Extending the Web of Things to standardize Service Discovery in Smart Cities

Tristan Damm
Hochschule Flensburg
Flensburg, Germany
tristan.damm@stud.hs-flensburg.de

Ellen Landschoof
Hochschule Flensburg
Flensburg, Germany
ellen.landschoof@stud.hs-flensburg.de

Lukas Obermann
Hochschule Flensburg
Flensburg, Germany
lukas.obermann@stud.hs-flensburg.de

Louis Richter
Hochschule Flensburg
Flensburg, Germany
louis.richter@stud.hs-flensburg.de

Tjorben Wade
Hochschule Flensburg
Flensburg, Germany
tjorben.wade@stud.hs-flensburg.de

ABSTRACT

European member states are required to make collected data from sensors available to the public while not imposing any standard on the format and scope of the provided information. This greatly decreases the discoverability and usability of that data for developers and thus the public. In this paper, we propose a new solution by creating a standard ontology for the Web of Things (WoT) standard.

To evaluate our approach, a reference implementation was built, consisting of a Thing Description Directory (TDD) and a showcasing real-life client. Our tests verified the approach as successful, while revealing potential problems and difficulties of the JSON for Linked Data (JSON-LD) format.

This paper provides the approach for handling open data in a uniform way, while being easily extendable for new use cases. It thus serves as a foundation for standardization work and further research on this topic.

KEYWORDS

Open Data, Service Discovery, Smart City, Web of Things

1 INTRODUCTION

The deployment of Internet of Things (IoT) devices and sensors in cities has increased significantly in recent years, and these systems play a crucial role in collecting data that provides various information about the environment. For example, sensors can measure the current temperature in the city center, or help a driver locate unused parking spots in the city center. In this paper, we investigate the development of a Web of Things (WoT) solution for the standardization of IoT systems interfaces used by cities.

The problem is that the interfaces of these sensors are not consistent, making it difficult for developers to access the data. A European directive [1] requires member states, among other things, to make the data collected by sensors available to the public, but they have the freedom to decide how to do so. This means that each city can use its own standards, which complicates interoperability. This makes it difficult for developers to create applications that can use data from multiple sources. Therefore, it is necessary to find a solution that allows data from different sources to be combined and a standard format to be set.

The authors of this paper previously conducted a study titled *Survey: Service Discovery in Smart Cities* [2] in which various solutions were examined. In this study, the WoT technology emerged as the best solution. It is used to extend the functionality of IoT and optimize Service Discovery (SD).

The goal of this study was to develop an extended WoT solution that allows cities to combine and make data from different IoT systems accessible. Furthermore, it will be demonstrated using a test environment that the solution can be successfully implemented and utilized.

As part of this paper, a backend system was developed to store and manage sensor information. The backend system uses a format that is based on WoT and has been extended by us. It allows for the integration and processing of data from various sensors and devices. Additionally, a mobile application was created as a frontend system that works with sensor data to test the backend. This application demonstrates how developers can use data from the backend to create applications that work with sensor data. In our example, the application can be used to display temperatures at different locations.

We have shown that it is possible to develop a backend that uses this new standard and that it is possible to create applications that can work with this data. We believe that the development of an extended WoT solution for the standardization of IoT systems interfaces will greatly benefit cities and developers by providing a unified and easily accessible data source.

Initially, important fundamentals will be explained, in order to subsequently delve into the relevant problem. Later on, the solution will be presented and the structure of the corresponding test environment will be explained. Afterwards, we will discuss the work and draw a conclusion.

2 RELATED WORK

One of the main aspects of our research were different SDs. We concentrated on three most suited for our work.

2.1 Semantic Service Discovery

The main aspect of semantic SD is to find services based on an ontology. An ontology is a way of describing how different properties are related to each other. They are used to build a shared vocabulary and unambiguous meaning across a subject area. While many

other SDs only work with functional properties (e.g. accepted message types, required inputs, ...), semantic SD uses functional and non-functional (e.g. performance, cost, reliability, ...) properties alike.

How a semantic SD is structured is highly dependent on the purpose it was built for and who it was built by. Since there is no standardization in place the discovery process as well as the used ontology can be adjusted to fit the developer's needs. This makes it possible for Iqbal, Sbodio, Peristeras, and Giuliani [3] to describe services using *SAWSDL* and *SPARQL* while Ben Mokhtar, Kaul, Georgantas, and Issarny [4] use *Amigo-S* and *S-Ariadne* in their work. There are also a lot of different approaches on how to select suitable services for users [5] [6] [7].

2.2 Quality-of-Service-based Service Discovery

Quality-of-Service-based (QoS) SDs are focused on non-functional evaluation criteria. The QoS aspect includes all measures over the quality of a communication service from the user's point of view (e.g. bandwidth, latency, transmission rate, ...). QoS-based SDs can be divided into subcategories [8]:

- Security-based
- Energy-consumption-based
- Network-infrastructure-based

Additionally, there are hybrid categories which can include a combination of two or all other categories.

The security-based aspect can be omitted for our work since we will be working with open data. Energy consumption and network infrastructure are considerably more important since most sensors have restricted resources.

In most cases, QoS-based SDs are based on other SD approaches like semantic SD. This makes them quite versatile. While the base SD works just like it always does. After a service was found, the QoS-based SD acts as a kind of QoS filter [9]. Thereby services that are not suitable get rejected.

One example of QoS-based SD is the concept of *SARL*, which is based on a peer-to-peer concept that by itself does not cover QoS factors.

2.3 Context-based Service discovery

Another SD is the context-based SD.

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [10]

A context like that can be defined and used to describe different services. Each area of use can be defined by a different context. How that context is built and structured depends on the usage.

If a system takes a context into account, it is called 'context-aware'. It can be used to locate the best possible service for a user.

2.4 Web of Things

One problem with SD in the IoT space is always interoperability and compatibility between different systems. One area where these aspects are highly relevant is the internet. Web technologies must work for a wide range of devices and applications. Thus, the internet has also been used as a source of inspiration for solving such and other problems with IoT devices. This combination is commonly referred to as WoT, which we examined in a previous survey paper [2].

The WoT offers a standard that covers the description of sensors as well as the implementation of SD. These descriptions are kept up-to-date and are very comprehensive compared to the SD structures examined. However, In some places the standard is not yet fully developed.

Another survey paper about the WoT by Sciullo, Gigli, Montori, Trotta, and Di Felice [11] covers various aspects of the specification such as architecture, communication protocols, security, and applications. It also covers challenges and future directions of WoT research.

3 STATE OF THE ART

3.1 Thing Description

Thing Description (TD) is a World Wide Web Consortium (W3C) standard to describe IoT devices and their properties. Developers can use them to describe devices in a clear and standardized way. The description can be used in applications or services to know how to interact with a device. A TD contains different properties and types. These include the type of device, available features or data, used protocols and the type of data a device is providing. An application using data from this device (e.g. the app discussed in this paper) should be able to read this standard and extract the needed information. TDs are written in JavaScript Object Notation (JSON) and use the JSON for Linked Data (JSON-LD) standard to provide the information.

A TD uses different vocabularies to define properties and types. The basic structure is defined by a core vocabulary. This contains a root object called *Thing*. A *Thing* object provides a Uniform Resource Identifier (URI), labels and metadata. In order to interact with a *Thing*, it additionally provides *Interaction Affordances*. There are three types of *Interaction Affordances*:

- *Property Affordances*
- *Action Affordances*
- *Event Affordances*

A *Property Affordance* describes where you can get information provided by the device. As in the example discussed in this paper, this could be the measured values of a sensor. An *Action Affordance* describes how to interact with the device. It should define what the input and output data should look like. An *Event Affordance* describes an interface where you will be notified of new information. It defines the data to subscribe or unsubscribe from an event.

Listing 1: Example for a TD

```
{
  "@context": [
    "https://www.w3.org/2022/wot/td/v1.1",
    {
      "sdsc": "https://sdsc.fly.dev/ontology#",
      "ssn": "http://www.w3.org/ns/ssn/",
      "om": "http://www.ontology-of-units-.../om-2/",
      "om18": "http://www.wurvoc.org/.../om-1.8/",
      "geo": "http://www.w3.org/.../wgs84_pos#",
      "saref": "https://saref.etsi.org/core/",
      "s4envi": "https://saref.etsi.org/saref4envi/",
```

```
      "time": "http://www.w3.org/2006/time#"
    }
  ],
  "@type": "sdsc:TemperatureSensor",
  "id": "https://hansen.de/temperature/1",
  "title": "Measuring station Hansen",
  "geo:lat": 54.789401,
  "geo:long": 9.416152,
  "s4envi:hasFrequencyMeasurement": {
    "@type": "s4envi:FrequencyMeasurement",
    "saref:isMeasuredIn": {
      "@id": "om18:reciprocal_hour"
    },
    "saref:hasValue": 1
  },
  "s4envi:hasTransmissionPeriod": {
    "@type": "s4envi:PeriodMeasurement",
    "saref:isMeasuredIn": {
      "@id": "time:unitSecond"
    },
    "saref:hasValue": 5
  },
  "saref:measuresProperty": {
    "@id": "sdsc:Temperature"
  },
  "securityDefinitions": {
    "opendata": {
      "scheme": "nosec"
    }
  },
  "security": ["opendata"],
  "properties": {
    "temperature": {
      "@type": "sdsc:TemperatureAffordance",
      "ssn:forProperty": {
        "@id": "sdsc:Temperature"
      },
      "type": "number",
      "unit": "om:degreeCelsius",
      "readOnly": true,
      "forms": [
        {
          "href": "https://.../temperature",
          "op": "readproperty"
        }
      ]
    }
  }
}
```

Listing 1 presents an example of a TD defined for a temperature sensor. The type of the sensor is defined by the `@type` property. In this project, two sensor types were defined.

The `sdsc:TemperatureSensor` defines a temperature sensor, while `sdsc:WindSpeedSensor` defines sensors for measuring wind speed.

The `id`-field must contain a unique identifier for the sensor. We are using the URI of thing to identify the sensor. The `title` can be defined by the owner of a sensor. It should contain a name that helps the user understand what the sensor does and where it is located.

Additionally, we added two new fields to the TD's vocabulary to offer the opportunity to set location data. This means that fields `geo:lat` and `geo:long` must be included in the TD in order to provide latitude and longitude for a sensor.

The `s4envi:hasFrequencyMeasurement` field provides information about the time between measurements. It contains a unit in which the duration is measured and the value. A similar field is `s4envi:hasTransmissionPeriod`. This field provides information about the duration of a transmission from the sensor to the server. It contains fields for the value and the unit. Both fields also do not appear in the standard vocabulary.

The WoT standard specifies `securityDefinitions` and its accompanying `security` field to provide data about authentication when accessing a sensor. A public API for fetching data in a smart city was planned. For this reason, a security scheme for open data with no authentication at the server was defined.

The TD shown in Listing 1 contains only one Interaction Affordance. It does not need Action Affordances, because the sensor does not provide an opportunity to change any settings, and Event Affordances, because the sensor does not send any events. The `properties` field contains Property Affordances for temperature or wind speed. It specifies a unit (degrees in this case) and the form from which one can get the actual value for this property.

3.2 JSON-LD

JSON-LD is the format used throughout the WoT, with its main use being TDs. It adds a semantic layer to a JSON structure to let every property be a globally unique identifier – a URI – based on ontologies. JSON-LD provides different ways how to represent data. The most common way is to provide a so-called 'context', so you can use more readable property names for the actual data. The context is a nested JSON structure that maps the property names to the actual identifiers. Listing 1 shows an exemplary TD document, where the WoT ontology [12] and other secondary ontologies are referenced in its context. Some URIs have been abbreviated for formatting reasons.

Property names use terms from a referenced context document, which provides a description and URI for the property. Auxiliary ontologies may be referred to using a prefix. The process of getting the actual structure and identifiers is called "expansion". Its result is a JSON document where each property is a URI. Due to the flexible nature of how to reference ontologies, two documents with the exact same semantic meaning may not have equal JSON-LD representations, i.e. they're not structurally equal. Only if documents are fully expanded, they are structurally comparable (excluding arrays where the order does not matter). Listing 2 shows an excerpt of the expanded example TD document from Listing 1, which is – at least seemingly – larger, due to the URI expansion, and deeper nested.

Listing 2: Example JSON-LD document expanded

```
[
  {
    "@type": [
      "https://...fly.dev/ontology#TemperatureSensor"
    ],
    "@id": "https://example.com/temperature/1",
    "https://www.w3.org/2019/wot/td#title": [
```

```
      {
        "@value": "Nowhere Temperature Sensor"
      }
    ],
    ...
    "https://.../td#hasPropertyAffordance": [
      {
        "@type": [
          "https://...ntology#TemperatureAffordance",
          "https://.../wot/json-schema#NumberSchema"
        ],
        "https://.../wot/td#hasForm": [
          {
            "https://.../wot/hypermedia#hasTarget": [
              {
                "@value": "https://.../temperature"
              }
            ],
            ...
          }
        ],
        ...
      }
    ]
  }
]
```

4 PROBLEM STATEMENT

The data published by cities does not have a uniform format, with both structured formats such as JSON and unstructured formats such as plain text being used, which makes automated processing of the data more difficult [13]. Currently, there is no comprehensive solution to this problem, as existing systems only provide partial solutions or are not specific enough for the Smart City sensor application domain.

Determining whether the WoT and its associated TDs are a good solution is part of our problem, as so far it appears to be the best approach, providing structure and semantic expressiveness for this use case. However, the WoT does not make assumptions about the content of descriptions and ontologies used for Smart City applications are neither standardized nor provide a common vocabulary that covers a wide range of Smart City sensor application domains. Additionally, the WoT does not include QoS aspects, which can be important for SD, enhancing the quality and preciseness of search.

In order to enable search, the WoT provides different search type options that may be implemented [14, section 7.3.2.3]. It is to be validated that at least of these search options is suitable for SD of public sensors.

A challenge is to extend the WoT to solve all the mentioned problems. Additionally, it must also be tested in practice to check for its functionality. In section 5, the implementation of a proof of concept is examined, and it is shown how this solution can be used in a test environment.

5 PROOF-OF-CONCEPT APPLICATION

5.1 Ontology

To overcome this problem, we propose a new ontology that is based on already established ontologies and aligns with the *Thing Description (TD) Ontology* for specific use in TDs and, at the same time, an easy way to extend the ontology if new application areas are needed.

The core element of the ontology is the `Sensor` class, which is built on top of the `Sensor` class from *SAREF: the Smart Applications REFerence ontology* and its extensions from the *SAREF extension for environment*. Also inheriting properties from the *Basic Geo (WGS84 lat/long) Vocabulary*, the `Sensor` class provides important QoS features such as the location, measurement frequency and data transmission period. The class is intended to be sub-classed for each sensor type to have a common set of important values available to retrieve independent of the sensor type.

Each sensor type consists of a set of different classes and named individuals, as seen in Figure 1. This includes the parts which are the same across all sensor types, which makes it easy to extend the ontology with new sensor types.

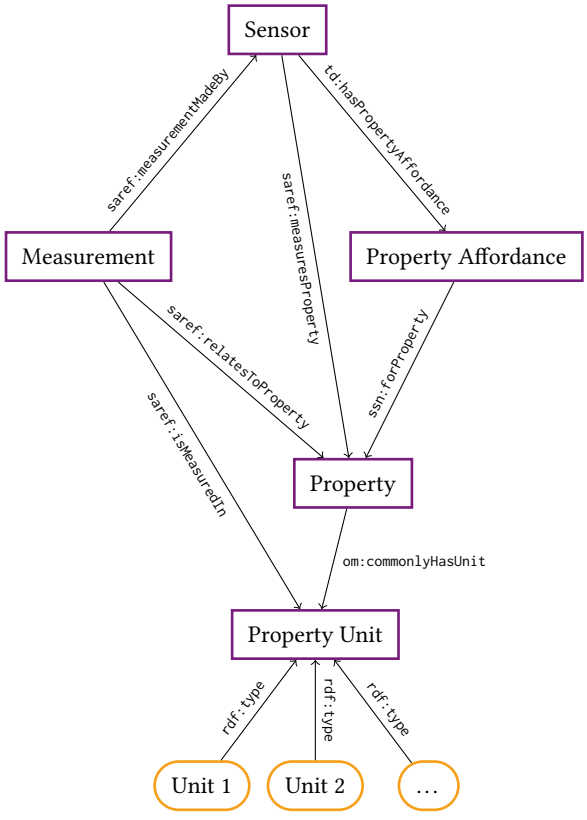

Figure 1: Ontology parts for each sensor type

The class that subclasses `Sensor` represents the root of the TD. The `Measurement` is not a direct part of the TD, which is defined by the other parts mentioned in Figure 1, but serves as the definition of the format used to serve the data of a single measurement. This

ensures a unified data format for all sensors of a type. The `Property` mainly serves as a description of the actual property to measure and to bind the property's unit classes. The `Property Unit` is only used to provide a common type for all named individuals that represent each possible unit of the property.

We applied this principle to multiple measurable values to verify it works for different use cases. Hereafter, we will discuss one of those examples in detail.

Our ontology is defined using Turtle syntax [18], since most of the existing ontologies are also defined using this syntax. To fully understand our ontology, we first introduce the used prefixes and their expanded URI forms in Listing 3.

Listing 3: Ontology prefixes

```
@prefix : <https://sdsc.fly.dev/ontology#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
    ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    .
@prefix saref: <https://saref.etsi.org/core/> .
@prefix s4envi: <https://saref.etsi.org/saref4envi/> .
@prefix ssn: <http://www.w3.org/ns/ssn/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#
    > .
@prefix td: <https://www.w3.org/2019/wot/td#> .
@prefix om: <http://www.ontology-of-units-of-measure.
    org/resource/om-2/> .
@base <https://sdsc.fly.dev/ontology> .
```

The `Sensor` class is the core class (Listing 4). It is only defined once and is sub-classed by `TemperatureSensor`, which is our example sensor type and on which its measured property and the temperature affordance for the TD are defined (Listing 5). The `Sensor` class also makes it possible to include QoS aspects like location, the frequency of measurements and the transmission period.

Listing 4: `Sensor` class

```
###  https://sdsc.fly.dev/ontology#Sensor
:Sensor
  rdf:type owl:Class ;
  rdfs:subClassOf
    geo:SpatialThing ,
    saref:Sensor ,
    [ rdf:type owl:Restriction ;
      owl:onProperty s4envi:hasFrequencyMeasurement ;
      owl:allValuesFrom s4envi:FrequencyMeasurement
    ] ,
    [ rdf:type owl:Restriction ;
      owl:onProperty s4envi:hasTransmissionPeriod ;
      owl:allValuesFrom s4envi:PeriodMeasurement
    ] .
```

Listing 5: `TemperatureSensor` class

```
###  https://sdsc.fly.dev/ontology#TemperatureSensor
:TemperatureSensor
  rdf:type owl:Class ;
  rdfs:subClassOf
    :Sensor ,
    [ rdf:type owl:Restriction ;
      owl:onProperty saref:measuresProperty ;
      owl:someValuesFrom :Temperature
    ] ,
    [ rdf:type owl:Restriction ;
      owl:onProperty td:hasPropertyAffordance ;
      owl:someValuesFrom :TemperatureAffordance
    ] .
```

The `TemperatureAffordance` class then restricts the permitted values for its property affordance to `Temperature` values (Listing 6).

Listing 6: `TemperatureAffordance` class

```
###  https://sdsc.fly.dev/ontology#
    TemperatureAffordance
:TemperatureAffordance
  rdf:type owl:Class ;
  rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:onProperty ssn:forProperty ;
      owl:someValuesFrom :Temperature
    ] .
```

A `Temperature` is defined by a common unit and can be labelled and described (Listing 7). The associated `TemperatureUnit` is defined as a class, which each of its actual units as named individuals (Listing 8). Those individuals are imported from another ontology and mapped onto the `TemperatureUnit` class.

Listing 7: `Temperature` class

```
###  https://sdsc.fly.dev/ontology#Temperature
:Temperature
  rdf:type owl:Class ;
  rdfs:subClassOf saref:Property ;
  om:commonlyHasUnit :TemperatureUnit ;
  rdfs:comment "Temperature is the extent to which an
      object is hot."@en ;
  rdfs:label "Temperature"@en .
```

Listing 8: `TemperatureUnit` class and its named individual(s)

```
###  https://sdsc.fly.dev/ontology#TemperatureUnit
:TemperatureUnit rdf:type owl:Class ;
  rdfs:subClassOf
    om:Unit ,
    saref:UnitOfMeasure ;
  rdfs:comment "The unit of measure for temperature"
      @en ;
  rdfs:label "Temperature unit"@en .

###  http://www.ontology-of-units-of-measure.org/
    resource/om-2/degreeCelsius
om:degreeCelsius
  rdf:type
    owl:NamedIndividual ,
    :TemperatureUnit .
```

The `TemperatureMeasurement` class connects the different parts by referencing the sensor class, the property, its unit and the format, in which the measurement is made (Listing 9).

Listing 9: `TemperatureMeasurement` class

```
###  https://sdsc.fly.dev/ontology#
    TemperatureMeasurement
:TemperatureMeasurement
  rdf:type owl:Class ;
  rdfs:subClassOf
    saref:Measurement ,
    [ rdf:type owl:Restriction ;
      owl:onProperty saref:measurementMadeBy ;
      owl:someValuesFrom :TemperatureSensor
    ] ,
    [ rdf:type owl:Restriction ;
      owl:onProperty saref:relatesToProperty ;
      owl:someValuesFrom :Temperature
    ] ,
    [ rdf:type owl:Restriction ;
      owl:onProperty saref:isMeasuredIn ;
      owl:someValuesFrom :TemperatureUnit
    ] ,
    [ rdf:type owl:Restriction ;
      owl:onProperty saref:hasValue ;
      owl:allValuesFrom xsd:float
    ] .
```

5.2 Backend

The WoT only provides the specification for a Thing Description Directory (TDD) service [14, section 7.3], but there are no full implementations of the standard yet. Therefore, we implemented the relevant parts of the Thing Description Directory specification.

A centralized service is necessary for identifying and accessing the TDs provided by multiple sources. The TD will be obtained through a single Uniform Resource Locator (URL) that leads to a provider service, which returns the TD metadata. The service caches the TD metadata in a database for efficient retrieval and provides an Application Programming Interface (API) for accessing it. Additionally, the service keeps track of the respective providers for each TD and allows them to register and manage their own TDs, as well as store additional information about the providers themselves. Furthermore, the implementation of the service follows the TDD guidelines specified by the WoT.

A SD process starts with a request being sent to a thing provider, which returns the TD metadata. This is stored in a database along with other associated data such as the provider itself. A client can then retrieve one or more TDs via the API.

5.2.1 API. The API is a trimmed down implementation of the WoT Directory Service API and is responsible for retrieving, listing and searching TDs, while also offering an Events API to notify clients about the changes to TDs. The ontology described earlier in this paper is provided over a separate endpoint.

The following list shows all openly accessible endpoints to the clients:

- `/ontology` shows our defined ontology.
- `/api/things` returns a paginated list of TDs.
- `/api/things/{id}` returns a single TD, where the `id` parameter is the URI of the TD.
- `/api/search/jsonpath?query={query}` searches TDs with an *JSONPath* expression as the query parameter.
- `/events/{type}` informs about changes to TDs via Server-Sent Events (SSE) where the type specifies the event.

5.2.2 Admin-Panel. The administration panel is a web app that manages the data of the TDD as well as information about the organizations that provide TD metadata. Organizations administrators can register new TDs and edit their organization information.

5.3 Frontend

We developed an Android application to utilize our API as outlined in the introduction. The app serves as a reference implementation for connecting the backend functions shown in subsection 5.2.

The following requirements were defined by us in advance:

- Filter for geographic location and a radius, minimal frequency of measurements (QoS aspects)
- Quick overview of the sensors in a smart city (around location)
- Capability to display sensors metadata and the measured values

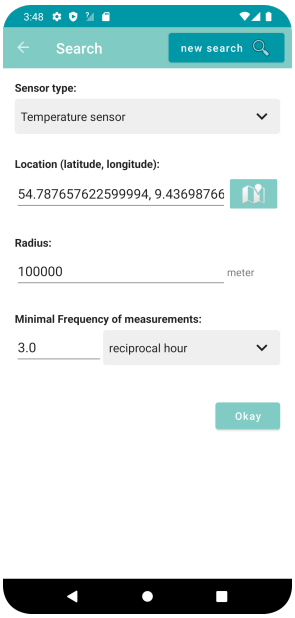

Figure 2: Filter view

In order to meet the requirements, we developed a filter view, which is shown in Figure 2. The view offers the option to filter by sensor type, radius around a location, and minimum frequency of measured values. The location can be specified in longitude and latitude, or selected from a map.

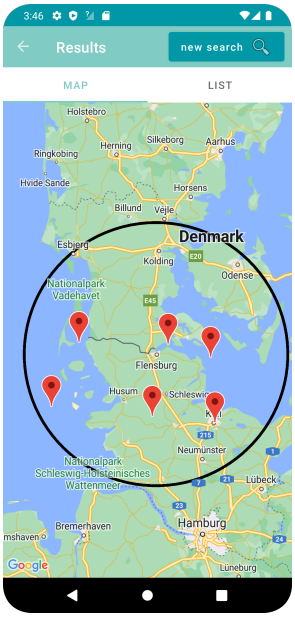

Figure 3: Map view showing the results

The sensors can be presented in a map or list view. This gives the user a quick overview of the surrounding sensors.

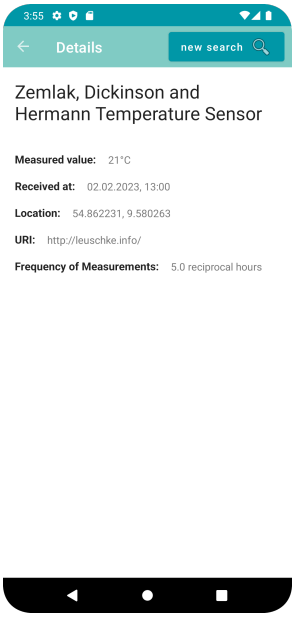

Figure 4: Detail view of a sensor

The measured values and metadata will be displayed in a detailed view. It reads the data from the TD and presents them clearly to the user.

5.3.1 JSON-LD. Since the server may hold heterogeneous data from different providers, the structure of JSON-LD documents may vary, which prevents direct decoding. It first has to be normalized by expanding the data. Normalization with a different method, which could result in simpler data structures, may not work, because it could result in heterogeneous data. For example, multiple values for the same property would be encoded as an array, while a single value would be encoded as-is. Expansion would always result in an array of values, even if only a single value is present.

5.4 Search

As stated earlier, we used *JSONPath* as the search option for our implementation to check whether WoT search is viable and advisable for use in open data contexts.

With *JSONPath*, we cannot filter the TDD by the exact filter options we have, since we cannot filter by a radius. We can only filter all TDs within a bounding box of the circle that would be the result of the radius. The filtering by the actual radius has to be done in a second step on the client side.

Listing 10 shows an exemplary *JSONPath* query that would be used to search for all temperature sensors that are within the bounding box of the circle with a radius of 10 km around the specified center (54.8075525, 9.463745) and that have a minimum measurement frequency of 3 times per hour.

The actual query does not contain line breaks or indentation, but they have been added here for readability reasons. Some additionally required frequency measurement unit filters have been omitted for brevity as well.

Listing 10: Example JSONPath query

```
$[*]?(
  @."geo:lat"<54.8973845209824
  &&
  @."geo:long"<9.6196146650183
  &&
  @."geo:lat">54.7177214790176
  &&
  @."geo:long">9.307875334981617
  &&
  @."@type"=="sdsc:TemperatureSensor"
  &&
  (
    (
      @."s4envi:hasFrequencyMeasurement"."saref:
          hasValue">=3.0
      &&
      @."s4envi:hasFrequencyMeasurement"."saref:
          isMeasuredIn"."@id"=="om18:reciprocal_hour"
    )
    ||
    (
      @."s4envi:hasFrequencyMeasurement"."saref:
          hasValue">=0.33333334
      &&
      @."s4envi:hasFrequencyMeasurement"."saref:
          isMeasuredIn"."@id"=="om18:hertz"
    )
    ||
    ...
  )
)
```

6 RESULTS

In this study a reference implementation for working with sensors in a smart city was developed. It consists of a backend that is able to store and manage sensor information and a frontend in form of an android app to showcase the range of options of a Thing Description Directory and give a context to possible usage.

The WoT is the only standardization in our area of work that enables interoperability. We found that it offers the required expressiveness and can be used for use cases similar to ours. As a result, the data collected by providers and the data requested by users can be connected to each other in a clearer way using the WoT. The hardware and software requirements of the WoT can be easily met due to the fact that the WoT mainly works with technologies that are already widespread (e.g. Hypertext Transfer Protocol (HTTP), JSON, …). The only downside we found is that the WoT does not include QoS aspects. We solved this problem by creating an ontology with obligatory usage that does include them.

The ontology aligns with the *Thing Description (TD) Ontology* for specific use in TDs. It is well suited for our use case and can easily be altered to fit similar works to ours. Due to the uniform class hierarchy for each sensor type it is a straightforward process to extend the ontology to include more or different sensors. Additionally, we based our own ontology on well-established ones. These ontologies have been used in various different ontologies and projects, making it easier to compare our ontology to others and adjust existing ontologies based on our work.

One of the cruxes in our work was the JSON-LD expansion and the handling of expanded data. In most cases, it is best to do this on the server side. There are however use cases where this is not sensible. Especially if the client gets data from multiple servers, it might be necessary to do the expansion on the client side to ensure that the data is parsed consistently. This would however also lead to a complex way of expanding the data. Regardless of where the expansion takes place, it can quickly become complex and there are not enough libraries for it to be easily usable on all typical platforms.

JSONPath and thus syntactic searches are suitable for service discovery, but require that the TDs are normalized on the server side and that the type of normalization is known to consumers of that TDD. For consistency, the normalization should be done using expansion. Difficulties can arise when a property used for filtering can be specified in different units, since all possible units have to be provided a filter for. This complicates the filtering process and makes it more prone to errors. In addition, since multiple servers may use different normalization strategies, queries would not be universally usable. Semantic search, in contrast, is harder to implement, but easier to use, since it does not require normalization, which aligns more with the idea of JSON-LD that has been outlined before. In both cases, client-side normalization is strongly recommended, if not required.

Thing Descriptions for Service Discovery itself is an important step and more work needs to be done be able to use it. Through the usage of ontologies, all data about the sensors is provided in a uniform format. However, when requesting data from a specific sensor the format still depends on the sensor itself.

7 CONCLUSION

We implemented a reference of how to serve TDs and the variety of search options in a WoT context. A mobile application was used to show realistic use cases.

The lack of a standard ontology for Smart City contexts in the WoT specification has been solved by creating a new ontology based on established ontologies. It has been designed for easy extension of missing sensor types and the integration of search-relevant QoS aspects.

Despite the missing ontology, the proven expressiveness of and widespread availability of required hard- and software for the WoT suits SD in open data contexts, providing a unifying concept for making different data endpoints available to a variety of users.

The usage of JSON-LD throughout the WoT imposes normalization and verbosity difficulties on both servers and clients, especially if syntactic search methods are used, due to the possible heterogeneity of multiple TDs.

GLOSSARY

API Application Programming Interface

HTTP Hypertext Transfer Protocol

IoT Internet of Things

JSON JavaScript Object Notation
JSON-LD JSON for Linked Data

QoS Quality of Service

SD Service Discovery
SSE Server-Sent Events

TD Thing Description
TDD Thing Description Directory

URI Uniform Resource Identifier
URL Uniform Resource Locator

W3C World Wide Web Consortium
WoT Web of Things

REFERENCES

[1] Generaldirektion Kommunikationsnetze, Inhalte und Technologien. [n. d.] Das datenportal für deutschland. Retrieved 01/30/2023 from https://digital-strategy.ec.europa.eu/de/policies/psi-open-data?%20-%3E%20nur%20in%20Deutschland%20oder%20EU-weit%20etc.?%20https://digital-strategy.ec.europa.eu/de/policies/psi-open-data.

[2] Ellen Landschoof, Tristan Damm, Lukas Obermann, Tjorben Wade, and Louis Richter. 2022. Survey: service discovery in smart cities. (July 26, 2022). Retrieved 01/30/2023 from https://github.com/elyukai/sdsc-survey-paper.

[3] Kashif Iqbal, Marco Luca Sbodio, Vassilios Peristeras, and Giovanni Giuliani. 12/3/2008 - 12/5/2008. Semantic service discovery using sawsdl and sparql. In *2008 Fourth International Conference on Semantics, Knowledge and Grid*. 2008 Fourth International Conference on Semantics, Knowledge and Grid (SKG) (Beijing, China). IEEE, 205–212. DOI: 10.1109/SKG.2008.87.

[4] Sonia Ben Mokhtar, Anupam Kaul, Nikolaos Georgantas, and Valérie Issarny. 2006. Efficient semantic service discovery in pervasive computing environments. en. In ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing. Springer, Berlin, Heidelberg, 240–259. DOI: 10.1007/11925071_13.

[5] S. Majithia, A. S. Ali, O. F. Rana, and D. W. Walker. 2004. Reputation-based semantic service discovery. In *Proceedings of the thirteenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. June 14-16, 2004, University of Modena and Reggio Emilia, Italy*. Thirteenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (Modena, Italy). IEEE Computer Society, Los Alamitos, California, 297–302. ISBN: 0-7695-2183-5. DOI: 10.1109/ENABL.2004.52.

[6] Bing Jia, Wuyungerile Li, and Tao Zhou. 2017. A centralized service discovery algorithm via multi-stage semantic service matching in internet of things. In *22017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, 422–427. ISBN: 978-1-5386-3220-8. DOI: 10.1109/CSE-EUC.2017.82.

[7] Shuai Zhao, Le Yu, Bo Cheng, and Junliang Chen. 2017. Iot service clustering for dynamic service matchmaking. *Sensors*, 17, 8. ISSN: 1424-8220. DOI: 10.3390/s17081727.

[8] Meriem Achir, Abdelkrim Abdelli, and Lynda Mokdad. 2020. A taxonomy of service discovery approaches in iot. In *2020 8th International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 2020 8th International Conference on Wireless Networks and Mobile Communications (WINCOM) (Reims, France). IEEE, 1–6. ISBN: 978-1-7281-9015-0. DOI: 10.1109/WINCOM50532.2020.9272512.

[9] Selahattin Kosunalp and Kubilay Demir. 2020. Sarl: a reinforcement learning based qos-aware iot service discovery model. *Journal of Electrical Engineering*, 71, 6, 368–378. DOI: 10.2478/jee-2020-0051.

[10] Sagar Sukode, Shilpa Gite, and Himanshu Agrawal. 2015. Context aware framework in iot: a survey. 1-9. Retrieved 06/09/2022 from https://www.researchgate.net/publication/273456830_CONTEXT_AWARE_FRAMEWORK_IN_IOT_A_SURVEY.

[11] Luca Sciullo, Lorenzo Gigli, Federico Montori, Angelo Trotta, and Marco Di Felice. 2022. A survey on the web of things. *IEEE Access*, 10, 47570–47596. DOI: 10.1109/ACCESS.2022.3171575.

[12] Victor Charpenay, Maxime Lefrançois, María Poveda Villalón, and Sebastian Käbisch. 2023. Thing description (td) ontology. (January 13, 2023). https://www.w3.org/2019/wot/td.

[13] FITKO. [n. d.] Das datenportal für deutschland. Retrieved 01/30/2023 from https://www.govdata.de/.

[14] Andrea Cimmino, Michael McCool, Farshid Tavakolizadeh, and Kunihiko Toumura, editors. [n. d.] Web of things (wot) discovery. Retrieved 06/14/2022 from https://www.w3.org/TR/wot-discovery/.

[15] Laura Daniele, Raúl Garcia-Castro, Maxime Lefrançois, and Maria Poveda-Villalon. 2020. Saref: the smart applications reference ontology. (May 29, 2020). https://saref.etsi.org/core/v3.1.1/.

[16] Raúl Garcia-Castro and Maria Poveda-Villalon. 2020. Saref extension for environment. (June 5, 2020). https://saref.etsi.org/saref4envi/v1.1.2/.

[17] Dan Brickley, editor. 2006. Basic geo (wgs84 lat/long) vocabulary. (February 1, 2006). https://www.w3.org/2003/01/geo/.

[18] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. 2014. Rdf 1.1 turtle. Eric Prud'hommeaux and Gavin Carothers, editors. (February 25, 2014). https://www.w3.org/TR/turtle/.