TDLIoT: A Topic Description Language for the Internet of Things

Ana Cristina Franco da Silva[1(✉)], Pascal Hirmer[1], Uwe Breitenbücher[2], Oliver Kopp[1], and Bernhard Mitschang[1]

[1] Institute for Parallel and Distributed Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{franco-da-silva,hirmer,kopp,mitschang}@informatik.uni-stuttgart.de
[2] Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
breitenbucher@informatik.uni-stuttgart.de

Abstract. In recent years, the Internet of Things has emerged as an important paradigm that comes with the digitization within all domains. Cheap hardware devices and powerful network connections enable the deployment of a large number of devices equipped with sensors and actuators to enable effective monitoring and management of environments. Famous examples are smart cities or smart factories. Usually, the access and control to sensors and actuators in the IoT is conducted through topics, which originate from the publish-subscribe pattern. Topic consumers can subscribe to these topics or publish information in order to build IoT applications that monitor sensor values or control actuators. However, in large environments, such as smart cities, it is usually not known which topics exist to build applications, and, furthermore, which functionality they provide. In this paper, we introduce the Topic Description Language for the Internet of Things that enables a description of topics. Furthermore, we show how they can be accessed and used. We introduce a real-world scenario in the domain of smart cities to validate the practical feasibility of our approach.

Keywords: Internet of Things · Publish-subscribe Description Language

1 Introduction

In the last years, advances in the Internet of Things (IoT) [22] have enabled the deployment of innumerous applications, such as situation recognition in the IoT [29], smart homes [24], smart factories [30], or smart cities [14]. Those applications are based on the monitoring of distributed sensors and the control of actuators to react on occurring situations. For example, the detection of an unoccupied parking space in a smart city could lead to the notification of drivers nearby through an application on their smart phone. In order to develop such

T. Mikkonen et al. (Eds.): ICWE 2018, LNCS 10845, pp. 333–348, 2018.
https://doi.org/10.1007/978-3-319-91662-0_27

IoT applications, it is required to know (i) which sensors and actuators exist in the context of the application, (ii) what data a sensor provides and in which format, (iii) which actions can be triggered for actuators, and (iv) how sensors and actuators can be accessed to be used.

In the IoT, the access to sensors and actuators is usually realized through the publish-subscribe communication model [17] based on *topics*. These topics are used to provision sensor values or enable access to actuators. In the scope of this paper, a topic is an entity that allows to receive and send data in a uniform manner. Topics could be realized through various communication models, such as publish-subscribe or request-response, using different protocols. In the topic-based publish-subscribe model, e.g., realized through MQTT [2], subscribers register on topics relevant for their context to get asynchronously notified when publishers send messages to these topics [12]. In the request-response model, instead of getting notified, applications need to request the data, e.g., through HTTP requests. In this paper, we support both models.

The management of subscriptions and delivery of messages is usually realized by middlewares, such as FIWARE [28], Mosquitto [23], OpenMTC [31], or RMP [16]. For example, given that sensor values are provided through topics hosted in such IoT middlewares, using the publish-subscribe model, a smart city parking application subscribes to topics of all sensors monitoring parking spots and gets notified once a parking spot is detected as *unoccupied*. An overview of IoT middlewares is provided by Mineraud et al. [25] and Guth et al. [15].

Currently, application developers depend on knowing everything about the topics they can use to build IoT applications. This information, which includes, for example, the data structure or how it can be accessed, is usually only known if the application developers own the involved devices, sensors, and actuators. Other available topics, which could provide additional information about the environment, are oftentimes not considered but could lead to a significant improvement of the application, e.g., through a higher coverage by additional sensors.

The goal of this paper is to provide means to describe topics in the IoT with all their characteristics. Furthermore, we provide a *topic catalog*, similar to UDDI [32] in web services, that enables finding available topics, which can be used to build specific IoT applications.

To enable this, we propose the *Topic Description Language for the Internet of Things (TDLIoT)*. The TDLIoT provides a simple means to describe and find topics of sensors and actuators by (i) a holistic description of the topics, (ii) a topic catalog to browse the topic descriptions, and (iii) an effective way to find suitable topics that offer access to sensors and actuators. In this way, IoT application development can be eased through an abstraction from specific IoT middlewares. The topics can be found, for example, based on a specific location, sensor or actuator type, data types, or data units. We derive requirements for the TDLIoT through expert interviews as well as a scenario in the smart city domain. These requirements form the basis for our prototypical implementation.

The remainder of this paper is structured as follows: Sect. 2 introduces a motivating scenario. Section 3 defines requirements for our approach. Section 4 describes the TDLIoT. Related work is described in Sect. 5. Finally, Sect. 6 concludes the paper.

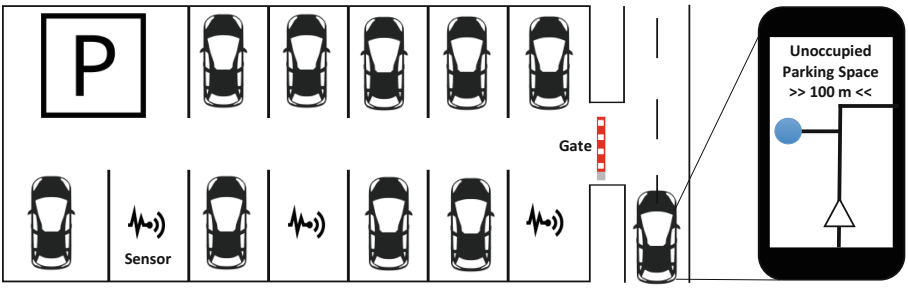

Fig. 1. Motivating scenario: smart parking application

2 Motivating Scenario: Smart Parking

Nowadays, due to a growing amount of vehicles, it is difficult to find a free parking space in large cities. To cope with this issue, the goal of this scenario is to build an IoT application that finds available parking spaces in the nearby location of drivers and to notify them once a parking spot is available. Furthermore, this application can be used by the city itself, e.g., to find out whether vehicles are parked for a long time on time-limited parking spots. The scenario is depicted in Fig. 1 and is inspired by the smart parking application by *Communithings* [10].

Building such an application requires parking space sensors, which monitor parking space availability, and a smart phone application, which notifies drivers or employees of the city about currently used or unoccupied parking spaces. The sensor data detecting whether a parking space is occupied or available is provided through topics the smart parking application can subscribe to. We assume that a large amount of such topics are available that provide information about parking spaces. Those topics can either be provided as open data, e.g., by the city or by external providers that sell the data to applications.

In our scenario, it is important to note that the smart parking application developer is not the owner of the sensors, thus, does not know how to access them and which data is produced. As soon as drivers enter an area in which they want to park, the smart phone application can use the current GPS coordinates to search for available parking space sensors in the area and to check the sensors for unoccupied parking spaces. Furthermore, the driver can be automatically notified by a push notification if an unoccupied parking spot appears in the area. Once a parking spot is available, the driver could trigger actuators through the smart phone applications, for example, to open gates to the parking space. In case of the city employees who aim for monitoring of parking spaces, meter maids could get automatically notified once a car is parked longer than allowed.

As mentioned above, in order to develop such an application, topics are used for the communication between sensors and actuators. However, if there are many topics available, it is currently difficult to find the most relevant ones. Furthermore, heterogeneous sensors could exist for different parking areas. Those sensors might publish different data formats and structures to the topics. This needs to

be known by the IoT application developer in order to properly parse the published data. In a conventional approach to realize such a scenario, usually an IoT middleware is provided – either through the application developer or through an external provider. This middleware connects to the sensors and actuators. Furthermore, it provides access to them through topics. However, a lot of communication effort is required between the provider of the IoT middleware and, in our scenario, the smart parking application developer: First, the paths (or identifiers) of the provided topics need to be known by the application developer, the protocols used (e.g., MQTT [2] or HTTP) and, furthermore, which sensors and actuators are represented through the topics. Second, the concrete structure of the message has to be familiar. This, however, leads to high costs for application development, and, furthermore, to a potential vendor lock-in regarding the middleware provider. Moreover, if new sensors or sensor types are available, new topics are required to be defined, which may lead to changes in the application.

Consequently, building this or similar IoT applications efficiently, requires means to find suitable topics solely based on sensor or actuator metadata. Through these means, the communication effort between middleware providers, who make topics available, and application developers could be significantly reduced. To enable this, we introduce the Topic Description Language for the IoT. We describe our concepts and approach based on this scenario throughout the paper.

3 Requirements

In this section, we specify requirements for the TDLIoT. These requirements originate from experiences with the above described scenario, as well as from expert interviews in the scope of the German industry projects SmartOrchestra [3] and IC4F [1]. These projects have many industry partners with expertise in IoT applications. The goal is the definition of a commonly accepted language that is applicable for most use cases. The expert interviews and the literature research resulted into the following five requirements:

(*A*) **Genericness.** The IoT is very heterogeneous with respect to the existence of different devices, sensors, and actuators. In order to develop a future-proof topic description, the TDLIoT is required to be generic in a way that it supports all kinds of devices, sensors, and actuators, and is not restricted on specific kind of hardware.

(*B*) **Technology independency.** The technologies used in the IoT are very heterogeneous. Many communication protocols, gateways, and middleware solutions exist to build IoT applications. Therefore, the TDLIoT concepts need to be technology independent in order to be realized, for example, using different protocols.

(*C*) **Self-containment.** A TDLIoT description needs to be self-contained, which means it contains all required information to describe, find, and access topics of sensors and actuators. Consequently, it should not be necessary to access external sources to retrieve additional information about the topics of interest.

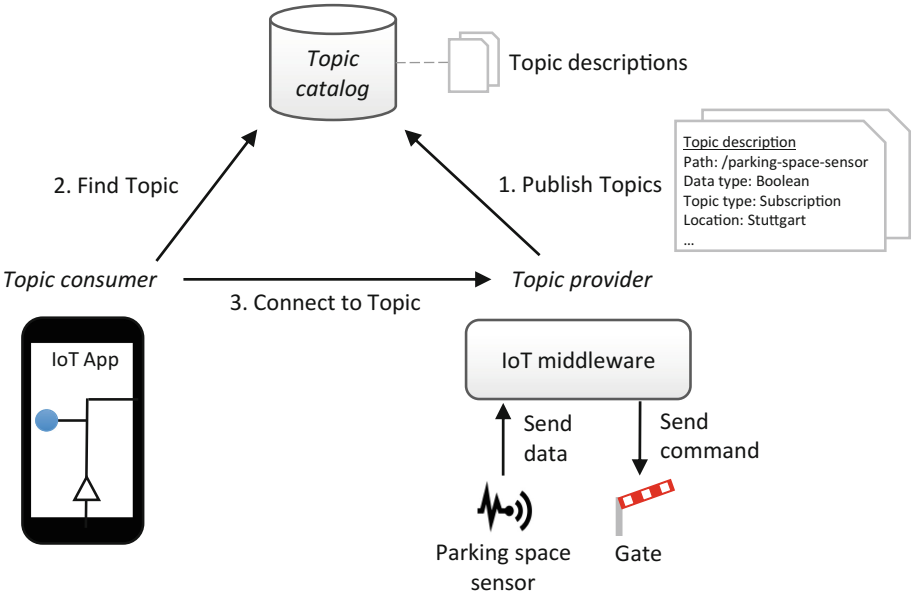

Fig. 2. Overview of the approach for topic description and retrieval

(*D*) **Lightweightness.** The TDLIoT needs to be lightweight in order to support IoT scenarios with limited resources. More precisely, the size of TDLIoT descriptions should be compact, which requires choosing lightweight data formats, such as JSON or YAML. Furthermore, the TDLIoT should be kept as simple as possible to reduce the required effort to get familiar with it.

(*E*) **Extensibility.** The TDLIoT should be extensible with new attributes in the future due to the constantly changing IoT resources. Hence, high extensibility is of vital importance.

4 Topic Description Language for the IoT

In this section, we introduce the Topic Description Language for the IoT and how it can be employed by IoT applications. Our approach is depicted in Fig. 2. It is composed of three main roles: the *topic provider*, the *topic consumer*, and the *topic catalog*. The topic provider creates *topic descriptions* based on the TDLIoT and publishes them to the topic catalog. The topic consumer searches for interesting topic descriptions in the topic catalog, and directly connects to topic providers to either publish data or receive the data published to subscribed topics. These roles are described in detail in Sects. 4.2, 4.3 and 4.4.

This approach is inspired by the Service Oriented Architecture (SOA) Triangle [32], which provides a means to discover and bind services. In Sect. 5, we explain how we differ from the traditional SOA approach. In the following

sections, we explain in detail (i) how to describe topics using the TDLIoT, and (ii) how to publish and retrieve topics from the topic catalog.

4.1 Topic Description

In this section, we describe the content and structure of a *topic description* using the TDLIoT. Each topic description contains several attributes. Their values are represented as strings. We use examples from the motivating scenario described in Sect. 2 to clarify the TDLIoT attributes:

- **data type** the data type of the values provided by the topic, e.g., *boolean*, which is *true* if the parking space is occupied or *false* otherwise
- **hardware type** *(optional)* the type of hardware represented by the topic, i.e., a specific sensor or actuator, e.g., *occupation detection sensor*
- **location** the location of the sensor or actuator represented by the topics. It contains the *location type*, e.g., GPS or a specific city name, as well as the *location value*, e.g., specific *GPS coordinates*
- **message format** the format of the message provided by the topic, e.g., *JSON*, *YAML*, or *XML*
- **message structure** the structure of the message defined as metamodel to understand its content. It contains the *metamodel type*, e.g., *JSON schema* or *XML schema* and the specific *metamodel*
- **middleware endpoint** the endpoint of the IoT middleware hosting the topic, e.g., the endpoint of a message broker running on a server
- **owner** the name of the topic provider, e.g., *City of Stuttgart*
- **path** path of the topic, e.g., */parking-space-monitor*
- **protocol** the communication protocol being used, e.g., *MQTT* or *HTTP*
- **topic type** the type of the topic, i.e., *subscription* or *command*
- **unit** *(optional)* the unit of the data provided through the topic, e.g., *Celsius* if the topic provides temperature values in degree Celsius

Figure 3 shows the TDLIoT metamodel as entity-relation model in the notation of Chen [8]. This model describes the relations between the entities of TDLIoT descriptions. Apart from the *hardware type* and *unit* attributes, all of its entities must only occur once within a single TDLIoT description. However, these entities could occur in an arbitrary amount of descriptions, hence, they are not globally unique. In case topics provide aggregated data, originating from different sensors, or data that does not originate from hardware at all, the *hardware type* attribute is not required.

A special case for the TDLIoT is a sensor that measures different values at once and sends them within a single message, i.e., a multisensor. An example is the sensor ZigBee ZBS-121[1], which measures temperature, brightness, and detects motion. Consequently, the attributes *data type* and *unit* would have to occur multiple times, i.e., for each measured value of the multisensor. In order

[1] ZigBee ZBS-121: http://www.pikkerton.de/_objects/1/26.htm.

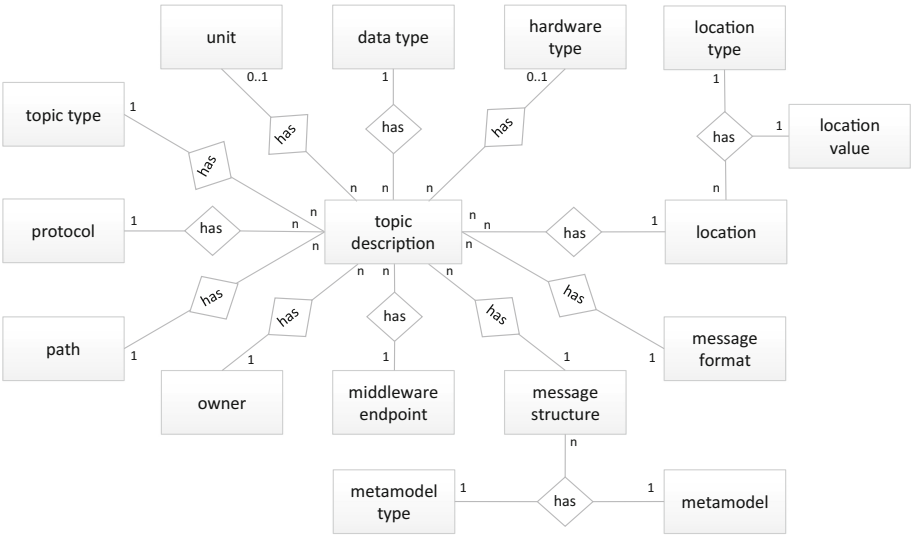

Fig. 3. Data model associated with the TDLIoT

to avoid matching issues, the data type attribute should, for example, have the value *MultiSensorDataType*. Note that the *data type* attribute can be useful in the TDLIoT, even though this information is redundant in the *message structure* attribute. When searching for suitable topic descriptions, the data type of the (e.g., sensor) values could be an important criteria. This attribute directly provides the underlying data type and omits a message parsing overhead. Furthermore, the *topic type* attribute indicates whether this topic can be subscribed to or whether it is used to publish command messages, e.g., to actuators.

To prevent conflicts or confusion regarding the values of the TDLIoT attributes, ontologies can be used to detail the semantic description of these values, such as a specific hardware type having more than one name. For example, the type *occupation detection sensor* could also be named *parking sensor* or *PSensor* in different topic descriptions. Grangel-González et al. [13] introduce such a ontology-based vocabulary for the IoT based on ontologies, which can be used as foundation for this work. Furthermore, other semantic relations can be expressed through the ontology, for example, whether the represented hardware is a sensor or actuator. Using such a vocabulary enhances the TDL with semantics, i.e., the meaning of attribute values can be understood through reasoning approaches. Without this semantic information, finding topic descriptions for specific use cases can become difficult.

```
{ "data_type": "boolean",
  "hardware_type": "occupation detection sensor",
  "location": {
      "location_type": "city name",
      "location_value": "Stuttgart"
```

```
  },
  "message_format": "JSON",
  "message_structure": {
    "metamodel_type": "JSON schema",
    "metamodel":"{"title": "provider_schema",
                  "type": "object",
                  "properties": {
                     "value": {"type": "boolean"},
                     "timestamp": {"type": "integer"},
                     "time_up": {"type": "string"} },
                  "required": ["value", "timestamp"]}"
  },
  "middleware_endpoint": "http://example.com",
  "owner": "city-of-stuttgart",
  "path": "/parking-space-monitor",
  "protocol": "MQTT",
  "topic_type": "subscription"
}
```

Listing 1.1. Example of a topic description based on JSON

When creating topic descriptions, topic providers can query such an ontology in order to find out the specific attribute values they can use, e.g., the type name of the hardware represented by the topic. Only names appearing in the ontology vocabulary should be used in the topic descriptions. New names, for example, of a new type of sensor, should be added to the vocabulary first. The goal of Grangel-González et al. [13] is to accomplish a common vocabulary for the IoT, which all involved partners (e.g., sensor manufacturers) agree on. In their case, they specifically focus on the domain Industrie 4.0. However, such a vocabulary could be provided for a wide range of domains. We do not focus on the specific characteristics of such ontologies since it has already been intensively discussed by Grangel-González et al. [13].

As an example for the TDLIoT, we show a topic description for the motivating scenario based on the JavaScript Object Notation (JSON) in Listing 1.1. We choose this format because it is lightweight and established in the IoT domain. In this example, a topic description based on the TDLIoT is shown for an occupation detection sensor. This description contains all the information necessary to connect and subscribe to the represented topic. Through the *endpoint* attribute value, the topic can be reached using the defined protocol, in this case MQTT. This information is sufficient to subscribe to this topic and receive messages. To develop IoT applications that can automatically process these messages, the *message structure* is described inside the topic description using JSON schema. This metamodel contains information about the data structure and the data types. The JSON schema can be used to derive or implement parsers for the messages received from the topic. The other attributes of the description, such as location, path, hardware type, or owner, can be used to search for the topic description in the topic catalog. For example, an IoT application developer

aiming to implement the smart parking application of our scenario might search for all topics representing sensors of type *occupation detection sensor* in *Stuttgart*.

4.2 Topic Provider

In our approach, the topic provider can be any entity that owns hardware that either produces data (e.g., sensors) or that allows to perform certain activities (e.g., actuators), and wants to make this hardware available to further applications. In our motivating scenario, a topic provider is a parking area owner that deployed several sensors in parking spaces to monitor the current occupancy state. Providing access to these sensor data allows application developers to create applications to, for example, dynamically detect unoccupied parking spaces.

The topic provider usually provides an IoT middleware, which hosts and manages topics for sensors and actuators. This is depicted in Fig. 2 at the bottom right. Furthermore, the topic provider knows specific information about their sensors and actuators (e.g., hardware type) and how to access them (e.g., endpoint, protocol). In order to enable sensors and actuators to be discovered as topics, the topic provider creates topic descriptions for them using the TDLIoT. An example of a topic description for an occupancy detection sensor can be seen in Listing 1.1. This description contains all information necessary to find topics and bind to a selected topic.

The topic descriptions can be published to the topic catalog and are retrievable by topic consumers. Furthermore, once a topic is published, the topic providers can either update a topic description, e.g., when the endpoint to the IoT middleware changes, or delete a topic, for example, when the topic owner does not want to allow access to its hardware anymore.

4.3 Topic Consumer

In our approach, the topic consumer can be any entity that wants to receive sensor data or send commands to actuators. For example, sensor data could be used to build IoT applications, such as a monitoring dashboard [18]. In our motivating scenario, a topic consumer is an application developer aiming to develop a GPS-based smart phone application to detect unoccupied parking spaces near to the current location of the drivers.

To do so, the developer requires access to sensors detecting parking space occupancy, and actuators that can be controlled, for example, gates that enable access to parking spaces and that can be automatically opened. Imaging multiple parking spot providers that have different sensors and actuators deployed and require different access protocols. Topic consumers need to find the most suitable topics for their applications, which could depend on the location of the parking spot, the data types of the sensor values, or the protocol used. In order to develop such applications, the topic consumer searches for interesting topic descriptions in the topic catalog, for example, for topic descriptions of *occupancy detection sensors* at the location *Stuttgart*. An example for such a search can be seen

```
POST /topics HTTP/1.1
Content-Type: application/json

{ "data_type": "boolean",
  "hardware_type": "occupation detection sensor", ... }

HTTP/1.1 201 CREATED
topic_id: 7321
```

Listing 1.2. Example of publishing a new topic to the catalog based on JSON

in Listing 1.4. Once topic consumers find relevant topic descriptions, they can directly bind to the corresponding topics. The necessary binding information, i.e., the endpoint, the topic names, the required message structure, and protocol, is contained in the topic description.

It is also possible that two different topics exist that represent the same sensor or actuator. For example, one topic could provide the sensor data in the JSON format, the other one in an XML representation. The topic consumer can then choose the most fitting topic description to build the desired application. Furthermore, based on the topic descriptions, topic consumers could automatically generate code stubs for the binding to the corresponding topics through parsing the message structure from the corresponding catalog entry.

4.4 Topic Catalog and REST API

The topic catalog contains a data store for topics described in TDLIoT notation. It provides a REST API to enable topic providers to publish topic descriptions and topic consumers to retrieve those descriptions. To publish a topic to the topic catalog, topic providers first need to model their topics in the description structure presented in Sect. 4.1 and exemplified in Listing 1.1 and then submit these descriptions to the topic catalog through its REST API.

An example of how to use the API to submit a new topic to the catalog is shown in Listing 1.2. In this example, the topic depicted in Listing 1.1 should be added to the topic catalog. To do so, a HTTP POST request is sent to the topic catalog, in this example, being available through the resource URI */topics*. The header of this request contains the data format of the TDLIoT description, in this case JSON. In the body, the TDLIoT description itself is provided. The response of the request contains a unique id for the created topic, *topic_id*, which is generated by the topic catalog. With the topic id, topic providers can either change a topic description or remove it.

However, changes in the topic descriptions could lead to issues by topic consumers that are currently using the topics, for example, when the endpoint or message structure changes. This also concerns deletion of topic descriptions. To cope with these issues, the topic catalog provides a means to register on changes in the topic descriptions. For that, the topic consumer needs to provide a callback

```
GET /topics HTTP/1.1
Accept: application/json

HTTP/1.1 200 OK
[ {"data_type": "boolean", ...},
  {"data_type": "float", ...}, ... ]
```

Listing 1.3. Requesting all registered topics from the catalog

```
POST /topics HTTP/1.1
Accept: application/json Content-Type: application/json

{ "filters": {
    "location": {
        "location_type": "city name",
        "location_value": "Stuttgart"
    },
    "hardware_type": "occupation detection sensor"
  } }

HTTP/1.1 200 OK
[ {"data_type": "boolean", ...},
  {"data_type": "float", ...}, ... ]
```

Listing 1.4. Requesting occupation detection topics located in Stuttgart

endpoint to which notifications about changes are send to. When using NoSQL databases to store topic descriptions, such notification functionality is usually provided natively, for example, in CouchDB [6]. However, this requires versioning of the topic descriptions. This is not yet considered in our approach, however, it can be achieved by adding an additional version attribute to the TDLIoT.

The topic catalog allows topic consumers to search for interesting topics through the provided API. For that, topic consumers either request a list of all published topics and browse this list manually or conduct a refined search by specifying filters for the topics. Listing 1.3 shows how to use the API to list all registered topics in the catalog. To realize this, a HTTP GET request is sent to the topic catalog URI. The type of the description can be defined by the *Accept* header attribute of the HTTP request. This is especially important if TDLIoT descriptions exist in several data formats (e.g., JSON, XML, etc.). If no accept header is specified, the JSON representation is returned by the catalog.

To realize a refined search, topic consumers specify filters based on the attributes defined in Sect. 4.1. For example, if a topic consumer is interested in all registered *occupation detection* topics located in *Stuttgart*, a refined search by hardware type and location can be used, as shown in Listing 1.4. Based on the specified filter, the topic catalog generates queries that are executed by the data store of TDLIoT descriptions. Finally, the found descriptions are returned

to the topic consumers. When querying the TDLIoT descriptions based on its attributes, the catalog can check the aforementioned ontology (cf. Sect. 4.1) for synonyms of the values in order to query for all suitable topics, even though their attributes are named differently. In this way, topic consumers can also retrieve related topics that fulfill their requirements.

When providing the topic catalog in a centralized manner, this could lead to scalability issues and become a single point of failure. Consequently, a distributed, replicated data store, such as CouchDB, should be used in combination with cloud computing capabilities. In this way, the topic catalog can be scaled throughout a large number of instances (even automatically) and, thus, scalability issues can be reduced significantly.

Similar to web services, after finding suitable topics in the topic catalog, topic consumers still need to negotiate the specific usage terms with the topic providers. For example, access control mechanisms could be in place, which requires a generation of access tokens in order to connect to the topics. Such tokens need to be exchanged directly between topic providers and consumers. Consequently, access control and other specific usage terms need to be handled by the topic consumers and providers themselves. This is not in the scope of the topic catalog.

4.5 Prototypical Implementation

We implemented an open-source prototype available on github under the Apache 2.0 license[2]. The prototype is also available in Docker hub[3]. The topic catalog was implemented using MongoDB[4] to store and manage the topic descriptions, which are in JSON notation. Moreover, we created the proposed REST API to access the topic catalog. The used REST framework is Java Spring[5]. A documentation of the API is provided using Swagger[6]. We provide a Java-based client application to access the topic catalog and subscribe to and publish to topics using the MQTT protocol. Finally, we provide a web user interface, in which the current topic descriptions are visualized and can be edited through the REST API.

5 Related Work

Grangel-González et al. [13] introduce a vocabulary for the Internet of Things, especially for the domains Industrie 4.0 and smart cities. This vocabulary is based on ontologies, thus, defining semantic relations between entities in the IoT, e.g., devices, sensors, and actuators. In our approach, we use this vocabulary as

[2] Github repository: https://github.com/IPVS-AS/TDLIoT.
[3] Dockerhub repository: https://hub.docker.com/r/ipvs/tdl-catalogue/.
[4] MongoDB: https://www.mongodb.com/.
[5] Java Spring: https://spring.io/.
[6] Swagger: https://swagger.io/.

basis for the attribute values that can be used in the TDLIoT descriptions. By doing so, we can avoid confusion or conflicts when creating such descriptions.

A common API for publish-subscribe was proposed by Pietzuch et al. [27], which supports three levels of compliance: (i) the lowest level, specifying abstract operations, (ii) the middle level, describing interactions using a light-weighted XML-RPC mechanism, and (iii) the highest level, enforcing a XML-RPC data model. In our approach, we provide an API as well, however, we aim at facilitating the description and querying of topics. The specific realization of the communication between topic providers and topic consumers is out of scope.

Dai and Wang [11] argue that WSDL does not enable the description of an IoT object with all its information, since WSDL does not provide methods to represent non-functional aspects of services. Therefore, they propose a flexible extension of WSDL to describe non-functional attributes, so that it enables the complete description of a physical IoT object, including what it is and what it can do. In our approach, we aim at the description of an IoT object in the form of topics. In addition, we provide means to search for interesting topics based on IoT object attribute values, such as unit (e.g., Celsius) or location (e.g., Stuttgart).

In service-oriented architectures (SOA) [26,32], similar roles exist to enable building applications based on services. The roles Service Provider, Service Consumer, and Universal Description, Discovery and Integration (UDDI), also referred as *SOA Triangle*, match the described roles topic provider, topic consumer, and topic catalog. The SOA Triangle is an approved means to find services, therefore, we build on these concepts and apply them to topics in the IoT. In the UDDI, different so-called *pages* exist. In the *white* pages, information about the service provider are stored, including how they can be contacted. In the *yellow* pages, services are being classified based on their attributes. This is necessary to find them effectively. Finally, in the *green* pages, the concrete interface specification of the services are contained, i.e., how to access them. In contrast to the UDDI, our approach does not differentiate among these pages. All information contained in these pages is integrated within a single TDLIoT description. Due to the fact that this information is very lightweight, it does not need to be split up onto several pages. The overview can still be kept clear.

The Web Service Description Language (WSDL) [9] is an approved means to describe web services [32]. When using messaging, there is usually a SOAP/JMS binding available [4]. In our approach, we aim for a lightweight, easy to use way to describe topics. The TDLIoT is tailored for topics in the IoT and uses terms specific to it. Realizing these concepts using WSDL is possible, as shown by Dai and Wang [11]. However, an approach specialized to the needs of the IoT (see Sect. 3) could lead to simplified and more intuitive application development.

Jimenez et al. [20] introduce IPSO Smart Objects, an abstraction of communication protocols, such as HTTP or the Constrained Application Protocol (CoAP) [7]. The goal is to provide high level interoperability between devices and connected software applications. Furthermore, the Open Mobile Alliance

Lightweight Specification (OMA LWM2M) [5] is used, which provides a set of management interfaces based on CoAP.

However, even though the IPSO Smart Objects provide a good abstraction to access devices in the IoT, there are no means to browse available devices and to find the most suitable one to develop a specific IoT application. Combining the TDLIoT with resource description standards, such as IPSO, OMA LWM2M, or early approaches, such as UPnP, is currently not considered in our first approach but will be part of our future work.

In summary, we enabled a SOA-inspired approach to describe topics in the IoT. Current approaches extend WSDL in order to achieve this goal. However, in our approach, we aim for a low complexity and a very lightweight description of topics tailor-made for the IoT. This can be achieved through the TDLIoT.

6 Summary and Future Work

In this paper, we introduce the Topic Description Language for the Internet of Things (TDLIoT). It enables a simple means to describe and find topics of sensors and actuators by providing (i) a description of the topics, (ii) a catalog to browse the topic descriptions, and (iii) an effective way to query and find suitable topics. In this way, IoT application development can be eased through an appropriate abstraction from specific IoT middlewares. The topics can be found, for example, based on a specific location, sensor or actuator type. The TDLIoT descriptions build on a data model that defines their content.

Our approach defines three roles. The topic provider, which creates topic descriptions representing their sensor and actuator and publishes them to the topic catalog. The topic catalog, storing the topic descriptions, and enabling effective means to query them. Finally, the topic consumer, searching for suitable topics to build IoT applications. How the topic catalog can be queried is simplified by a REST-based API, which abstracts from the specific query languages.

To show the applicability of our approach, we present a motivating scenario in the domain smart cities, which describes a smart parking application based on topic descriptions. In this application, occupation sensors and actuators to open gates to the parking spaces are represented by corresponding topics described using the TDLIoT. Furthermore, we provide an open-source prototype of our concepts that is available online[7].

In future work, we plan to describe non-functional properties of the topics, such as availability, costs, or quality of service, e.g., the accuracy of a provided sensor value. This can be achieved by extending the TDLIoT with new attributes.

Furthermore, we plan to integrate service level agreements (SLA) [21] so that topic providers and consumers can agree on specific usage conditions. Moreover, through the integration of the introduced topic descriptions into access control frameworks [19], security mechanisms can be provided.

[7] https://github.com/IPVS-AS/TDLIoT.

Acknowledgments. This work is funded by the BMWi project SmartOrchestra (01MD16001F).

References

1. IC4F Research Project. https://www.ic4f.de/
2. MQTT V3.1 Protocol Specification. http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html
3. SmartOrchestra Research Project. http://smartorchestra.de/en/
4. Adams, P., Easton, P., Johnson, E., Merrick, R., Philips, M.: SOAP over Java Message Service 1.0 (2012)
5. Alliance, O.M.: Lightweight machine to machine technical specification. Technical Specification OMA-TS-LightweightM2M-V1 (2013)
6. Apache: CouchDB NoSQL Database. http://couchdb.apache.org/
7. Bormann, C., Castellani, A.P., Shelby, Z.: CoAP: an application protocol for billions of tiny internet nodes. IEEE Internet Comput. **16**(2), 62–67 (2012)
8. Chen, P.P.S.: The entity-relationship model - toward a unified view of data. ACM Trans. Database Syst. **1**(1), 9–36 (1976)
9. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001)
10. CommuniThings: Smart Parking Application. http://www.communithings.com/smartparking/
11. Dai, C., Wang, Z.: A flexible extension of WSDL to describe non-functional attributes. In: 2nd International Conference on E-business and Information System Security, pp. 1–4 (2010)
12. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. **35**(2), 114–131 (2003)
13. Grangel-González, I., Halilaj, L., Coskun, G., Auer, S., Collarana, D., Hoffmeister, M.: Towards a semantic administrative shell for industry 4.0 components. In: Proceding of the 10th International Conference on Semantic Computing (ICSC), pp. 230–237. IEEE (2016)
14. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
15. Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of IoT platform architectures: a field study based on a reference architecture. In: Cloudification of the Internet of Things (CIoT), pp. 1–6. IEEE (2016)
16. Hirmer, P., Breitenbücher, U., Franco da Silva, A.C., Képes, K., Mitschang, B., Wieland, M.: Automating the provisioning and configuration of devices in the Internet of Things. Complex Syst. Inform. Model. Q. (CSIMQ) **9**, 28–43 (2016)
17. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
18. HomeAssistant: Home Assistant. https://home-assistant.io/
19. Hüffmeyer, M., Hirmer, P., Mitschang, B., Schreier, U., Wieland, M.: SitAC – a system for situation-aware access control - controlling access to sensor data. In: Proceedings of the 3rd International Conference on Information Systems Security and Privacy, vol. 1. SciTePress (2017)
20. Jimenez, J., Koster, M., Tschofenig, H.: IPSO smart objects. In: Proceedings of the IOT Semantic Interoperability Workshop (2016)

21. Kearney, K.T., Torelli, F.: The SLA model. In: Wieder, P., Butler, J., Theilmann, W., Yahyapour, R. (eds.) Service Level Agreements for Cloud Computing, pp. 43–67. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-1614-2_4
22. Lee, I., Lee, K.: The Internet of Things (IoT): applications, investments, and challenges for enterprises. Bus. Horiz. **58**(4), 431–440 (2015)
23. Light, R.A.: Mosquitto: server and client implementation of the MQTT protocol. J. Open Source Softw. **2**(13), 265 (2017)
24. McDonald, H., Nugent, C., Hallberg, J., Finlay, D., Moore, G., Synnes, K.: The homeML suite: shareable datasets for smart home environments. Health Technol. **3**(2), 177–193 (2013)
25. Mineraud, J., Mazhelis, O., Su, X., Tarkoma, S.: A gap analysis of Internet-of-Things platforms. Comput. Commun. **89–90**, 5–16 (2016)
26. Papazoglou, M.P.: Service-oriented computing: concepts, characteristics and directions. In: Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE), pp. 3–12 (2003)
27. Pietzuch, P., Eyers, D., Kounev, S., Shand, B.: Towards a common API for publish/subscribe. In: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems (DEBS), pp. 152–157. ACM (2007)
28. Ramparany, F., Marquez, F.G., Soriano, J., Elsaleh, T.: Handling smart environment devices, data and services at the semantic level with the FI-WARE core platform. In: IEEE International Conference on Big Data (Big Data), pp. 14–20 (2014)
29. Franco da Silva, A.C., Hirmer, P., Wieland, M., Mitschang, B.: SitRS XT - towards near real time situation recognition. J. Inf. Data Manag. **7**(1), 4–17 (2016)
30. Tao, F., Cheng, Y., Zhang, L., Nee, A.Y.C.: Advanced manufacturing systems: socialization characteristics and trends. J. Intell. Manuf. **28**, 1–16 (2015)
31. Wahle, S., Magedanz, T., Schulze, F.: The OpenMTC framework - M2M solutions for smart cities and the Internet of Things. In: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1–3, June 2012
32. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR, Upper Saddle Rive (2005)