Capturing the Structure of Internet of Things Systems with Graph Databases

for Open Bidirectional Multiscale Data Mediation

Dana Popovici, Gilles Privat
Orange Labs
Grenoble, France
email:dana.popovici@gmail.com, gilles.privat@orange.com

***Abstract*—The deep structure of Internet of Things (IoT) environments understood as complex Cyber-Physical Systems (CPS) is made up of all interwoven relationships of physical actuation, sensing, proximity, grouping and containment between their constituent subsystems. We describe and assess three solutions for capturing and exposing this structure as a persistent graph of matching links between the informational proxies that represent these subsystems. A graph database may provide an access-efficient persistence support for this graph, tightly coupled with the mediation platform. Alternatively, the Resource Description Framework (RDF) may be used directly as a standard, open and extensible means to represent this graph and its associated semantics, with triplestores as a persistent repository supporting queries with standard languages such as SPARQL. A third solution would be, if fine-grain hyperlinked REST interfaces are provided to subsystems or their states viewed as resources, to delegate the creation and maintenance of this graph to an external web crawler and search engine.**

Keywords*—*Internet of Things(IoT); Cyber-Physical Systems (CPS); Sensor-Actuator Networks; Ontologies; Linked Open Data; Resource-Oriented Architectures(ROA); Resource Description Framework (RDF)

I. INTRODUCTION

The Internet of Things (IoT) may, in its extended acceptation [1], reach beyond connected devices to encompass all kinds of physical entities, be they legacy appliances, passive items or subsets of space. These entities get identified and represented together with the attached devices through Internet of Things platforms [2] and may as such be the target of varied applications operating on top of these platforms. The information maintained by these platforms may comprise both real-time information about the state of the identified entities and devices as well as structural semi-static information about the relationships between these. We focus here on the solutions for the representation and management of the graph made up of all these relationships, which may correspond to the following:

- device used as primary sensor for an entity
- device used as secondary sensor for an entity
- device used as actuator for an entity
- device acting upon an entity as a side effect
- entity containing another
- entity adjacent to another
- device connected to another through the network

This clearly goes much beyond networked device management, to get closer to the structural representation of Cyber-Physical Systems (CPS) decomposed as interacting relevant subsystems. Typical examples of these IoT environments modeled as Cyber-Physical Systems could be smart homes, smart buildings and smart cities. We are interested in these rather than in more traditional one-of-a-kind industrial CPS because they stand to gain the most from the use of shared platforms instead of dedicated and vertically integrated design. We can take advantage of the generic character of the categories of entities/subsystems that make up these systems and of their invariance from one of their instances to another. All buildings are thus made up of rooms, corridors, floors, appliances and furniture items pertaining to broadly similar categories, while cities comprise streets, crossings, blocks, parking places, etc. These categories may be drawn from shared domain-specific ontologies and the entities we target as nodes of our extended IoT graph will be instances of these classes, providing a semantic reference for their eventual identification.

We present an architecture template for an IoT platform in the smart home, building or city domains in section II. We explain how the graph of relationships between entities can be the core of this platform in section III. We assess comparatively the two database solutions we propose, graph data base and RDF triplestore, in sections IV and V respectively and explain how they get interfaced to the platform in section VI. Section VII presents the alternative to having just a RESTful interface to the platform provided as a complete set of hyperlinked resources and delegating the capture of the corresponding graph to an external web crawler.

II. ARCHITECTURE OF AN INTERNET OF THINGS PLATFORM

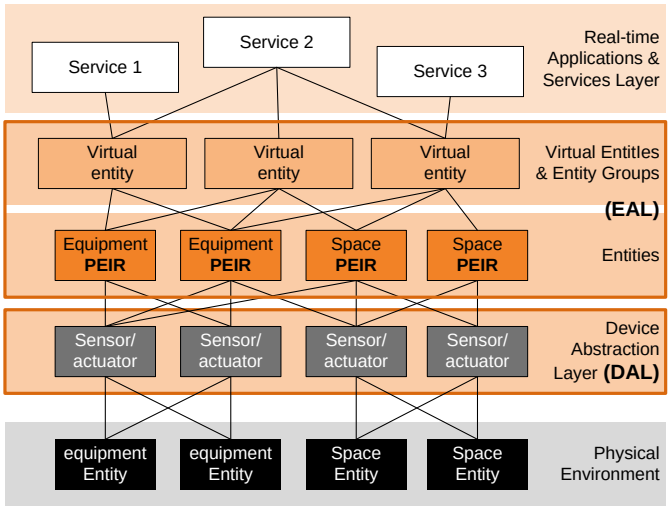

Figure 1. IoT platform architecture

An IoT platform mediates data between the target physical environment and applications, both upwards by fusing /aggregating/abstracting sensor data, and downwards by passing on commands to actuators[2][3]. The architecture of such an IoT platform uses, in our approach, software modules called “Physical Entity Informational Representatives” (PEIR) that serve as proxy for the individual entities, making up an Entity Abstraction Layer (EAL). This may comprise additional

 ISBN: 978-1-61208-408-4

levels of abstractions when these entities get regrouped as "virtual entities" according to functional or physical criteria (Figure 1). Beneath this, the Device Abstraction Layer (DAL) provides a uniform interface to networked devices (sensors & actuators) that serve as physical intermediaries to the entities, regardless of protocols and technologies. DAL and EAL each have their own REST interface for a full decoupling of the two.

III. CAPTURING THE DEVICES & SUSBSYSTEMS INSTANCE GRAPH

Our approach does partially disregard the heterogeneity and actual complication of individual entities (subsystems) by mapping them to simple or even simplistic generic models drawn from domain ontologies as proposed above. This makes it possible to focus on the more relevant complexity of the overall system, as it results from the composition of these individual subsystems and, crucially, the web of relationships of different kinds into which they are caught.

We will use throughout this paper the example of a smart building, renting office space to a number of companies as a prototype IoT environment. Each floor may comprise similar spaces and pieces of equipment, but there is a multitude of relationships to account for between these physical entities and other relevant virtual entities (e.g., sets of offices rented by the same company). Moreover, certain devices or entities can be used for purposes other than their primary function, generating contextual information through indirect relationships, for example a computer acting as presence sensor (if someone is typing, the office is occupied). Figure 2 shows a small subset of the different types of relationships captured by the graph representing our smart building example as an IoT system, showing parts of its three interconnected sub-graphs: the nodes of which are respectively ontology categories, physical entities and devices. The ontologies are imported from online repositories, but a local copy is needed on the IoT platform for efficient access. The external ontologies being referenced may be upper ontologies, domain ontologies (e.g., for smart home/building/city), and transversal device ontologies. The links between nodes are also of diverse types, representing either semantic relationships (e.g., instanceOf, subClassOf) or structural (e.g., isOn, actuated by) relationships. The resulting overall graph is thus heterogeneous and unites sub-graphs with different types of nodes, linked by different types of arcs.

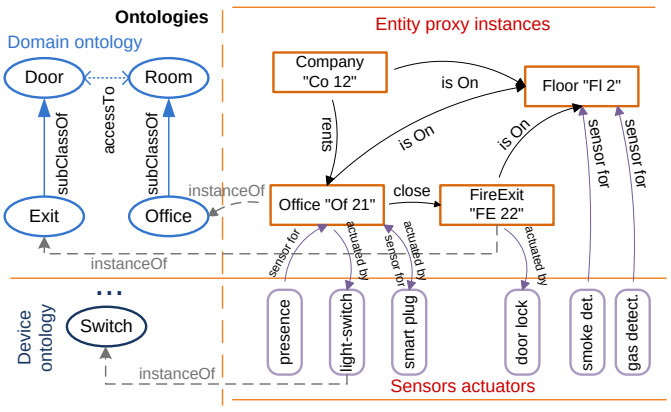

Figure 2. Example smart building graph

Contrary to the closed & fixed systems, which are the usual target of embedded systems design, these relationships and the systems configuration they capture will vary over time, as new entities will appear, move or be moved.

To give just a tiny example of the use of this graph by a very simple Smart Building IoT application, we could imagine a situation where an alert from a gas detector triggers turning off electrical equipment that belongs to a given company in the corresponding area. A query could be used to find the smart plugs that should be turned off for the company "Co 12". Expressed in a SPARQL-like language, it would be:

```
SELECT ?smart_plug WHERE
{?smart_plug location:adjacent ?gas_detect;
?gas_detect state:hasState Alert;
?smart_plug co:belongs "Co 12".}
```

IV. GRAPH DATABASE REPRESENTATION

The first solution that we investigate for capturing the structure of the IoT system through the relationships between physical entities that it represents is a graph database. This type of database focuses on the software objects that are the nodes of the graph and provides optimized algorithms for the traversal of the graph, with the benefit that nodes connected through directed paths are fast and easy to retrieve and unrelated nodes are not traversed. Property graphs provide rich information about both nodes and relationships, through key-value-pairs that describe them. A solution such as Neo4j, an open-source graph database implemented in Java, offers both an embedded and a server version. Neo4j queries are expressed in Cypher, an SQL-inspired, declarative language that describes patterns on the graph.

A. *Integrating database engine into the IoT platform*

A graph database seems well suited for the task of capturing the complex connections between the IoT sub-systems and entities. Based on the target platform there are several aspects to consider, among them the scale and latency requirements. Our approach includes two abstraction layers that are not necessarily hosted on the same machine, resulting in the distribution of nodes between these two platforms. The scale of the graph may vary widely, from smart homes with a few hundreds of nodes, up to smart buildings and smart cities. Even for the smaller scale of a smart home, it would still be possible to have two different platforms, typically a HomeLAN gateway and a dedicated home automation server. Tight coupling between the core of IoT platform (the Entity Abstraction Layer made up of all PEIRs) and the database (with nodes matching the PEIRs), could be achieved by hosting both on the same platform.

The database is built and updated from two sources, the IoT platform, during configuration and reconfiguration phases, and the domain ontologies stored in online repositories. Although graph databases can represent semantic information, this requires another step to import them from the OWL format with the OWL API.

B. *Advantages and disadvantages of a graph database*

The most important advantage of the graph database solution is, for our purposes, besides their query performance and scalability, their potential tight integration with the IoT platform, if both could share the same Java execution environment as proposed before. The disadvantage of using such a database is the added effort of its integration to the platform, both for representing structural relationships when they are discovered and for semantic relationships that need to be transformed and included from ontology repositories. Reasoning from the graph (e.g., inferences drawn from combinations of structural and semantic relationships) is not

 ISBN: 978-1-61208-408-4

"native" to graph databases such as Neo4j, requiring one more step of data transformation to RDF triples and the use of a reasoner such as Pellet.

V. RDF REPRESENTATION & TRIPLESTORE-BASED PERSISTENCE

Representing the relationships of entities in the IoT platform through RDF and storing the information in triplestores (the corresponding "native" solution) could be considered as the "opposite" solution of a graph database. RDF natively represents semantics and needs to be extended to include structural information whereas the graph database does the contrary. RDF is a W3C standard and represents relations (statements) as triples: subject, predicate, object. RDF databases are a prominent standards-based NoSQL solution. This representation is the basis of Linked Open Data, offering a means to publish structured and semantically annotated data that supports database-style queries. It is also one of the data models used for the representation of OWL-based ontologies. All these advantages can be used to our benefit, as our approach includes importing ontologies and sharing the generated data.

A. Exporting the database as RDF graph

Depending on the target system, the data generated and stored in the triplestore should be made available to the general public or to a subset of authorized stakeholders. Some of the information could be useful if exposed directly as Linked Open Data. For example, in a smart city, car parks with their location and number of available parking places could be shared. A smart building might have a public list of companies that are currently renting offices, as well as the number of available offices for rent. It is especially appropriate to share data for the smart city, but even some of the data from the smart home might be of interest to the general public. For example, temperature information or luminosity, as measured by outside weather stations can be shared and used to compute mean values for a city or area. It should be noted though that in the smart home and smart building domains the access to the RDF graph should be tightly restricted, as most of the information it contains is strictly private.

B. Query languages, rule specification & reasoning tools

Representing the system graph as RDF triples can profit from the existing tools to better exploit the information. Powerful open source triplestores exist today, such as Jena, Sesame, Virtuoso, Bigdata and many others. Most of them support SPARQL, a standardized and interoperable query language that is fairly simple to use. Some of the frameworks provide more precise functions, including reasoning tools for the stored data, and may even be built for that purpose, like Ontotext GraphDB, an OWL-based triplestore.

In this paper, we use the example of Jena, one of the most popular triplestores. Although its native persistence implementations are not the best ones from a scaling point of view, Jena can be used to interface several other triplestores. It also provides good inference support through several reasoners and their rules. It includes a generic rule reasoner, as well as an RDFS, OWL and a transitive reasoner for different levels of inference. Other advantages include the SPARQL server that is provided out of the box (Fuseki for Jena), allowing distant access to the graph.

These tools are used to enrich the knowledge stored in the graph, and help support smart environment-specific requirements. In a smart environment, the applications target not only devices and physical entities, but also entity groups and virtual entities. Some of the groups are static or semi-static, requiring them to be represented in the graph at all times (e.g. the group of offices being rented by a company). Other groups might be created in an ad-hoc fashion when they are needed, by querying the RDF database. This, plus the reasoning tools available, opens the IoT platform to unlimited possibilities.

C. Advantages and disadvantages of RDF databases

When compared to other NoSQL databases, RDF triplestores have some important advantages. Data is represented by a simple, uniform, standard model, which allows for portability and interoperability. This also means that the inner graph representation is vendor-independent, and it can easily be imported to another database. Having a high-level declarative query language is another major advantage.

Compared to graph databases, there are several differences. RDF can be considered as a graph, but is relation-centric (as opposed to node-centric) and is composed solely of labeled arcs. Representing undirected edges would require coupling two arcs in opposite directions between the corresponding nodes. Another issue concerns literal properties that are objects in the RDF representation, causing the resulting graph to have "leaf nodes".

Alternatively to "native" RDF triplestores, the persistent storage of RDF graphs may be provided through multiple methods, including tuple stores, graph databases and even traditional SQL-based databases. Graph databases such as Neo4j store data directly as a graph and thus benefit from optimized traversal algorithms. In general, native RDF databases offer slower performance than other solutions (especially graph databases) and they scale badly, making them a solution that is not perfectly adapted to applications with harder requirements.

VI. INTEGRATING THE DATABASES WITH THE IOT PLATFORM

We have presented two solutions for the representation of the relationships between entities of an IoT platform and compared their relative performance. In this section, we wish to address some of the issues that are shared between them and can be discussed jointly. Implementations of both RDF and graph databases can be either tightly or loosely coupled with the IoT platform. We use as example two representing databases, Jena for triplestores and Neo4j for graph database. Both offer Java APIs that can be used through direct invocations and that imply the database should be on the same machine as the rest of the IoT platform. This raises several questions about possible implementation solutions. As explained in the previous sections, we consider at least two separate abstraction layers, which might be on different machines, one for the connected devices and another for the entity representation. It seems more appropriate in this case to host the database on the same machine as the entity abstraction layer. In this way, the database can still store information coming from the device abstraction layer, but it will be more tightly coupled with the entities. Given that IoT applications target the entity level, the database enriches the quality of the applications through semantic and structural information. This solution might work very well in a smart home IoT system, but less so in a smart city, where the abstraction layers are physically distributed.

The second option, once again available for both Jena and Neo4j, is to use servers and a RESTful interfaces over HTTP.

 ISBN: 978-1-61208-408-4

Jena has the Fuseki server that can be used for SPARQL queries and Neo4j is first and foremost a server. This option prompts another choice about the location of the server with regard to the abstraction layers: should we use a regular cloud-based solution, or use "edge of cloud" ("fog") computing [4] which extends cloud computing to embedded platforms on the outer edges of the network. Fog computing combines the benefits of centralized data processing and proximity to the end-user, enabling flexible and virtualizable platform support for real-time applications of the IoT. Regarding the application domains that we consider, it seems that fog/edge of cloud solutions could provide a good tradeoff while being equally adapted for different types of systems: smart home, smart building or smart city.

VII. EXPORTING THE GRAPH FROM HYPERLINKED ROA INTERFACES

Both of the previously presented solutions imply an effort to create and maintain the structural relationship graph for the IoT system within the platform and keep it consistent with the internal operational representation at all times, thus placing a heavy burden on the software design of the platform. A third option is more in line with a full "web of things" [5] approach. Assuming we provide a full REST [6] interface to the platform, which means each entity of the Entity Abstraction Layer, each state of these entities and each device of the Device Abstraction Layer would be a resource in the REST sense, each with its own URI and exposed hyperlinks to other resources it interacts with, we could dispense with creating and maintaining a database of the corresponding graph of hyperlinks inside the IoT platform and delegate it to another platform or third party general-purpose web tools. This is what search engines have done for the original web of documents: mapping it as a graph and exploiting the structural properties of this graph for information retrieval. This would also fit better with a preference for of a minimal and loosely-coupled IoT platform.

In a "pure" ROA interface, (corresponding to the third level of the Richardson Maturity Model, "HATEOAS" [6]), no global functional description is required, all resources are self-descriptive and provide their own URI that can be interpreted by applications without requiring any "out-of-band" information or prior knowledge, with their behavior, semantic mappings, associated resources and sub-resources accessible through hyperlinks. In our architecture, these hyperlinks correspond to the relationships between entity groups, entities and devices as maintained by the graph database proposed in the previous two approaches, with additional links corresponding to the allowed state transitions between states viewed as sub-resources. Non-functional and semantic information is attached to each entity as read-only resources, accessible together with entity states through HTTP GET, while the controllable states can be updated through corresponding HTTP PUT. In this approach, the graph representation of the system does exist, but only implicitly through these hyperlinks, just as is the case in the web viewed as a graph. All the information that may be required from applications is in principle available by traversing the graph made up of these hyperlinks, providing the equivalent of interface introspection, discovery and dynamic service composition from more traditional SOA approaches. Just as the public web itself, this minimal web of things platform requires external or third party tools to provide the equivalent of the functionalities that are natively provided by the databases provided in the previous two approaches, especially responding to database-like queries. The graph structure could thus be extracted by using a crawler tool similar to web crawlers that would systematically and exhaustively traverse the graph of REST hyperlinks to recreate a fully indexed, searchable data structure, possibly using a graph database of its own. This database could in turn respond to queries such as the one mentioned in section III (possibly with restrictions on the primitives involved), providing direct links to the target entities for applications to monitor or control through the corresponding resources addressed individually through their REST-compliant URIs, thus making it possible for these applications to bypass the database when direct control will be involved.

VIII. CONCLUSION

Most present-day Internet of Things applications are limited to monitoring and data collection. If they involve control, it is usually not part of their automatic operation and occurs through human operators. The architecture we propose here is clearly designed to go beyond these in order to support bidirectional data mediation for applications that involve direct real-time automatic control of the same entities being monitored. With this in view, the proposed graph representation is essential in determining the perimeter of entities that may be impacted by a given control action, to avoid undesirable or potentially cascading and catastrophic side effects of any action, which could be done by just tracing directed paths of relevant actuation relationships through this graph. More fundamentally, a platform based on such a representation is intended to be a Cyber Physical System platform, not only an IoT platform, and the proposed graph representation should be understood as representing this CPS as a complex system, in keeping with received graph-based modeling approaches for such systems where node (subsystem) models may be grossly simplified as long as the complexity and structural accuracy of their interrelationships and of their physical grounding is fully accounted for.

ACKNOWLEDGMENTS

We are most grateful to Mengxuan Zhao and Laurent Lemke, who have contributed to shape and evolve the infrastructure on which this work is based, to Didier Donsez and Florence Maraninchi, for many in depth discussions around the issues presented here, and to Wenbin Li who has taken on the challenge to follow up on this work.

REFERENCES

[1] G. Privat, "Extending the Internet of Things", Communications & Strategies, Digiworld Economic Journal , vol 87, 2012, pp101-119

[2] G. Privat, M. Zhao, and L. Lemke, "Towards a Shared Software Infrastructure for Smart Homes, Smart Buildings and Smart Cities", International Workshop on Emerging Trends in the Engineering of Cyber-Physical Systems, Berlin, April 14, 2014

[3] Z. Hu, G. Privat, S. Frenot and B. Tourancheau, "Self-configuration of Home Abstraction Layer via Sensor-Actuator Network", Ambient Intelligence. Springer Berlin Heidelberg, 2011, . pp146-150.

[4] F Bonomi, R Milito, J Zhu and S Addepalli, "Fog computing and its role in the internet of things", Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, 2012. p. 13-16

[5] D. Guinard, V. Trifa and E. Wilde, "A resource-oriented architecture for the web of things", Internet of Things (IOT),. IEEE, 2010. p. 1-8. .

[6] L. Richardson and S. Ruby, "RESTful web services" O'Reilly Media, 2008.

 ISBN: 978-1-61208-408-4