The BIG IoT API: Semantically Enabling IoT Interoperability

Arne Broring , Siemens AG

Andreas Ziller , Siemens AG

Victor Charpenay , Siemens AG

Aparna S. Thuluva , Siemens AG

Darko Anicic , Siemens AG

Stefan Schmid , Robert Bosch GmbH

Achille Zappa , National University of Ireland

Mari Paz Linares , Univesitat Politecnica de Catalunya

Lars Mikkelsen , Aalborg University

Christian Seidel , VMZ Berlin GmbH

ABSTRACT

The Internet of Things as the aggregate of all Internet- or Web-connected "things" has reached tremendous size and is still rapidly growing. However, it cannot be seen as one homogeneous infrastructure. In fact, most of such "things" are today within vertical silos of IoT platforms that define their own interfaces. This makes it difficult to build cross-platform IoT applications, which eventually results in lost economic opportunities. Hence, this paper presents the BIG IoT approach to close this gap of interoperability. It comprises an API that gains genericness by employing agnostic access interfaces and semantic models that can be adapted to various IoT platforms. We demonstrate an application that shows how to build an overarching ecosystem with this approach.

NOTE: this is NOT the CAMERA READY, Pre Print version

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics • **Networks** → Network reliability

KEYWORDS

Internet of Things, Interoperability

1 INTRODUCTION

Today, the Internet of Things (IoT) is an increasing commercial reality and connected "things" are already largely outnumbering Web users. Gartner has estimated that by 2020 the number of connected things will have increased from 6 billion to over 20 billion things [1]. Hence, the IoT plays a tremendous role in digitalization efforts of small-, medium-, and large companies. Possible applications and market areas range from smart home, smarter cities, eHealth systems, to autonomous manufacturing and Industry 4.0. In order to provide access to data (and functionalities) of connected things, dozens of IoT platforms have emerged, such as ThingWorx[2], Xively[3], Siemens' MindSphere[4], or Bosch's IoT Suite[5].

However, there is still a crucial issue on the IoT: the missing interoperability between platform protocols and interfaces. Today, IoT platforms are vertically oriented and mostly closed systems. A heterogeneous landscape of standards is used by some IoT platforms (e.g., OMA LWM2M [1], OGC SWE [2], OneM2M [3], or OPC UA 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 [4]. A recent McKinsey study [5]

[1] http://www.gartner.com/newsroom/id/2636073
[2] www.thingworx.com
[3] www.xively.com
[4] https://www.siemens.com/global/en/home/products/software/mindsphere.html
[5] https://www.bosch-si.com/iot-platform/iot-platform/iot-platform.html

estimates that a 40% share of the economic potential of the IoT directly depends on interoperability between IoT platforms.

This work aims at addressing these interoperability issues. Naturally, it is impossible to dictate one language (i.e., protocols and interfaces) for the IoT, since purpose, targets, and domains of IoT platforms differ. So, instead of defining a single new IoT language or standard, this work follows an approach of reaching interoperability through *description* of existing interfaces and protocols. This description needs to be holistic in a sense that it covers the syntax as well as the semantics of the IoT platform's interface. We developed for this purpose the concept of IoT *Offering Descriptions*, with an offering being the resource offered by an IoT platform (either data or functions). This offering description model is aligned with the works of the *W3C Web of Things* (WoT) group[6].

Further, we define an API that enables the discovery of and access to IoT platforms based on their offering descriptions – we call this the BIG IoT API. This API can be seen from two complementary perspectives, as (1) a programming API implemented as libraries (libs) of an SDK and (2) a Web API. Both perspectives are covered by this work. Thereby, the programming API abstracts from the underlying communication protocols defined in the Web API.

The remainder of this paper is structured as follows. Section 2 presents Background, Section 3 describes our approach of a generic IoT API that reaches interoperability through semantic descriptions. Section 4 outlines an application of the implemented API. Section 5 critically discusses our approach. Finally, we conclude this paper in Section 6 and provide an outlook to future work.

2 BACKGROUND & RELATED WORK

// Our approach is conceptually grounded in IoT architecture research such as [7] and [8]. These works provide high-level layering and abstraction concepts, which is valuable for the research community to develop a common set of terms and definitions around IoT architectures. Nevertheless, the BIG IoT approach goes further than a sole architectural proposal, as a concrete API and its implementation as a library is provided. In this sense, our approach is more similar to [9]. However, it is crucial to note that we do not aim to develop yet another platform as it is done there. Instead, we develop an API that is designed to be applied to existing platforms.

3 A SEMANTICALLY-ENABLED API FOR INTEROPERABLE IOT ECOSYSTEMS

This section describes our approach of a generic API that can be used to enable interoperable access to existing IoT platforms. Section 3.1 introduces the involved components to build an IoT ecosystem. The semantic models for Offerings and domain terms are then outlined in Section 3.2 and Section 3.3, respectively. Finally, Section 3.4 details the different functionalities of the API.

3.1 Overview of the IoT Ecosystem

Fig. 1 provides an overview of the interplay of components of an IoT ecosystem based on the BIG IoT API.

A central pillar of the ecosystem is the *Marketplace*. Here, a *Provider* registers their platform resources by uploading an Offering Description (OD) (Section 3.2). To facilitate a Provider in conforming with the BIG IoT API to integrate their IoT platform with the ecosystem, the Provider Library can be utilized. It offers a gateway access to the actual platform. This library implements the BIG IoT API and the various interactions and workflows (Section 3.4). The library authenticates with the Marketplace and registers the offerings of the platform.

A *Consumer* in this ecosystem builds IoT applications that consume offerings of one or multiple IoT platforms. The Consumer discovers suitable offerings by formulating and registering a query on the Marketplace. This standing query is continuously checked by the Consumer to retrieve matching offerings directly from the Marketplace. The conformance with the BIG IoT API is facilitated through the Consumer Library. It authenticates with the marketplace and allows submitting a query as a discovery request and subscribing to resulting offerings. Utilizing libraries on both ends, on consumer- and provider-side, the consumer application can access the discovered platform offerings from a gateway service (a). Alternatively, a fully descriptive adaptation of the IoT platform can be followed, i.e., the OD is enriched with information about how to interpret the protocols of the IoT platform. Then, these protocols are interpreted on consumer-side by the library and an access can be established (b). This is detailed in Section 3.5.

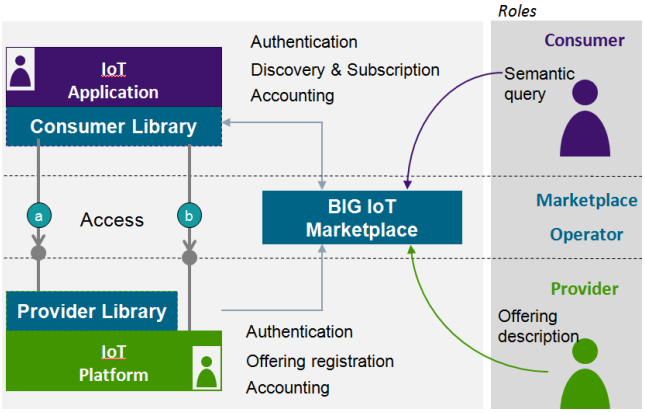

Figure 1: IoT ecosystem components

Beyond these mechanisms for providing access to IoT platforms, the Marketplace enriches the ecosystem with the possibility of monetizing such access. Therefore, both sides, the Consumer and Provider, report access, e.g., download of a dataset, back to the Marketplace, which performs the needed accounting. This is the basis for charging and billing, and the foundation for business opportunities around the ecosystem.

[6] https://www.w3.org/WoT/

The above described IoT ecosystem is platform-scale independent. I.e., IoT platforms can operate either on cloud-level (e.g., server, data center), on fog-level (e.g., gateway, cellular communication base station), or on device-level (e.g., a Raspberry PI, wearable, Smart phone) [6]. The BIG IoT API can be mutually used independent of this scale of the platform.

3.2 Offering Description Model

An *offering* represents a resource offered by an IoT platform or service. 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 that is registered on the Marketplace and made available for discovery by potential consumers. Most essential, the offering description comprises information on how to access the described resources. On one hand this entails the syntactic definition of data types needed as input to access the resources as well as data types contained in the output as a response of the access. On the other hand, these inputs and outputs are semantically annotated to enable discovery and correct matchmaking between consumers' demands and providers' supply.

Additionally, the OD comprises metadata about the offering that supports the discovery. This includes information about the spatial extent (e.g., a city or region) of the represented resources, the license of the provided data, or the price for accessing the resources. For being able to describe offerings and register them on the Marketplace, we have defined a lightweight ontological model [7] as shown in Fig. 2. Offering Descriptions, as instances of this model, are stored together with the ontology in a triple store in the backend of the Marketplace. The model is based on the *Thing Description* model [8], which is being developed in the W3C Web of Things Working Group[7].

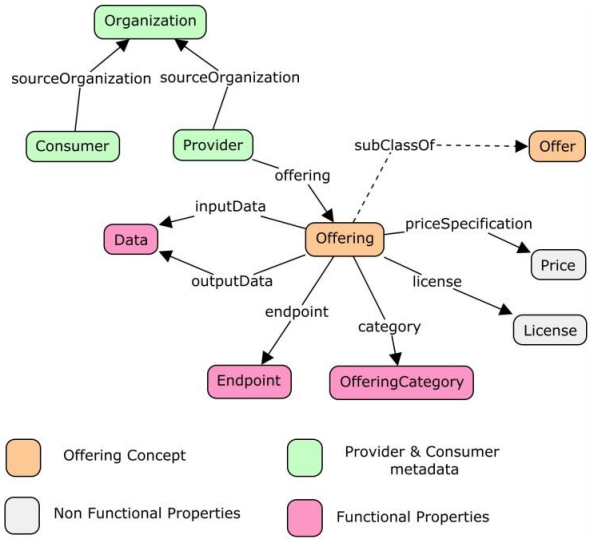

Figure 2: Overview of the offering model (Source: [7])

At the heart of the model is the Offering, which is owned by a Provider organization. As we align our models with schema.org (see also Section 3.3), it inherits from "schema:Offer" Class. An offering has a category that puts it in context and can be used for discovery as a search keyword. Further, the offering has endpoints details that specify the address, protocol and method (e.g., HTTP POST or CoAP GET) that define how it can be accessed. Crucial for enabling the access to the offering is the definition of its input and output data. While input data describes the parameters needed to successfully query the offering under the specified endpoint, the output data describes the types used in the response. Additionally, the offering defines non-functional properties such as price and license that support discovery and selection of suitable offerings.

The Listing 1 shows an example of an Offering Description encoded in JSON-LD [9] that provides outside temperature measurements and hence declares to belong to the *airTemperature* category.

The *inputData* and *outputData* defined for an Offering are crucial for reaching interoperability through the BIG IoT API. Both elements define clear syntax and semantics of the inputs required by the IoT platform and the outputs used to form responses. Looking at Listing 1, three inputData elements are defined: latitude, longitude, and radius, with the semantic annotation "schema:latitude", "schema:longitude", and "schema:geoRadius". The syntax of all three inputData is defined as number according to JSON Schema [10]. The outputData is defined as a complex structure, an array called "tempMeasurements" that contains multiple objects, each one having two members: "result" and "unitOfMeasure". Section 3.4 further explains the mapping of these OD specifications to the protocol and messages.

```
{"@context" : ["https://big-iot.org/ctx.jsonld"],
  "schema" :     "http://schema.org",
  "providerId" : "WeatherGuru",
  "name" :       "My-Temperature-Offering",
  "category" :   "bigiot:airTemperature",
  "endpoints" : [{
      "schema:url" :  "http://my-server/temp/access",
      "schema:type" : "HTTP", "method" : "POST",
      "mediaType" :   "application/json"} ],
  "inputData" : [
    { "name" :      "latitude",
      "rdfType" :   "schema:latitude",
      "valueType" : "number"},
    { "name" :      "longitude",
      "rdfType" :   "schema:longitude",
      "valueType" : "number"},
    { "name" :      "radius",
      "rdfType" :   "schema:geoRadius"
      "valueType" : "number" } ],
  "outputData" :
    { "members": [ {
      "type":    "array",
      "name":    "tempMeasurements",
      "rdfType": "schema:QuantitativeValue",
      "value": {
        "type": "object",
        "members": [
          { "name":    "result",
            "rdfType": "schema:value",
```

[7] https://www.w3.org/2016/12/wot-wg-2016.html

```
          "value": { "type": "number" } },
        { "name":    "unitOfMeasure",
          "rdfType": "schema:unitCode",
          "value": { "type": "string" } } ] } } ] } }
```

Listing 1: Example of an Offering Description

3.3 Domain Models 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 a schema.org approach as a basis for our domain model, as it provides a vendor-neutral, community-developed vocabulary for structured data. Nowadays, schema.org markup is widely used on millions of Web pages to semantically annotate their content. Its approach is beneficial for Web publishers because it allows improving the presentation of their content within search engines. Today, the core of schema.org covers a broad selection of terms ranging from e-commerce, over media, up to sports activities. In addition, other domains are covered as schema.org extensions (e.g., auto.schema.org or helath-lifesci.schema.org). IoT-related applications are heterogeneous and are inter-related with other application domains. So, many of the existing schema.org terms are also highly relevant for IoT applications. While schema.org is so far not covering specific IoT terminology, it is in process of extending its vocabulary to IoT devices and services within the new iot.schema.org extension that is developed within a W3C Community Group.

For the use cases from the smart mobility domain addressed in this work (Section 4), we have developed an extension called schema.big-iot.org/mobility. It includes definitions for relevant concepts, such as parking site, parking space, charging, or bus stop. Listing 2 shows the definition of the concept mobility:ParkingSpace.

```
mobility:ParkingSpace a rdfs:Class ;
  rdfs:label "ParkingSpace" ;
  dc:source <http://vocab.datex.org/terms#ParkingSpace> ;
  schema:isPartOf <http://schema.big-iot.org/mobility> ;
  rdfs:subClassOf mobility:ParkingFacility .
```

Listing 2: ParkingSpace definition in RDF/Turtle notation

As stated by the predicate dc:source, the term is mapped to a Linked DATEX II corresponding terminology[8], a format that is widely used in traffic information systems and is available as an linked open data vocabulary since 2016. This statement is for documentation and traceability purposes. The statement involving the predicate schema:isPartOf is used for categorization and specifies the namespace of the term.

All term definitions within a namespace are stored in an RDF store in one named graph. When the HTML documentation for a term is requested, the Web server queries the distant RDF store via SPARQL and renders the documentation dynamically. Fig. 3 shows the rendered page for mobility:ParkingSpace conform to the schema.org style of documentation.

Our implementation of the documenting Web pages has been designed along the same lines as the original schema.org[9] pages as users are used to this interface and there is a linked integration with the core of schema.org. Hence, our pages expose the same structure and layout - the main change is on how the vocabularies are managed in the background. Instead of having a core and extension vocabularies, our implementation simply defines namespaces, of equal importance and taken from the same source repository.

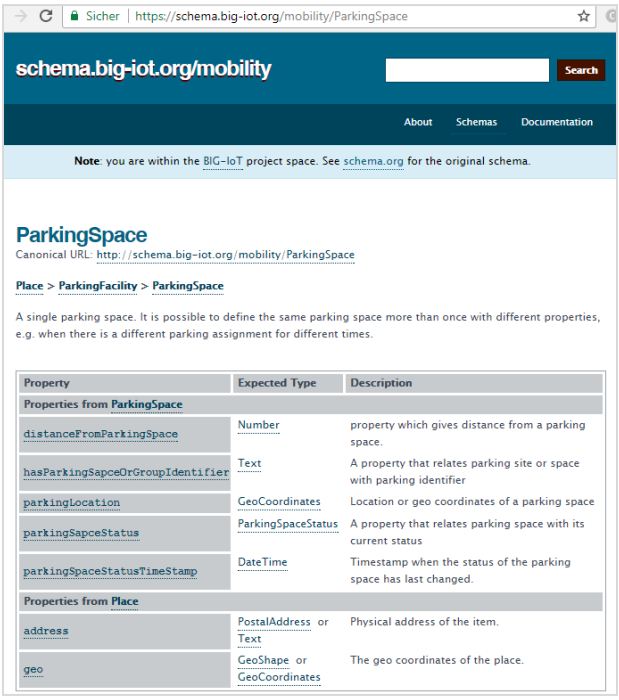

Figure 3: Documentation of the defined term ParkingSpace conform to schema.org

3.4 The Generic IoT API

As outlined in Section 3.1, there are two kinds of integration modes supported by the BIG IoT API: (a) provider-side integration by implementing a gateway service using the library to proxy the access to the existing IoT platform and (b) consumer-side integration through a fully descriptive adaptation of the existing platform interface using an enriched Offering Description. While this section describes the BIG IoT API at hand of the mode (a), the details on mode (b) are described in Section 3.5. This section further focuses on the specification of the Web API and excerpts of

[8] http://www.datex2.eu

[9] https://github.com/schemaorg/schemaorg

its implementation as a programming library are described in Section 4.

In the integration mode (a) the BIG IoT enablement of an IoT platform foresees the communication of the BIG IoT Consumer Lib with a Provider Lib, which is used by a gateway service in the environment of the provider. A data access of the consumer will be accepted by the gateway service logic, which proxies the call to the IoT platform. The application logic for that callback is implemented by the provider. Fig. 4 shows the typical workflow of accessing an offering in this provider-side integration.

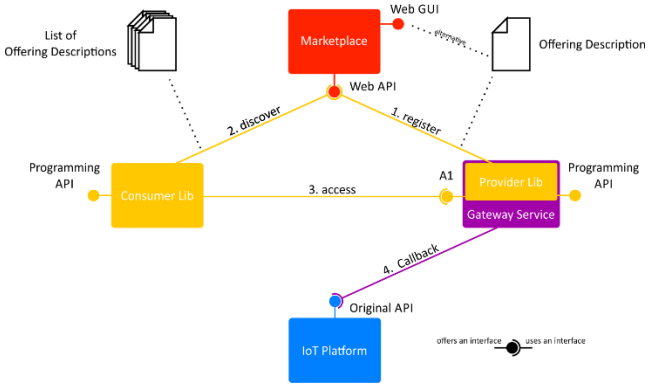

Figure 4: Workflow of provider-side integration mode (a)

In step 1, the provider registers the offering, i.e., an OD, at the marketplace, either programmatically or through the Web page of the Marketplace. Step 2 is initiated by the consumer for the discovery of offerings with a search query. A discovery may return multiple ODs conform to Section 3.2. The actual access to an offering happens in step 3. The offering's individual callback implementation in the gateway service accepts this call of the ACCESS interface and executes it in step 4 with the correct syntax and semantics to the IoT platform. I.e., this ACCESS interface is offered by the BIG IoT Provider Lib and used by the Consumer Lib. The specification of the ACCESS interface is shown in Table 1.

Table 1: Parameters of the ACCESS interface

Parameter	Type	Location	Required
Request			
token	String	header	yes
<input data>	mediaType	message	no
Response			
<output data>	mediaType	message	yes

The first field, *token*, authorizes the user. It is located in the header of the HTTP request and is retrieved from the Marketplace.

The *input data* and *output data* are key to the genericness of the BIG IoT API. They are filled by instances of the inputData or outputData elements as specified by the provider in the OD. In case of input data, each of these elements defines clear syntax and semantics of what parameters the IoT platform requires to be able to respond to a data retrieval request. The *input data* included in the request are encoded according to the mediaType specified in the OD. In the example of 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}*. In case of integration mode (a), the transformation of these input parameters to the parameters of the existing (adapted) IoT platform is then implemented by the platform provider.

Similarly, the *output data* are encoded according to the specified mediaType. In the example of Listing 1, the response from the gateway service is encoded in JSON and contains an array called tempMeasurements comprising objects such as *{"result": 23, "unitOfMeasure": "degC" }* according to the definition in the OD.

Besides retrieving data in the above described request/response manner (ACCESS interface), subscribing to data as a feed is important for IoT applications to retrieve forth-coming data on a regular time basis or on availability. This continuous way of accessing a data stream is supported by the FEED interface of the BIG IoT API. In case an offering provides a FEED interface, it marks the according endpoint in its OD.

Data feeds can be either synchronous or asynchronous. In case of synchronous feeds, the data is transmitted to the consumer in a regular time interval independently of the timely behavior of the emitting of new values on the data source. Asynchronous feeds foresee transmissions to the consumer whenever new data is emitted or the data producer decides to update its feed subscribers. Typically, synchronous feeds are good if the update rate of the data source is high and the application requires a lower data renewal frequency, e.g. because a data gauge in the UI has to be updated with a new value. As the timing behavior is deterministic here, the update is realized by periodic data polls initiated by the consumer. Asynchronous data feeds are beneficial if the data emitted at the provider has an irregular event character and the notification of the feed subscriber shall be immediate. As it is not predictable when new data is available, new data is transmitted here to the client when it is emitted. However, the producer must be able to initiate a transmission towards the consumer, which may not be possible, e.g., due to firewall rules.

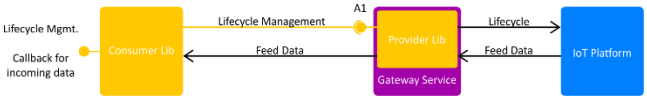

Figure 5: FEED interface of the BIG IoT API

Fig. 5 gives an overview of the FEED interface implementation in the BIG IoT API. Before a data feed can be retrieved, the lifecycle management of the feed has to be configured. I.e., the communication has to be initiated by subscribing to the feed, it may need to be paused, or entirely stopped. Table 3 shows the required parameters for the subscription to a FEED interface. It is agnostic

to the concrete interface protocol binding. Same as for the ACCESS interface, a security token and input data need to be provided to query for the FEED data. In addition, a lifetime (in seconds) of the feed subscription can be specified. In return from the consumer receives a successCode and a subscriptionId from the provider through which the actual data can then be retrieved.

Table 2: Parameters of the FEED interface

Parameter	Type	Location	Required
Request			
token	String	header	no
lifetime	Integer	message	no
<input data>	mediaType	message	no
Response			
successCode	String	message	yes
subscriptionId	String	message	yes

3.5 API mechanisms for Descriptive Adaptation

In Section 3.4, the usage of the API is described by utilizing a provider-side integration of the IoT platform. This integration mode requires the operator of the platform to implement and run a gateway service using the Provider Lib. To prevent this burden on the provider, the BIG IoT API also supports consumer-side integration through a fully descriptive adaptation of the existing platform interface. This integration mode (b) has a slightly different workflow, which is shown in Fig. 6: accessing the IoT platform (step 3) is performed directly from the Consumer Lib to the original API of the existing platform.

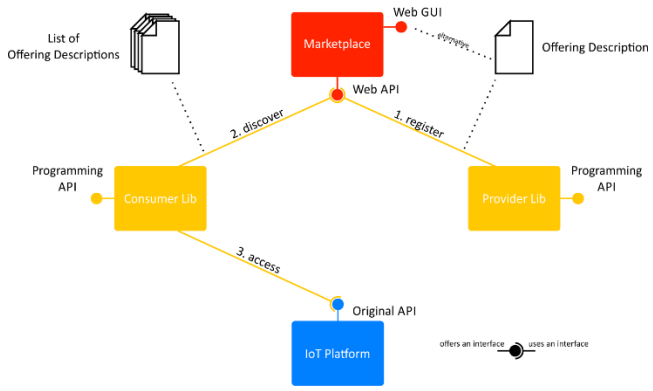

Figure 6: Workflow of consumer-side integration mode (b)

Naturally, this direct integration requires more information about the call conveyed to the consumer than in the mode (a). The information has to specify the exact *syntax* of the request to access data, i.e., how it is structured and formatted. Alongside, the *semantics* of the elements in the call need to be conveyed to the consumer so that a correct understanding of the call request can be formed. We are providing this information based on a simple, efficient and flexible *template approach* for the request and a *selector approach* for the response.

The example in Listing 3 shows a data access request of an existing IoT platform encoded in XML. However, instead of containing actual values, the content of the query-defining elements, latitude, longitude and radius, contain placeholders (marked between "@@" signs). This is essential for our approach, as the Consumer can now use this original request template for querying data from the platform by replacing the placeholders with actual values. This template and how to use it is described in the OD (Listing 5) and detailed in the following.

```
<?xml version="1.0"?>
<WeatherRequest>
  <observedProperty>
    schema:temperature
  </observedProperty>
  <location>
    <latitude>@@lat@@</latitude>
    <longitude>@@long@@</longitude>
    <radius>@@rad@@</radius>
  </location>
</WeatherRequest>
```

Listing 3: Original request call to access data from IoT platform including template placeholders highlighted in red

Listing 4 shows the response call of the IoT platform. In this example it is an XML SOAP-encoded message that contains temperature measurements and a unit in its body. The Consumer needs to be equipped with the information on which data elements should be selected for which outputs defined in the OD to enable applications to meaningfully use the data. This is described in the following.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
<soap:Body>
  <Measurements>
    <Measure id="1">23</Measure>
    <Measure id="2">42</Measure>
  </Measurements>
  <Unit>degC</Unit>
</soap:Body>
</soap:Envelope>
```

Listing 4: Original response from IoT platform

The excerpt of the OD is shown in Listing 5. The object "requestTemplate" contains the template shown in Listing 3. Each inputData contains a placeholder property (here only the input latitude is shown) that is used in the request template. This way the request template elements are linked to semantics as defined by the inputData objects (e.g., "@@lat@@" is of type "schema:latitude"). Instead of using templates, the mapping of response elements to outputData elements utilizes the data selector approach. This is due to the fact that responses to data requesting calls typically contain arrays of objects (e.g., multiple temperature measurements). This

can be more efficiently handled with selectors. E.g., in Listing 5, the temperature-value output is linked to the data selection "*Envelope.Body.Measurements.Measure*". These selectors represent paths in a tree data structure and are similar to XPATH [11].

```
"inputData" : [
  { "name" :        "latitude",
    "rdfType" :     "schema:latitude",
    "valueType" :   "number",
    "placeholder": "@@lat@@"},
  ...
  }],
"requestTemplate": "<as defined in Listing 3>",

"outputData" : [
  { "name" :        "temperature-value",
    "rdfType" :     "schema:value",
    "valueType" :   "number",
    "selector":     "Envelope.Body.Measurements.Measure"},
  { "name" :        "unit",
    "rdfType" :     "schema:unitCode",
    "valueType" :   "xsd:string",
    "selector":     "Envelope.Body.Unit"} ]
...
```

Listing 5: Excerpt of Offering Description including placeholder template linking and data selectors

The crucial challenge of this integration mode is to allow the adaptation of potentially arbitrary platform interfaces, protocols and data formats. Since there are nearly unlimited variations of Web APIs for IoT platforms, only a certain set of classes of such interfaces can be supported by this approach (see analysis in Section 5).

While the above example describes the descriptive adaptation for the ACCESS interface, the approach of using templates and placeholder linkage to semantics in the OD is the same for realizing the FEED interface. This way, the BIG IoT API can rely on the platform's original feed mechanism for data transport, while in integration mode (a) an own feed data mechanism between Provider Lib and Consumer Lib needs to transport the feed data. The original feed mechanism would be then used between the Provider Lib in the Gateway Service and the existing IoT platform. The benefit of integration mode (b) is that no repacking of feed data is necessary. However, integration (a) allows more control of the feed mechanism, e.g., in order to circumvent networking restrictions.

4 APPLYING THE BIG IOT API

The BIG IoT API and its implementation as a library have been utilized so far to integrate 10 IoT platforms with over 50 offerings, which are publicly accessible on the Marketplace [10]. In the following, we present as a proof-of-concept a consumer application that utilizes the BIG IoT API to access in an interoperable way different parking platforms.

The application (ParkFinder) comes as a result of a cooperation project between SEAT and UPC. The ParkFinder user will be able to get real time parking availability by filtering different types of parking places and organizing preferences (distance, price and the likelihood of finding parking) [12]. Once users simply enter where they are headed, the app connects to the on-board unit of SEAT cars using Mirror Link Technology, Car Play or Android auto[11] to get real time information about time to destination, distance and guidance to the parking spot (Fig. 7).

The key benefit for SEAT 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 different parking data providing IoT platforms (vendor-neutral). In this proof-of-concept, we implemented two parking data providers for Barcelona and Berlin.

Figure 7: Screenshot of SEAT application (Source: [12])

Listing 6 shows a simplified excerpt of the implementation of the gateway service for the IoT platform in Barcelona to provide parking data. The Java code instantiates and authenticates a provider, creates an Offering Description and sends it encoded in JSON-LD (see Listing 1) for registration to the Marketplace. These easy to use method calls allow specifying metadata such as the provider's IP address and security token, as well as the offering's category, region, license type, and the syntax and semantics of its inputs and outputs. Further, callback object is defined. It is executed once the offering is accessed and implements the business logic to respond with the correct parking spot information.

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

RegistrableOfferingDescription offeringDescription =
 provider.createOfferingDescription("MyOfferingID")

 .withInformation("BCN Parking", "
schema:parkingSpaceManagement")
 .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("parkingX", "schema:longitude", NUMBER)
 .addOutputData("parkingY", "schema:latitude", NUMBER)
```

[10] https://market.big-iot.org/

[11] https://www.android.com/auto/

```
 .addOutputData("status",    "
http://schema.org/parkingSpaceStatus", STRING)

 .withAccessRequestHandler(accessCallback);

offeringDescription.register();
```

Listing 6: Excerpt of provider code using Provider Lib

Listing 7 shows a simplified excerpt of the implementation of the SEAT application that discovers offerings providing parking data via the Marketplace and accesses them using the lib. After instantiating and authenticating a consumer, the query for offerings is created. Here, the consumer searches for offerings of the category "schema:parkingSpaceManagement" and with the inputData elements to specify user location and a radius. After retrieving a list of matching offerings from the Marketplace and subscribing to one, the offering is accessed by defining parameters according to the current user location.

```
Consumer consumer = new Consumer(ID, MARKET_URI);
consumer.authenticate(CONSUMER_SECURITY_TOKEN);

OfferingQuery query = OfferingQuery.create("ParkQuery")
 .withInformation("Parking Query",
"schema:parkingSpaceManagement")
 .inRegion(Region.city("Barcelona"))
 .withLicenseType(LicenseType.OPEN_DATA_LICENSE)
 .withInputData("schema:latitude", NUMBER)
 .withInputData("schema:longitude", NUMBER)
 .withInputData("schema:geoRadius", NUMBER);

List oList  = consumer.discover(query).get();
Offering o1 = oList.get(0).subscribe().get();

AccessParameters parameters = AccessParameters.create()
 .addRdfTypeValue("schema:latitude", 41.3889295)
 .addRdfTypeValue("schema:longitude", 2.1354683)
 .addRdfTypeValue("schema:geoRadius", 500);
AccessResponse r = o1.accessOneTime(parameters).get();
```

Listing 7: Excerpt of consumer code using Consumer Lib

5 DISCUSSION

This section analyzes benefits and limitations of the above described approach.

The case of the SEAT application (Section 4) shows the advantages of the presented approach and how this bridging of the interoperability gap has the potential of creating great economic value, as suggested by McKinsey [5]. By adhering to the BIG IoT API, various IoT platforms (e.g., parking information from various platforms) can be transparently accessed. Once according semantic models are defined, platforms from multiple domains (e.g., smart home, mobility, health) and regions can be combined. The semantic descriptions of IoT offerings can be flexibly registered on a BIG IoT-conform marketplace. This creates new opportunities as application development is facilitated and offerings can be easily discovered.

Thereby, 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. Regarding the domain models, we also aim at an alignment with standardization efforts and they have been based on the widely adopted schema.org vocabulary. 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 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 mobility, smart city, environment, or health.

The approach of the BIG IoT API gains its genericness by having generic input and output data fields whose syntax and semantics are defined by a description. The possible semantic definitions are restricted by the allowed domain vocabulary. Conceptually this is a delegation of the interoperability problem from the API to the models. Instead of defining a large API that covers all possible IoT platform interfaces, a semantic model is developed that needs to do this. The key difference however is that the semantic model is easier to extend and maintain in a community process. The successful example of schema.org that is used by millions of Web pages proves this.

The fully descriptive adaptation of IoT platforms as presented in Section 3.5 can significantly facilitate the integration into the ecosystem. Particularly when thinking this concept further towards tools that support, through intelligently designed user interfaces, the creation of the ODs enriched with protocol templates and data selectors. Once such an enriched OD is defined for a specific communication protocol (e.g., LWM2M [1] or OneM2M [3]) it can be even shared with other platform providers whose platform uses the same protocol.

In its current design, the fully descriptive approach can be applied to arbitrary Web-protocols, e.g., HTTP, CoAP, MQTT, or WebSockets. The outgoing messages (i.e., data request) 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 can be arbitrary formats, however, they have 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 currently. Also, multi-step data access cannot be approached with this design, e.g., a platform interface requires a first message to define which sensor will be accessed and in a second step the actual data is queried.

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 a security token to be able to access 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.

6 CONCLUSIONS & FUTURE WORK

In summary, we present in this paper a holistic approach for closing the interoperability gap on the IoT. Therefore, we present a Web API that is implemented as libraries of an SDK, which realizes the needed workflows within an IoT ecosystem (e.g., discovery, subscription, access, and accounting). The API is generic by incorporating a semantic annotation approach to specify input and output data of IoT offerings, as well as for their general metadata. For our application domains of mobility and environment, we present extensions of the popular schema.org models that we use to specify the semantics of IoT offerings. In order to bring an existing IoT platform into the ecosystem, we present two alternative modes: provider-side and 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, which enables a fully descriptive integration.

The design and functionalities of the BIG IoT API are currently covering the request/response access and the continued feeding of data streams. Reaching these operations is represented by different end points in the Offering Description. Still missing are operations for tasking of things, i.e., steering a device such as lifting a robots arm, or an event notification mechanism. Event notification can be seen as an extension of feeds whereby data is only received if certain rules trigger.

ACKNOWLEDGMENTS

This work is financially supported by the project "Bridging the Interoperability Gap" (BIG IoT) funded by the European Commission's Horizon 2020 research and innovation program under grant agreement No 688038.

NOTE : this is NOT the CAMERA READY, Pre Print version

REFERENCES

[1] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification, Candidate," OMA, 2015.

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

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

[4] W. Schladofsky *et al.*, "Business Models for Interoperable IoT Ecosystems," in *Interoperability and Open-Source Solutions for the Internet of Things: Second International Workshop, InterOSS-IoT 2016, Held in Conjunction with IoT 2016, Stuttgart, Germany, November 7, 2016, Invited Papers*, I. Podnar Žarko, A. Broering, S. Soursos, and M. Serrano, Eds. Cham: Springer International Publishing, 2017, pp. 91–106.

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

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

[7] A. Thuluva, A. Bröring, G. P. Medagoda Hettige Don, D. Anicic, and J. Seeger, "Recipes for IoT Applications," presented at the The 7th International Conference on the Internet of Things (IoT 2017), Linz, Austria, 2017.

[8] S. Kaebisch and T. Kamiya, *Web of Things (WoT) Thing Description*. W3C, 2017.

[9] M. Lanthaler and C. Gütl, "On using JSON-LD to create evolvable RESTful services," in *Proceedings of the Third International Workshop on RESTful Design*, 2012, pp. 25–32.

[10] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of JSON Schema," in *Proceedings of the 25th International Conference on World Wide Web*, Republic and Canton of Geneva, Switzerland, 2016, pp. 263–273.

[11] J. Clark and S. DeRose, *XML Path Language (XPath)*. W3C, 1999.

[12] C. Baumgartner and C. Seidel, "BIG IoT - Deliverable 5.4a: Demonstration and Assessment." European Commission - H2020.