its NGSI [2] version, still rely on hierarchical data models and predefined query patterns for data federation and utilization. Although they are able to support cross domain information management, two main limitations remain as: the information model lacks proper semantics for interoperability; the query patterns are predefined with little flexibility to support new query requirements.

In response to this, we propose in this paper an evolution framework for existing CIM platforms to enhance their information models and APIs towards semantics and graph-modeled interlinking of their informational resources. We describe this evolution under four aspects: context information model, context information representation, query patterns and architectures. This evolution strategy is based on the requirement to maintain backward compatibility to existing CIM platforms and reuse as much as possible existing specifications.

The remainder of the paper is organized as follows: Section 2 presents the general evolution framework; sections 3 to 6 respectively introduce the evolution strategy for context information model, context information representation, query patterns and architecture; section 7 concludes our contribution and sheds light on future perspectives.

II. Evolution Framework

The evolution framework for CIM platforms is presented in Fig. 1 with four levels from the bottom up as relevant to context information model, context information representation, query patterns and architecture. The framework starts with the evolution of context information models from hierarchical to graph-based models to provide a linked-open-data-friendly information model and support semantics; then the context information representation evolves from ad hoc or legacy serialization such as JSON to generic RDF serialization such as JSON-LD, to support fully open semantic interoperability; the query patterns for data utilization is improved by triple patterns from predefined fixed patterns for flexible data query; finally the architecture based on context information models and query patterns are driven to distributed and federated ones to support

cross domain federation. The evolution strategy for each level is detailed in the following sections.

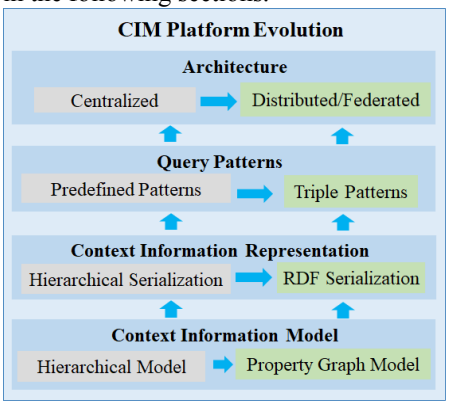

Fig. 1 CIM evolution framework

In order to make the explanation of each level more concrete, we use throughout this article a simple example from a piece of context information within a building. More detailed examples can be found in [3]. Fig. 2 is a graph representation of this example, where filled shapes represent two kinds of graph nodes (entities are represented by rectangles; literal values are represented by hexagons.), and unfilled shapes represent two kinds of graph edges (relationships between two entities are represented by diamonds; properties linking an entity, relationship or other property to a value are represented by ovals.)

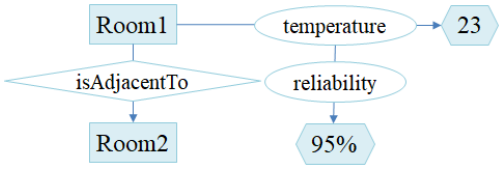

Fig. 2 Simple property graph

The example describes two adjacent entities *Room1* and *Room2*, the entity *Room1* has one *temperature* property with value 23, and the *temperature* property is further qualified by another property, *reliability,* with value 95%.

III. Evolution of Context Information Models

In the first wave of context-awareness research dating back to the 2000s [4], context used to be mostly, and implicitly, user-centric, typically capturing e.g. the activity or location of mobile users to adapt services offered to them. We are interested here in a broader definition of context, adapted to IoT and linked data platforms, where the very notion of context is de-centered and relative. The role of a CIM platform is to mediate and consolidate data gathered from multiple heterogeneous data sources (IoT networks or lower-level platforms, open data/linked data repositories, crowdsourced data, but also proprietary closed databases) to expose them to applications. In this perspective, the primary data of one application may be the context of another, and vice versa.

Viewing context as a graph is the most general and best-adapted possible data model for this kind of context information. This view aligns with the grand evolution of the Web towards linked open data (the Web of data) in a way that we will elicit further.

An early and widely publicized definition of context [5], dating from the first generation of context research, was "any information that can be used to characterize the situation of an entity". It may in fact remain valid in the broader context we outlined before if we represent entities as the nodes of a graph. Rather than through a vague notion of situation, we define context as the set of properties characterizing these nodes, together with the set of relationships that enmesh them together, and the properties of these relationships. Context is, thus decentered and broadly defined, the graph itself.

This information model, known as "property graphs" [6], or "attributed graphs" [7] has become a de facto standard from its widely shared use in graph databases. It could be characterized as standing midway between traditional object-oriented or entity-attribute-value information models and purely semantic models such as RDF.

A. Beyond hierarchical context: property graphs as a universal distributed context information model

Traditional (mono-centered) context fitted rather well a classical object-oriented or key-value hierarchical description, with a set of more or less detailed context features attached to a single entity (such as from the early definition in [5]). The multi-centered notion of context we address here requires breaking this rigid hierarchical structure and moving towards a more expressive, flexible and adaptable graph model, the only one fit to capture the complex web of looping and center-less inter-entity relationships that make up context information in the sense we defined it here. This information needs not be semantically defined from the outset: it may be natively structural information, capturing e.g., containment of adjacency relationships that define location in a most often implicit set-theoretic model [8], and the semantics of this context may get added onto it in a later stage of graph enrichment [9]. This model does also happen to fit the natively distributed nature of context data sources we are aiming to capture.

B. Bridging property graphs and RDF graphs

Superficially, RDF graphs could be seen as just an alternative way to capture context information cast as a graph, but they start from a very different premise, placing the emphasis on semantic rather than structural information, mixing individual-to-class (instance to category) with individual-to-individual semantics. RDF is a supremely parsimonious meta-model, defining only, from the theoretical grounding of first-order predicate logic, a basic <*subject* → *predicate* →*object*> triple as building block of a labeled graph, where a generic notion of property (instantiated in a predicate) does not distinguish relationships in the way property graphs do it (though OWL does make a somewhat similar distinction between object properties and datatype properties). RDF and the associated ontology languages (RDFS/OWL) do natively support all kinds of semantics but they cannot directly express more complex constructs that go beyond the expressivity of first-order logic, such as properties of relationships and their derivatives, which a graph database natively supports. Property

graphs are the implicit semi-formal common denominator data model underlying most graph databases, such as Neo4J, OrientDB, etc. They have gained widespread following as such, more in industry than in academia. They make it possible to attach properties to relationships, which RDF does not directly support, but they lack the standardization and formal underpinnings of RDF and do not interoperate directly with linked data and other RDF datasets. Also, they do not lend themselves to reasoning with RDF-based reasoning tools or querying with standard query languages such as SPARQL.

Several solutions have been proposed transform property graphs into RDF graphs. Reification in the RDF sense, i.e. morphing a statement into a resource, is the default way to support this transformation. Standard RDF reification [10] explicitly defines a new resource (with type *rdf:statement*) that encompasses a predicate jointly with its associated subject and object (i.e. a triple) This new statement resource is then linked back to the original subject, object and predicate of the statement through a trio of properties with types *rdf:subject*, *rdf:object* and *rdf:predicate*, respectively. A total of 4 additional statements (the infamous "reification quad") are thus required to fully define the reified statement as a resource, this only in order to make this resource the subject of other statement. This is a very cumbersome and heavyweight solution. Reification by way of blank nodes is a simpler way to achieve this. Assuming we want to reify this statement from our example:

<Room_1 → temperature → 23>

in order to make it the subject of another statement:

<<statement_1> → reliability → 95%>,

we just have to add a blank node to obtain an RDF-reified equivalent of our example property graph with three triples as:

<Room1 → temperature → _blank_node>
<_blank_node → reliability → 95%>
<_blank_node → value → 23>

This solution is especially convenient when the graph is serialized with JSON-LD (see following section) because blank nodes do not explicitly appear in the textual serialized description, and actually show up only when it is represented as a graph. It is thus possible for a developer to generate the JSON-LD payload of an API in a form that is very similar to what he would have generated in plain JSON, or in the previous version of the NGSI data model and API [2].

IV. Evolution of Context Information Representation

Once the graph-based information model for context is adopted, the next point to be handled is the representation of this model to be conveyed through a CIM API. There are two main challenges to be faced: 1) being expressive enough to capture all the richness of the underlying graph-based information model; 2) appealing to developers particularly in terms of consumption simplicity of consumption and tool availability. Since XML and JSON are the most popular serialization formats for existing CIM platforms, we mainly analyze these two formats.

During the first decade of the 21st century XML-based representation formats were largely used. The main advantages of XML are its hierarchical structure together with its capability of being both human and machine readable. Early implementations of Context Information Management like OMA NGSI [11] used XML and HTTP bindings. Soon after the large adoption, XML drawbacks have been observed as verbosity, lack of strong data types and absence of native support in the most popular programming languages. Soon, new standards appeared, such as XML Schema or XML Namespaces, trying to overcome the main issues posed by XML. However, developers considered these new standards as an extra layer of complexity.

Following this, the JSON format emerged and gained wide acceptance because of simplicity (based on name-value pairs), native support offered by programming languages, particularly JavaScript, and capability of seamlessly representing structured types with expressive type systems for strings, numbers or booleans. Along with the success of REST Web services, JSON and REST have become the most suitable pair of technologies to design RESTful APIs for CIM platforms. Most remarkably, the FIWARE NGSI API is recommended by GSMA [12] and TMForum implemented by several smart city projects in Europe. In NGSI API, JSON representation of context entities is done through JSON objects which contain an entity identifier, an entity type and a number of context attributes (properties). Likewise, context attribute values are encoded as JSON member values using a proper JSON data type. Optionally, a nested JSON object structure for encoding the so-called metadata (properties of properties) can be used.

However, XML and JSON are outdated to meet modern requirements of context information representation for two reasons. Firstly, no standardized mechanism exists for referencing the semantics of the represented data in either format. This is critical to certain domains such as smart cities, where a high degree of harmonization, interoperability and future regulation is expected, and thus results in the isolated CIM platforms without interoperability. Secondly relationships between entities cannot be expressed explicitly. In fact, context entities are strongly interrelated between each other. For instance, there is a relationship between parking houses and the vehicles inside; vehicles have relationships with their owners. In this case, context information forms a graph of data which has to be serialized using a proper encoding format, leveraging XML and JSON.

To address the two limitations, RDF/XML [13] and JSON-LD [14] have been proposed as semantic evolutions of XML and JSON. RDF/XML and JSON-LD are both designed to associate the represented data with machine understandable ontologies and capture relations between the represented data following linked data principles. Since RDF/XML reuses the XML structure to keep backward compatibility, RDF/XML shares the same aforementioned drawbacks of XML. Alternatively, JSON-LD is 100% JSON compatible with extra support of data semantic interoperability, evolvability and semantics reuse [15]. JSON-LD is being progressively adopted by developers and webmasters, and recommended by search engines to attach semantics to web pages with the schema.org harmonized vocabulary [16].

An example of the JSON-LD evolution for FIWARE NGSI representations is presented in Fig. 3.

```
{
 "@context": "http://vocabulary.example.org",
 "id": "urn:example:Room:Room1",
 "type": "Room",
 "temperature": {
   "type": "Property",
   "value": 23,
   "reliability":  {
      "type": "Property",
      "value": "95%"
    }
  },
"isAdjacentTo": {
   "type": "Relationship",
   "object": "urn:example:Room:Room2"
 }
}
```

Fig. 3 JSON-LD Evolution of FIWARE NGSI

In Fig. 3, the JSON-LD @context maps each term to a URI that uniquely identifies it in a target vocabulary. JSON-LD contexts can either be defined within the same JSON-LD document or by referencing URIs containing context definitions. By referencing URIs, the syntax and semantics in a JSON-LD description are separated into two documents to ensure the evolvability of the model [15]. Moreover, a JSON-LD @context can be composed of one or more sources. By context composition, we can specify different abstraction levels for semantics in a JSON-LD document, and promote semantics reuse. In this example, the context is composed of core terms (to convey the common vocabulary for providing the context of data, such as location, time or unit codes) and specific terms defined by each particular vocabulary, for instance "temperature" or "Room" in the example provided below.

Furthermore, the Properties and Relationships are differentiated as well by the standard JSON-LD type (mapped to @type in the context) keyword. Property nodes convey the value of a property by means of the "value" term; while relationship nodes convey the target of a relationship through the "object" term. The representation in Fig. 3 follows the property graph model and is reified by way of a blank node, as explained in the previous section to show the property "reliability" is the property of the property "temperature" provided by a sensor.

For the evolution of context information representation of existing CIM platforms, RDF/XML can be used to keep XML backward compatibility of a platform; JSON-LD provides more powerful options and is designed to be easily integrated into existing CIM platforms using JSON.

V. Evolution of Query Patterns

In our framework, context information based on the property graph model is represented by use of RDF-based serialization formats to provide a standard way of structuring data. To fully make use of the graph data model and federated RDF data, the evolution of CIM query interfaces towards semantics is expected to also provide users with more flexible query patterns for data utilization and knowledge discovery. Traditional queries on hierarchical model-based data rely on hard coded APIs to achieve predefined query patterns. When new query requirements expect patterns that are not supported by the current platform, the APIs have to be extended to take into account new requirements. The query design thus is not scalable and lacks the capability to interoperate with other platforms supporting different query patterns. On the contrary, the standard query patterns for RDF data, i.e., triple pattern queries, are able to answer any type of questions so that CIM platforms can easily support various query requirements and federate different existing platforms.

We use *s, p, o* to respectively denote the subject, predicate and object of an RDF statement. The triple patterns that a CIM platform could benefit from consists of 1-wise, 2-wise and 3-wise patterns that respectively query 1, 2 or 3 elements of a statement, as presented in Table 1.

Table 1 Triple Query Patterns

	data: <s, p, o>		
1-wise	<**s**, p, o>	<s, **p**, o>	<s, p, **o**>
	<Room1, ?, ?>	<?,isAdjacentTo,?>	<?, ?, Room2>
2-wise	<**s**, **p**, o>	<**s**, p, **o**>	<s, **p**, **o**>
	<Room1,temperature,?>	<Room1,?,Room2>	<?,temperature,23>
3-wise	<**s**, **p**, **o**>		
	<Room1, isAdjacentTo, Room2>		

In Table 1, seven query patterns are identified as basic triple query patterns, and the blue boldface letters represent the elements that a query pattern addresses. For each pattern, we present a query example, e.g., in the 1-wise pattern where we only indicate any subject *s*, we can query all information related to a subject which is the room Room1, and by such query all triples related to Room1 will be returned. Particularly, in the 3-wise pattern, we can query if a specific statement exists in the database, and the returned result is the same triple if the specific statement exists, otherwise no result is returned.

The evolution steps from a hierarchical query pattern to a triple pattern within the CIM platforms are proposed as follows:

1. Gap Analysis. The first step analyzes the gap between the query patterns supported by the platform with the seven triple query patterns. The triple query patterns that are not supported by the current platform (e.g., query on any statement's object in a data set is not supported) or partially supported by current platform (e.g., query on a specific subject type is supported, but query on any other subject types is not supported) will be recorded as gaps.

2. Extension Implementation. The second step involves the implementation of the identified gaps in current query interfaces as an extension to existing CIM platforms. Particularly, the implementation can contain a partial or full list of the identified gaps to support CIM architectures discussed in the next section for efficiency reasons. Ideally, once the extension is implemented, the CIM platforms support all triple pattern queries and are able to answer any questions by combing different triple queries.

Taking the FIWARE NGSI [2] interface as an example, the gap analysis result is presented in Table 2. We can see that 3 of 7 patterns in Table 2 are supported by FIWARE NGSI. The identified gaps help to understand the enhancements to be considered for future versions of NGSI interfaces so as to provide users with more flexible query interfaces.

Table 2 Gap Analysis of FIWARE NGSI

Query Patterns	Data: <s, p, o>		
1-wise	<**s**, p, o>	<s, **p**, o>	<s, p, **o**>
	/entities/{EntitiyID}	no	no
2-wise	<**s**, **p**, o>	<**s**, p, **o**>	<s, **p**, **o**>
	/entities/{EntitiyID}/atts/{attributeName}	no	no
3-wise	<**s**, **p**, **o**>		
	/entities?filter		

By use of the graph context information model and triple pattern queries, the basic interoperability among CIM platforms is ensured, and the distribution or federation of CIM platforms via corresponding architectures becomes possible.

VI. Evolution of Architecture

The evolution of data model and data query patterns further drives the CIM architecture evolution from a centralized one to a distributed or federated one. Simple IoT use cases can be handled by a centralized IoT architecture as depicted in the left part of Fig. 4. Such scenarios are characterized by a deployment in a specific domain, typically by a single provider in charge of everything from the hardware deployment to the applications. Context producers, e.g. sensors, update a centralized broker that stores all the information and context consumers request information from the centralized broker. The centralized broker may run in a cloud infrastructure, i.e., with as many computing resources as necessary. If short request processing times are needed to support real-time use cases, a centralized broker can also run on an edge node.

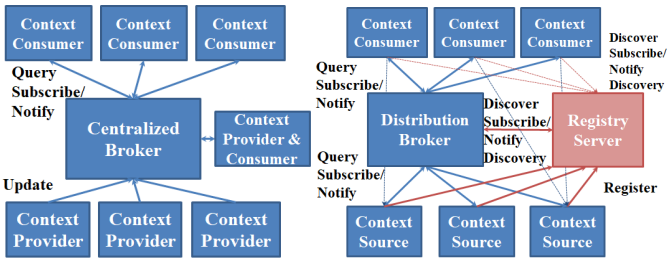

Fig. 4 Example of a centralized IoT architecture

Such a centralized broker can provide an NGSI-10 interface [17] or support a more expressive query language like SPARQL. Since all information is available locally, even complex requests can be efficiently supported. In FIWARE, the ORION Context Broker [18] is an example that can fulfil the role of a centralized broker.

If larger scale use cases are envisioned, e.g. a case where information from different existing sensor deployments are to be combined, a distributed IoT architecture is needed ideally without rebuilding the already running components. The right part of Fig. 4 shows an example of such an IoT architecture.

Here context sources can represent local IoT systems that already support a local use case, e.g., with real-time requirements, but the same information provided by local sensors may also be relevant in a larger scale use case and it would not be reasonable to deploy the same kind of sensors again to support the second use case. The distribution broker queries or subscribes to information from a set of distributed context sources. To do that, the distribution broker has to know which context source can provide which context information, and context sources should register the kind of information they can provide with a registry server. The assumption is that what kind of information is available, e.g., "I can provide the temperature of rooms in a geographic area" is much less dynamic than the actual information, i.e., the actual temperature of rooms which are only provided by the context source on request. If a context consumer requests the temperature of rooms from the distribution broker, the latter requests the set of context sources that have relevant context information from the registry server and then directly retrieves the speed information from these context sources and returns the aggregated results to the requesting context consumer.

Such a distributed IoT architecture can be realized using the NGSI-9 [19] and NGSI-10 interfaces. The registry server implements the NGSI-9 interface related to registering and discovering context sources, whereas the distribution broker and the context sources implement the NGSI-10 interface for requesting context information. In FIWARE, the Aeron IoT Broker [20] can serve as distribution broker, whereas the NEConfMan IoT Discovery [21] can serve as registry server.

Instead of integrating information from simple context sources, the information from whole domains represented by centralized or distributed brokers can be federated by a federation broker. In a smart city with multiple departments sharing their information to enable advanced applications., each department would run a broker for their own domain, serving local applications, but they also register (a subset of) the information they want to provide with the registry server on the federation level.

Overall, the different IoT architectures show an evolution path. It is possible to start with one or more small, centralized deployments. These can be combined in a distributed setup without having to completely rebuild the existing local setups. Finally, multiple domains, e.g. as they exist in smart cities, can be combined using a federation setup.

One important aspect for the distributed and federated approaches is that the broker can identify the context sources that have relevant context information for the respective request. In a large scenario, there could be hundreds or thousands of context sources with hundreds of entities each. It is important that the discovery of sources in the registry server is highly selective. Ideally the information in the request on the one hand and the information in the context source registrations on the other hand lead to only a limited number of matching context sources that have to be contacted.

The other important aspect is that the number of updates on the registry server does not get too high. In a large system with thousands of entities, changes can create a significant load. A possible approach that is supported in NGSI-9 is to allow different granularities of registrations, i.e. a context source can register each attribute for each specific entity (the current temperature of Room1), just the entity (Room1), just the entity type and attribute (temperature, room) or even just the type (room). Furthermore, it is possible to provide a geographic scope for which information can be provided, and this is especially relevant in the type-based case.

As can easily be seen, there is a trade-off between the granularity of registration and the selectivity of the match described above; the more detailed the registration, the more likely it is to get a highly selective match. If a number of context sources have registered that they provide information about rooms, they may all have to be contacted; restricting the area according to a geographic scope may nevertheless help. Thus, finding the best trade-off between granularity and selectivity is important when deciding on a deployment.

The support of query patterns as described in section V is also related to the information available in the registrations. The choice will be made by balancing costs and benefits e.g., query flexibility, efficiency, usability and impact. In a dataset stored in a centralized architecture where entity relationships and values barely change, all patterns could be supported; on the other hand, for one or more datasets organized in a distributed or federated architecture, and entity relationships or values that frequently change, query patterns on entity values and relationships could not be efficiently supported.

VII. Conclusion

An evolution framework is proposed to maximally reuse existing CIM platforms specifications and bring graph-based semantics to CIM platforms enabling interoperability and eventually cross domain federation. This framework is adopted by the ETSI Industry Specification Group on cross-cutting Context Information Management (ISG CIM), and the resulting specifications comprise context information model and context information management APIs, as shown in the ISG group specification CIM API [22].

As further works, analysis will be deepened to bridge the gaps between the proposed property-graph-based model and purely RDF derived semantic models on aspects such as entailment, graph rewriting, reference to upper ontologies and domain-specific ontologies, information indexing, and implementation on top of various NoSQL databases, moving from the current use of MongoDB to native graph databases to further improve utilization effectiveness and efficiency. The evolution framework will also involve strategies to enable richer query patterns on specific data types such as queries on geolocation and time based on existing standards.

Acknowledgments

This work has been supported by the European Union's Horizon 2020 research and innovation programme within the WISE-IoT project under grant agreement No. 723156, the SynchroniCity project under grant agreement No. 732240 and the FI-NEXT project under grant agreement No 732851-3.

References

[1] Bauer, M., Kovacs, E., Schülke, A., Ito, N., Criminisi, C., Goix, L. W., & Valla, M.. The context API in the OMA next generation service interface. In *(ICIN), 2010 14th International Conference on Intelligence in Next Generation Networks,* 2010.

[2] FIWARE-NGSIv2-Specification https://orioncontextbroker.docs.apiary.io/#

[3] ISG CIM Use Cases, http://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=51347

[4] F. Ramparany, R. Poortinga, M. Stikic, J. Schmalenstroer, and T. Prante, "An open context information management infrastructure the IST-amigo project," in *2007 3rd IET International Conference on Intelligent Environments*, 2007, pp. 398–403.

[5] A. K. Dey, "Understanding and Using Context," Personal Ubiquitous Computing, vol. 5, no. 1, pp. 4–7, Jan. 2001.

[6] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines," *Bulletin of the Association for Information Science and Technology*, vol. 36, no. 6, pp. 35–41, 2010.

[7] Angles, R. (2012, April). A comparison of current graph database models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on* (pp. 171-177). IEEE.

[8] Flury, T., Privat, G., & Ramparany, F. (2004). OWL-based location ontology for context-aware services. *Proceedings of the Artificial Intelligence in Mobile Systems (AIMS 2004),* 52-5

[9] W. Li, G. Privat, and F. L. Gall, "Towards a semantics extractor for interoperability of IoT platforms," in *2017 Global Internet of Things Summit (GIoTS)*, 2017, pp. 1–6.

[10] J. Frey, K. Müller, S. Hellmann, E. Rahm and M.-E. Vidal, "Evaluation of Metadata Representations in RDF stores", in Semantic Web Journal, 2017.

[11] OMA NGSI Specification, http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf

[12] GSMA IoT Big Data NGSIv2 profile, https://www.gsma.com/iot/wp-content/uploads/2016/11/CLP.24-v1.0.pdf

[13] RDF1.1 XML Syntax, https://www.w3.org/TR/rdf-syntax-grammar/

[14] JSON-LD W3C Recommandation, https://www.w3.org/TR/2014/REC-json-ld-20140116/

[15] W. Li and G. Privat, "Cross-Fertilizing Data through Web of Things APIs with JSON-LD," in Services and Applications over Linked APIs and Data, ESWC2016 workshop, 2016.

[16] schema.org, http://schema.org

[17] NGSI-10 Open RESTful API Specification, https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification

[18] Orion Context Broker, https://fiware-orion.readthedocs.io

[19] NGSI-9 Open RESTful API Specification, https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification

[20] Aeronbroker, https://github.com/Aeronbroker/Aeron

[21] NEConfMan, https://github.com/Aeronbroker/NEConfMan

[22] CIM API, http://www.etsi.org/deliver/etsi_gs/CIM/001_099/004/01.01.01_60/gs_CIM004v010101p.pdf, ETSI GS CIM 004