The BIG IoT API - Semantically Enabling IoT Interoperability

Arne Bröring
Siemens AG

Andreas Ziller
Siemens AG

Victor Charpenay
Siemens AG

Stefan Schmid
Robert Bosch GmbH

Aparna S. Thuluva
Siemens AG

Darko Anicic
Siemens AG

Achille Zappa
National University of Ireland

Mari Paz Linares
Univesitat Politecnica de Catalunya

Lars Mikkelsen
Aalborg University

Christian Seidel
VMZ Berlin GmbH

Today, IoT platforms offer proprietary interfaces and protocols. To enable interoperable interaction with those platforms we present the generic BIG IoT API that employs a novel approach for self-description and semantic annotation to fully adapt arbitrary IoT platforms. We have deployed this approach for multiple platforms from the mobility domain.

Today, the Internet of Things (IoT) is an increasing commercial reality and connected 'things' are already largely outnumbering Web users. Hence, the IoT plays a tremendous role in digitalization efforts. Possible applications and market areas range from smart cities to autonomous manufacturing. In order to provide access to data (and functionalities) of connected things, dozens of IoT platforms have emerged, such as ThingWorx, Siemens' MindSphere, or Bosch's IoT Suite.

However, there is still a crucial issue in the IoT: the missing interoperability between platform protocols and interfaces. Today, IoT platforms are vertically oriented and often still closed systems. A heterogeneous landscape of standards is used by some IoT platforms and others are solely relying on proprietary interfaces. This fragmentation of the IoT and the lack of interoperability prevent the emergence of IoT ecosystems that could bring substantial economic value. A recent McKinsey study [1] estimates that 40% of the economic potential of the IoT directly depends on interoperability.

BIG IoT [2] aims at addressing these interoperability issues. Naturally, it is impossible to dictate one language (i.e., protocol and interface) for the IoT, since purpose and domain of IoT platforms differ. Instead of defining a single gigantic IoT API that covers all possible IoT platform functionalities, this work reaches interoperability through the *BIG IoT API* that defines a generic

base structure, which can be instantiated by annotation with terms from a semantic vocabulary. The advantage is that the semantic vocabulary is easier to be developed in a community process, which is essential for building a multi-platform IoT ecosystem. In this way the API is kept slim and easy understandable while being generic and applicable in different domains.

RELATED WORK

Previous work on IoT architectures (e.g., [3]) provide high-level layering and abstraction concepts. We go further than a mere architectural proposal here and provide a concrete API and its implementation. This is similar to [4], although, we do not develop yet another platform, but instead propose a *generic* API to be applied to existing platforms. The key difference to the manifold IoT standards and adaptation approaches already out there is that the BIG IoT API provides only a base structure and relies on semantic annotation to make it useful. Thereby, managing interoperability is transferred from the API to the semantic model.

An example for the contrary approach, i.e., defining one *specific* API to integrate all kinds of IoT functionalities, is the meSchup IoT platform [5]. Via a JavaScript API, a naming scheme allows accessing properties from a list of integrated device types. I.e., for each new device type the meschUp IoT platform is adjusted. This works well for a single IoT platform with full control over the API. As soon as multiple platforms are involved to form an ecosystem, a community process needs to be established, which can be better supported through extending a common semantic model than a full API definition. Established semantic models, which have already proven that they can be efficiently managed by a community, are schema.org [6] or Haystack[1].

There is a heterogeneous landscape of existing standards for IoT platforms, such as OMA LWM2M[2], OneM2M [7], OGC SWE [8] or OPC UA[3]. Our approach does not aim to supersede those standards. Instead, by defining an according extension of our semantic model, components that follow these standards can be integrated with our ecosystem by implementing the BIG IoT API and describing the data offered semantically. The W3C standard Hydra[4] also allows documenting a REST API. Operations on resources can be specified and allow a client to understand how to interact with the API. This is similar to BIG IoT offering descriptions as both describe the interface using JSON-LD. However, our approach goes beyond the interface description by providing semantic domain models that can be used to annotate it. Also, similar to this is the HyperCat standard[5] that describes available IoT resources and semantically annotates them at a centralized hub. Our approach is different as it does not solely focus on discovery and further avoids a single point of failure through a hub, but keeps the autonomy at the IoT platforms that are directly accessed via the BIG IoT API.

In [9], the Semantic Gateway Service (SGS) is presented as a bridge between devices and applications and translates between device messages and higher level data models. To enable analysis and reasoning, the data received from the device is semantically annotated. As a central model W3C's SSN ontology [10] is proposed. The SGS approach is similar to the BIG IoT approach; however, functionalities such as discovery and accounting are not covered. While we define a semantic domain model, SGS leaves this open, which ultimately leads to less interoperability.

The challenge of semantic interoperability on the IoT is addressed in [11] by introducing the concept of smart spaces. A Semantic Information Broker (SIB) captures the context of a local smart space (e.g., a room or vehicle) and is used for discovery and data storage. This is similar to our Marketplace; however, goes beyond the SIB by supporting accounting of data access and aims for monetization of IoT resources. On the other hand, data storage is intentionally not supported, but kept at the responsibility of each platform. This is important for many IoT platform providers, as their data is protected and needs to remain behind their own security mechanisms. The SIB intentionally captures the data of all agents within the smart space to decouple the agents from each other.

[1] https://project-haystack.org
[2] http://www.openmobilealliance.org/release/LightweightM2M/V1_0_2-20180209-A
[3] https://opcfoundation.org/developer-tools/specifications-unified-architecture
[4] http://www.hydra-cg.com/spec/latest/core/
[5] http://www.hypercat.io/standard.html

Addressing semantic interoperability, Kotis & Katasonov [12] propose a semantic gateway design that supports the semi-automated translation of things' data. Therefore, alignments of the things' ontological descriptions with a well-defined schema are computed. This allows the provision of data in a uniform way. Our approach requires the provider to register an offering description at the Marketplace and therefore translate the metadata beforehand manually. An automated alignment could be included in the future. However, it requires a preexisting ontological description of things, which is today rarely present.

A SEMANTICALLY-ENABLED API FOR INTEROPERABLE IOT ECOSYSTEMS

Figure 1 provides an overview of the interplay of components of an IoT ecosystem based on the BIG IoT API: The central pillar of the ecosystem is the Marketplace. Here, a *Provider* (e.g., an IoT platform offering parking space data) registers its resources as so-called *Offerings*. To facilitate a provider in conforming with the BIG IoT API, the Provider Library (Lib) can be utilized.

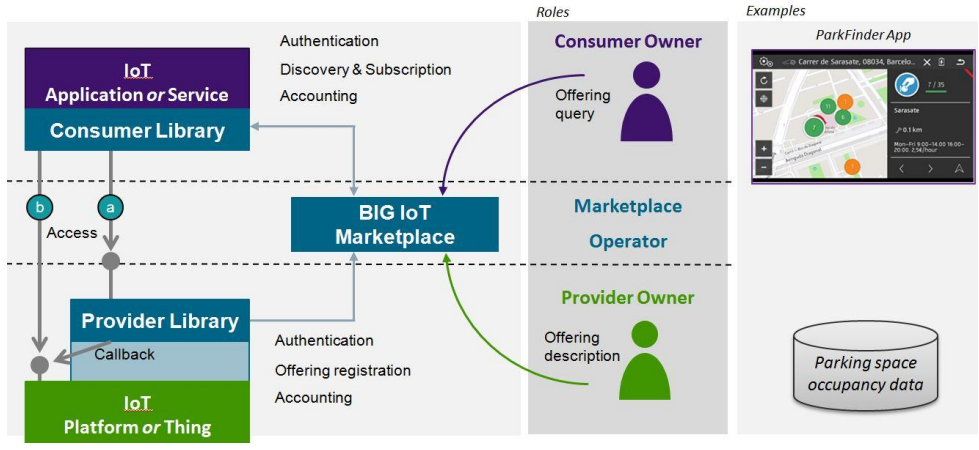

Figure 1: IoT ecosystem components

A *Consumer* in this ecosystem is an IoT application or service that consumes offerings of one or multiple IoT platforms. A consumer discovers matching offerings at runtime based on a so-called *Offering Query*. Both sides, the consumer and provider, report accounting data (e.g. number of resource records obtained/provided) back to the Marketplace, which enables the monetizing of resources.

A consumer example is the *ParkFinder* application that we developed for the SEAT S.A. car manufacturer. Since cars are roaming between cities, the app needs to be capable of accessing parking data from multiple platforms in an interoperable way, i.e., without a per-platform integration. We implemented ParkFinder as a smartphone app that is incorporated with the car's dashboard. The user can get real time parking availability by filtering on preferences (e.g., distance or price). Once the user simply enters where she is headed, the app provides real time guidance to the parking spot. Further description on the implementation of this app with the BIG IoT API is given below.

Offering Description Model

An *Offering* represents a resource provided by an IoT platform. This could be data, such as a timeseries of temperature data, or functions, such as changing the sampling rate of a thermometer. An offering is defined by an Offering Description (OD) that can be registered on the Marketplace. The OD comprises information on how to access the described resources, syntactic definition of used input and output types, as well as semantic annotations to enable discovery and correct matchmaking between consumers' demands and providers' offers. The OD model is

shown in Figure 2 and based on the *Thing Description* model[6] developed in the W3C Web of Things Working Group.

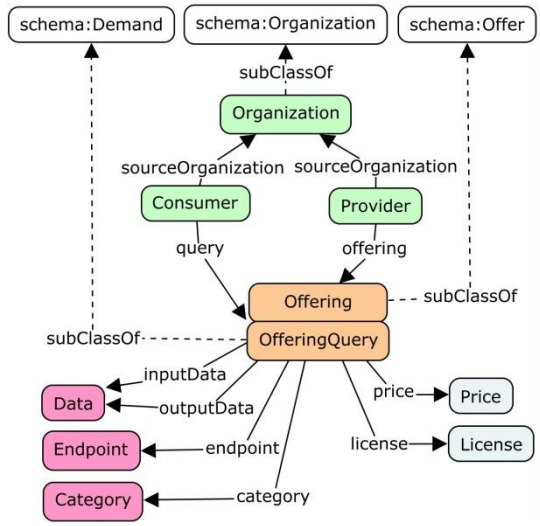

Figure 2: Overview of the offering model

At the heart of the model are the *Offering* and its conceptual counterpart the *OfferingQuery*. Both classes have the same properties. While an offering is linked to a provider, an offering query is created by a consumer to search for matching offerings via the Marketplace. This design follows the mechanism of demand/offer, modeled in schema.org [6] via the classes "schema:Demand" and "schema:Offer".

An offering has a *Category* that puts it in context and can be used for discovery as a search keyword. Further, the offering has endpoint details that specify the address, protocol and method (e.g., HTTP POST or CoAP GET). Crucial for enabling the access to offerings is the definition of its input and output data. Input data describe the parameters to query the offering. Output data describe the types used in the response. Additionally, the offering defines non-functional properties such as spatial extent, price or license, to refine the offering matching process.

Listing 1 shows an example of an OD as an instance of that model encoded in JSON-LD. This offering provides outside temperature measurements and hence declares to be of category *AirTemperature* from our environment vocabulary. Three inputs are defined: latitude, longitude, and radius. The outputData is defined as a complex structure that contains multiple objects, each one having two members: "temperature-value" and "unitOfMeasure".

```
{"@context" :          "https://schema.big-iot.org/ctx.jsonld",
  "schema" :           "http://schema.org",
  "providerId" :       "WeatherGuru",
  "name" :             "My-Temperature-Offering",
  "category" :         "http://schema.big-iot.org/environment#AirTemperature",
  "endpoints" : [ {
    "schema:url" :     "http://my-server/access/temp",
    "schema:type" :    "HTTP", "method" : "POST",
    "mediaType" :      "application/json"} ],
  "inputData" : [
    { "name" :         "latitude",       "rdfAnnotation" : "schema:latitude"},
    { "name" :         "longitude",      "rdfAnnotation" : "schema:longitude" },
    { "name" :         "radius",         "rdfAnnotation" : "schema:geoRadius" } ],
  "outputData" : {
    "rdfReference":    "schema:QuantitativeValue",
    "members": [
      { "name":        "temperature-value",   "rdfAnnotation": "schema:value" },
```

[6] https://w3c.github.io/wot-thing-description/

```
        { "name":       "unitOfMeasure",  "rdfAnnotation": "schema:unitCode" } ]
…
```

Listing 1: Example of an Offering Description

Vocabulary for the Semantic Annotation

For the semantic annotation of offering descriptions, e.g., defining the semantics of inputs, outputs and offering category, a well-defined vocabulary of domain terms is needed. This vocabulary should be widely shared and agreed upon so that all consumers and providers of IoT platforms can rely on it. Further, it should evolve in an open community process to allow active engagement by ecosystem stakeholders.

We have selected schema.org [6] as a basis for our domain model, as it provides a vendor-neutral, community-developed vocabulary for structured data. Schema.org markup is widely used on Web pages to annotate their content. Today, the core of schema.org covers a broad selection of terms ranging from e-commerce to sports activities. Other domains are covered as extensions (e.g., health-lifesci.schema.org). For the deployments in the mobility domain addressed in this work, we have developed an extension called schema.big-iot.org/mobility. It includes definitions for relevant concepts, such as parking site or bus stop. We have documented all defined concepts as Web pages in the schema.org style (see e.g. http://schema.big-iot.org/mobility/ParkingSpace).

Crucial for our generic API approach is to enable the management of the underlying semantic model in a community process. We support this via the Marketplace user portal. When registering a new offering, the provider can specify the semantic annotations. For example, an offering representing traffic speed data shall be registered. The user would find the category “Mobility” and its sub-category “Traffic”, but “Traffic Speed” is not part of the vocabulary. Hence, the provider can manually *propose* it as a new term by simply submitting it via the portal. Henceforth, this term is added to the vocabulary as “*proposed:trafficspeed*” and the offering is listed by the Marketplace by marking its category as proposed.

Once a new term is proposed to the system, a community process is initiated and moderated by an ontology engineer to include the term officially into the vocabulary or respond with a rejection of the addition. As the Marketplace is the focal point for providers and consumers to engage in the ecosystem, also other management tasks on the vocabulary, e.g., the proposal of deletion of terms or the re-organization of hierarchies, will be proposed and moderated here in the future.

The Generic IoT API

There are two kinds of integration modes supported by the BIG IoT API: (a) provider-side integration and (b) consumer-side integration. Both integration modes are illustrated in Figure 1.

In integration mode (a), an IoT platform uses the Provider Lib to proxy a data access call to an offering of the IoT platform. The application logic for that callback is implemented by the provider. To enable consumers to find the offering, the provider registers its OD at the Marketplace. To discover offerings, the consumer sends a search query to the Marketplace. A discovery may return multiple matching ODs. Once a consumer has selected an offering, it subscribes to it and obtains an access token via the Marketplace. Upon access to an offering, the Provider Lib processes the request, validates the access token, and calls the offering’s callback function. I.e., here the *access* interface is offered by the BIG IoT Provider Lib directly.

The access interface comprises the following parameters: First, the *token* field authorizes the user and is located in the header of the request. Second, the *input data* and *output data* fields are filled according to the specification in the OD. The *input data* included in the request are encoded according to the mediaType specified in the OD. E.g., in Listing 1, parameters are encoded as JSON and sent to the HTTP endpoint as a payload of the POST method. The JSON message in the payload could e.g. be *{"latitude":50.22,"longitude":8.11,"radius":500.0}*. Similarly, the *output data* are encoded according to the specified mediaType. In Listing 1, the response from

the provider is encoded in JSON and contains an array comprising objects such as *{"temperature-value": 23, "unitOfMeasure": "degC" }*.

In integration mode (b), the BIG IoT API supports a fully descriptive adaptation of the existing platform interface to provide direct access to offerings. As shown in Figure 1, the Consumer Lib accesses the legacy API of the IoT platform directly. This integration mode requires more information in the OD on the exact *syntax* of the request and its response, i.e., how it is structured and formatted. The required additional information is provided via a *template approach* for the request and a *selector approach* for the response.

The example in Listing 2 shows a request *template* to access an existing IoT platform to request temperature values with a location filter. Here, the request is encoded in XML, although, this template approach is language-agnostic. Instead of containing actual values, the request template contains placeholders (marked between "@@" signs). This is essential for our approach, as the consumer can now use this request template for querying data from the platform by replacing the placeholders with actual values that fit the specification of the inputData in the OD. A statement is included for each inputData element of the OD to link it with a placeholder; e.g., the statement "placeholder": "@@y@@" is added to the "latitude" input in Listing 1.

```
<?xml version="1.0"?>
<WeatherRequest>
  <observedProperty>schema:temperature</observedProperty>
  <location>
    <circle-y>@@y@@</circle-y>
    <circle-x>@@x@@</circle-x>
    <circle-radius>@@radius@@</circle-radius>
  </location>
</WeatherRequest>
```

Listing 2: Request template to access data from an IoT platform

To convey the semantics of the original response to the consumer, the OD needs to be also extended by the information of which data elements shall be mapped to which outputs. Therefore, the OD is extended by including *selectors*, which map response elements to outputData elements. E.g., an XML SOAP response "*<Envelope><Body><Measurements><Measure time="13:19">23</Measure>…"* contains an array of temperature measurements. These values can be linked to the output "*temperature-value*" of the OD in Listing 1 by adding the statement *"selector": "Envelope.Body.Measurements.Measure"* into the outputData element. Such a selector represents a path in the tree data structure (similar to XPATH). This goes beyond a similar integration mechanism described in [13], as also arrays of data values can be mapped.

DEPLOYING PLATFORMS & APPLICATIONS WITH THE BIG IOT API

The BIG IoT API has been utilized so far by 10 IoT platforms to provide over 60 offerings, which are publicly accessible on the Marketplace (https://market.big-iot.org/). In our deployments, we have focused so far on four geographic regions: Barcelona (Spain), Piedmont (Italy), and Berlin / Wolfsburg (Germany).

In Barcelona, WorldSensing's IoT platforms *FastPrk* and *Bitcarrier* have been incorporated in the ecosystem using the BIG IoT API to provide parking data and traffic information. Also, the *Sentilo* and *OpenIoT* platforms have been integrated to provide information on bike sharing, noise levels, and e-car charging stations. To detect people density in the city, Wifi probe sensors (stationary and mounted on busses) have been integrated. Further, we have integrated air quality sensors hosted on cars. In Piedmont, the regional *smartdatanet* platform has been integrated to provide data on parking, air quality, traffic and bike sharing. In Berlin / Wolfsburg, we integrated VMZ's *Traffic Information Center* and Bosch's *IoT Suite* to provide parking, live bus data and charging station availability.

Intentionally, we are providing similar data sets for different regions and platforms. Using the BIG IoT API, one application can demonstrate cross-platform interoperability[2][2][2][2][2]. We illustrate this aspect in context of the *ParkFinder* application (Figure 1) implemented for SEAT S.A. The key benefit of using the BIG IoT API for this app is that data from different platforms can be automatically discovered through the Marketplace. I.e., the system is efficiently extendable to different cities and parking data providers. We demonstrate this ability by integrating the app with the FastPrk platform and the VMZ platform. Their ODs both refer to the category "mobility:ParkingSpace", hence the platform offerings are found via the Marketplace.

Listing 3 shows the implementation of the provider for the FastPrk platform. The Java code authenticates a *provider* object, creates an offering description and sends it to the Marketplace. The Lib allows easy appending of metadata to the offering (category, region, inputs, outputs, etc.). A request handler (callback) is defined that implements the business logic to respond with the correct information when the offering is accessed. Similar to the provider code, the Consumer Lib is used for the ParkFinder application. A *consumer* object can discover and subscribe to offerings and access it with concrete input parameters.

```
Provider provider = new Provider(PROVIDER_ID, MARKET_URI, IP, PORT);
provider.authenticate(PROVIDER_SECURITY_TOKEN);

RegistrableOfferingDescription offeringDescription =
 provider.createOfferingDescription("MyOfferingID")
 .withCategory("mobility:ParkingSpace")
 .inRegion(Region.city("Barcelona"))
 .withLicenseType(LicenseType.OPEN_DATA_LICENSE)
 .addInputData("long",     "schema:longitude", NUMBER)
 .addInputData("lat",      "schema:latitude", NUMBER)
 .addInputData("radius",   "schema:geoRadius", NUMBER)
 .addOutputData("parkX",   "schema:longitude", NUMBER)
 .addOutputData("parkY",   "schema:latitude", NUMBER)
 .addOutputData("status", "mobility:parkingSpaceStatus", STRING)
  .withAccessRequestHandler(accessCallback);

offeringDescription.register();
```

Listing 3: Excerpt of provider code using Provider Lib

EVALUATING THE BIG IOT API

In this section, we evaluate the API concerning performance as well as usability. An analysis of these results is provided in the next section. Both aspects are important to introduce the API as an enabler for IoT ecosystems.

For evaluating the performance, we used the OpenWeatherMap[7] platform as an example and implemented a Java application that compares the access between two cases: (i) utilizing the Consumer Lib to access a Provider that proxies the platform in a BIG IoT-conform way and (ii) native communication with the platform API. We limited the evaluation to one offering, since it is the impact of using the BIG IoT API that is of interest, and not the difference between varied sizes of offering data.

The overhead caused by BIG IoT is measured as the difference in *transfer time* and *transferred data*. The transfer time is measured for a full data access cycle (i.e., start of the request call until response is received by application). The transferred data is measured as the total amount of bytes of the involved HTTP requests and responses. Thereby, case (i) involves additional HTTP communication between consumer and provider, while both were located on the same machine so that the measurements were not subject to public network cross traffic. The interaction with the Marketplace for registering and discovering the offering was not captured, as this only happens during application initialization.

[7] https://openweathermap.org/api

We ran this performance test 50 times, which resulted in an averaged transfer time of 126.6 ms for case (i) and 108 ms for case (ii). This overhead of 18.6 ms of the BIG IoT approach is due to the additional provider logic (e.g., validation of access token, and mapping of received data to semantically-defined outputs). Further, the BIG IoT approach caused an overhead of 240 bytes in transferred data. This is mainly due to the access token (a JSON Web token) that is sent by the consumer to the provider for authentication purposes.

For evaluating the usability, we conducted an online survey that was filled by 12 developers, external to the BIG IoT project, as part of a hackathon and an open call for ecosystem extensions. In the first question, most participants (84 %) rated their experience as developers from medium to high (3-4) on a scale from 0-5. Next, the survey consisted of 4 questions each for the usability of the Consumer Lib and the Provider Lib. They related to how much time it took to consume / provide the first and the last offering, how convenient the Libs were perceived, and how easy it was to get started.

To consume the first offering it took 67% of the participants 30-60 minutes, while the last offering could be consumed by 75% in less than 10 minutes. The provisioning of a first offering took 30-60 minutes for 50% and the last offering could be registered in under 30 minutes by 86% of participants. The convenience of the Consumer Lib was rated by 86% of participants medium to high, and for the Provider Lib 63% specified a high convenience. The ease of getting started was rated by most developers (67%) as high for the Consumer Lib and medium to high by 88% for the Provider Lib.

After these multiple choice questions, free-text fields allowed the participants to provide feedback. Positive aspects noted related to the good developer guide, the availability of complementary data on the Marketplace, and the semantic alignment with schema.org as it gives clear documentation. Suggestions for improvement were given regarding the support of more programming languages, the advancement of the Marketplace (e.g., regarding stability or better support for monetization), and guidance to choose the right semantic terms.

LESSONS LEARNED

With multiple real-world deployments of the API, we have shown how our approach facilitates interoperability. Deployments of IoT platforms from different domains and geographic regions can be transparently accessed by an application that utilizes the API. The case of the *ParkFinder* application shows the advantages of the presented approach and how this bridging of the interoperability gap has the potential of creating economic value. The ParkFinder app only needed to be implemented once, and can henceforth work with different IoT platforms providing parking information. This fulfils the "*Cross Platform Access*" pattern described in [2].

The API is generic insofar that no assumption is made about the semantics of inputs and outputs of IoT offerings. Our *generic* approach allows for annotations from separate semantic models and stands in contrast of defining one gigantic *specific* API that covers all possible functionalities of all platforms. Conceptually this is a delegation of the interoperability problem from the API to the semantic models. The key difference however is that *semantic models are easier to extend* and maintain in a community process. Project Haystack[8] and schema.org [6], which is used for millions of Web pages, show how such models can be efficiently managed by a community. For establishing a multi-platform IoT ecosystem, extending and maintaining a semantic model in a community process is easier than doing it for an ever-growing API. As only fractions of the semantic model are used by one platform, the impact is reduced – while in case of a specific API, it is expected that a platform implements most parts. A semantic model is intrinsically extensible (e.g. by adding terms to a taxonomy), thus it is capable to cover evolution of existing and addition of future use cases and domains.

As schema.org does not yet sufficiently cover IoT-related terms and also specific terms for our application domains, we have extended the schema.org vocabulary. We aim to integrate these

[8] https://project-haystack.org

extensions to the schema.org core or to new domain specific extensions of schema.org so that it will cover in future also IoT-relevant application domains such as mobility, environment, or health. The model of the offering description has been aligned with the Thing Description model developed in the W3C Web of Things group, which is on its way to become an official recommendation.

In order to bring an existing IoT platform into the ecosystem, we present two alternative modes: (a) provider-side and (b) consumer-side integration. While the provider-side integration relies on the easy-to-use libraries, the consumer-side integration incorporates an interface and protocol description of the platform API, which enables a fully descriptive integration.

This fully descriptive adaptation of platform APIs can facilitate integration of new IoT platforms into the ecosystem. Particularly when thinking this concept further towards tools that support the creation of the ODs through intelligently designed user interfaces, enriched with protocol templates and data selectors. Once such an OD is defined for a specific communication protocol (e.g., LWM2M or OneM2M) it can be even shared with other platform providers that use the same protocol.

Our deployments have shown, that the fully descriptive approach can be applied to Web APIs with different protocols, e.g., HTTP, CoAP, MQTT, or WebSockets. The outgoing messages may use arbitrary structures; however, they have to be ASCII-encoded to represent them as a template. The incoming messages transmitting data from the platform to the consumer has to represent a tree structure so that the data selector approach can be applied. This is the case for the widely-used JSON and XML formats. Although this scope of the approach covers most IoT platform interfaces today, there are clear limitations in this design, e.g., binary data encodings cannot be covered.

Also not yet covered in the proposed design for descriptive adaptation is a way to support different authentication protocols on the platform. In the integration mode (a) the authentication is granted by the Marketplace and the consumer receives an access token to be able to request data from the gateway service of the provider. The authentication mechanisms used between gateway and platform are oblique to the consumer. In case of the integration (b), the authentication has to happen directly with the platform. Here, a workflow description for the authentication method employed by the platform can be supported in the future.

Our evaluations of the API have confirmed its usefulness. The performance evaluation showed that the transfer time and data overheads caused by using the API are low. The measured overheads stem from additional functionalities provided by the BIG IoT approach. Mainly, they are due to the validation and transfer of the access token, which is basis of the security mechanism in BIG IoT. Further, the introduced interoperability causes some overhead, as the mapping of received data to semantically-defined outputs requires some computation time. The usability evaluation showed generally positive feedback. The survey confirmed that most developers were able to use the API efficiently. Nevertheless, developers indicated the need for improvements, such as better guidance on choosing semantic terms.

CONCLUSION

With multiple real-world deployments, we have shown how our approach facilitates cross-platform interoperability. Deployments of IoT platforms from different domains and geographic regions can be transparently accessed by an application that utilizes the BIG IoT API. Both, the API libraries and the central Marketplace component are being published as open source under the Eclipse IoT umbrella[9].

In future, the functionalities of the BIG IoT API will be extended, such as operations for event notification and controlling of things. Then, the idea of a Web of Systems with autonomously interacting things [14] can be implemented utilizing the API. To support quality of service capabilities offerings need to be enabled to indicate e.g. their maximum access frequency, which the API needs to enforce.

[9] https://projects.eclipse.org/proposals/eclipse-big-iot

Another future development on our agenda is the incorporation of mechanisms for semantic and syntactic mediation. As the likelihood of mismatches during offering discovery is high, an approach for mediating between queried and available offerings is needed. This could start with simple unit transformations and could expand to advanced semantic concept conversion [15].

Further, validation of offerings on registration is a topic for future work. Currently, the Marketplace does not check the validity of the specification of an uploaded offering description against actual data retrieved from the platform. This may lead to falsely described offerings on the Marketplace. We are working on an automated validation of inputs and outputs with a sample data sets from the platform.

REFERENCES

[1] J. Manyika *et al.*, "The Internet of Things: Mapping the Value Beyond the Hype," *McKinsey Global Institute*, 2015.

[2] A. Bröring *et al.*, "Enabling IoT Ecosystems through Platform Interoperability," *IEEE Softw.*, vol. 34, no. 1, pp. 54–61, Jan. 2017.

[3] A. Bassi *et al.*, *Enabling things to talk, Designing IoT solutions with the IoT Architectural Reference Model*. Springer, 2013.

[4] J. L. Pérez, Á. Villalba, D. Carrera, I. Larizgoitia, and V. Trifa, "The COMPOSE API for the Internet of Things," in *Proceedings of the 23rd International Conference on World Wide Web*, New York, NY, USA, 2014, pp. 971–976.

[5] T. Kubitza and A. Schmidt, "meSchup: A Platform for Programming Interconnected Smart Things," *Computer*, vol. 50, no. 11, pp. 38–49, 2017.

[6] R. Guha, D. Brickley, and S. Macbeth, "Schema. org: Evolution of structured data on the web," *Communications of the ACM*, vol. 59, no. 2, pp. 44–51, 2016.

[7] J. Swetina, G. Lu, P. Jacobs, F. Ennesser, and J. Song, "Toward a standardized common M2M service layer platform: Introduction to oneM2M," *IEEE Wirel. Commun.*, vol. 21, no. 3, pp. 20–26, 2014.

[8] A. Bröring *et al.*, "New Generation Sensor Web Enablement," *Sensors*, vol. 11, no. 3, pp. 2652–2699, 2011.

[9] P. Desai, A. Sheth, and P. Anantharam, "Semantic Gateway as a Service Architecture for IoT Interoperability," in *2015 IEEE International Conference on Mobile Services*, 2015, pp. 313–319.

[10] M. Compton *et al.*, "The SSN ontology of the W3C semantic sensor network incubator group," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 17, pp. 25–32, 2012.

[11] J. Kiljander *et al.*, "Semantic Interoperability Architecture for Pervasive Computing and Internet of Things," *IEEE Access*, vol. 2, pp. 856–873, 2014.

[12] K. Kotis and A. Katasonov, "Semantic Interoperability on the Internet of Things: The Semantic Smart Gateway Framework," *Int. J. Distrib. Syst. Technol. IJDST*, vol. 4, pp. 47–69, Jul. 2013.

[13] A. Bröring, S. Below, and T. Foerster, "Declarative Sensor Interface Descriptors for the Sensor Web," in *ISPRS Archives - WebMGS 2010: 1st International Workshop on Pervasive Web Mapping, Geoprocessing and Services*, Como, Italy, 2010, vol. 38.

[14] F. Michahelles and S. Mayer, "Toward a Web of Systems," *XRDS*, vol. 22, no. 2, pp. 62–67, Dec. 2015.

[15] M. Mrissa, C. Ghedira, D. Benslimane, and Z. Maamar, "A context model for semantic mediation in web services composition," in *International Conference on Conceptual Modeling*, 2006, pp. 12–25.

ABOUT THE AUTHORS

Arne Bröring is a research scientist at Siemens' corporate research and the scientific and technical coordinator of the BIG IoT project. His research interests include the Internet of Things, sensor webs, and semantic interoperability. Bröring received a PhD in geoinformatics from the University of Twente (NL).

Andreas Ziller is research scientist at Siemens' corporate research. Ziller received an MSc from the Technical University of Darmstadt.

Victor Charpenay is a doctoral researcher at Siemens' corporate research. Charpenay received an MSc from the University of Passau and INSA Lyon.

Stefan Schmid is a senior expert in the IoT at Bosch Corporate Research. His research interests are knowledge-based services, machine intelligence, and semantic interoperability. Schmid received his PhD in computer science from Lancaster University.

Aparna S. Thuluva is a doctoral researcher at Siemens' corporate research. Thuluva received an MSc from the Technical University of Dresden.

Darko Anicic is a research scientist at Siemens Corporate Technology. His research interests are the development and standardization of the Web of Things, the Semantic Web, and complex-event processing. Anicic received a PhD in computer science from the Karlsruhe Institute of Technology.

Achille Zappa is a post-doctoral research associate at the National University of Ireland, Galway. His interests include Semantic Web Technologies, Semantic Data Mashup, Linked Data, Big Data Management, and Knowledge Engineering. Zappa received a PhD in Bioengineering from the University of Genoa.

Mari Paz Linares is an assistant professor at the department of Statistics and Operation Research of the Universitat Politecnica de Catalunya (UPC). She received a PhD on dynamic traffic assignment and mesoscopic traffic simulation from UPC.

Lars Mikkelsen is a post-doctoral researcher at Aalborg University from where he also received a PhD in wireless communication networks.

Chrisitan Seidel is a consultant at VMZ Berlin. His interests focus on mobility and smart cities.