Abstract—The Web of Things (WoT) Thing Description (TD) of W3C is a JSON template representation of Thing properties (e.g., purpose, data types, and operations). In recent work, we proposed that Things can be treated as Web services, and be described using OpenAPI. OpenAPI specification provides a method for documenting RESTful services so that a user or another service can comprehend their purpose and reuse them in applications. The resulting descriptions by either approach (i.e., W3C and OpenAPI) can be semantically extended using ontologies or can be mapped to an ontology to allow a machine to better understand the inherent meaning of Thing descriptions and interact with them. Then, Things can be discovered (i.e., queried) using Semantic Web query languages (e.g., SPARQL) and checked by reasoners (e.g., Pellet) for consistency or, to infer new properties. In this work, both approaches are discussed and compared in terms of the completeness of their representation.

Keywords—Web of Things, Thing Description, Web of Thing Architecture, OpenAPI, Ontology;

I. INTRODUCTION

Nowadays, devices have become part of people’s daily lives in various application domains, such as healthcare, transportation, agriculture, education, environmental purposes, monitoring, physical exercise, and many others. There are more than 20 billion interconnected devices in the world and the number is growing rapidly.

The Web of Things (WoT) initiative [1] aims to unify the world of interconnected devices over the Web. A Thing may refer to any device: a humidity or proximity sensor, a window actuator, a smart refrigerator, or a smart car. WoT suggests that Things should advertise their identity and properties on the Web so that, they can be discovered by Web search engines and be reused in applications. The Semantic Web of Things (SWoT) [2] is the semantic extension of WoT that allows Things to become machine discoverable using Semantic Web tools (e.g., SPARQL).

The Web of Things (WoT) Architecture W3C recommendation [3] defines an abstract architecture and sets the requirements for interacting with Things on the Web using the REST architectural style [4]. The WoT Thing Description (TD) of W3C [5] is a representation of Thing properties (e.g., data types and operations) based on JSON-LD [6]. TDs are used to expose Thing metadata on the Web so that other Things or clients (i.e., services or users) can interact with them. Leveraging the context information mechanism of JSON-LD, WoT Architecture proposes a representation of TDs contents using ontologies [7].

An independent approach to representing Things resorts to OpenAPI [8]. OpenAPI is a description format for REST APIs. It is a widely adopted industry standard endorsed by Linux Foundation and supported by large software vendors like Google, Microsoft, and many others. OpenAPI format is based on JSON (or YAML) and comprises a large set of properties for composing service descriptions. In our previous work [9], Things are described as similar to RESTful services. The resulting description is a lightweight version of the general-purpose OpenAPI description for REST services specialized for Web Things [9]. It is an alternative to the TD of the W3C Web Architecture and offers a more informative and elaborate mechanism for the description of Things exposing their functionality on the Web as RESTful services.

Semantic OpenAPI [10], [11] is a method for converting Web service descriptions to an ontology. The method applies to Web services and Thing Descriptions equally well. The idea of using ontologies is not new. The motivation for using ontologies is that they are closer to the way machines interpret and process information. It is, therefore, more manageable for a machine to search the Web for Things with the desirable properties. Existing ontologies for Web services fit well the needs of remote procedure call technologies such as SOAP [12]. However, the emergence of REST generated new difficulties in the representation of hypermedia-driven APIs (such as REST) that call for the dynamic discovery of resources at run-time (referred to as HATEOAS [13]). This feature is not supported by known service ontologies (e.g., OWL-S [14] for SOAP services).

In this work, the two candidate ontology approaches...
for Thing Descriptions are presented and compared based on their capability to support the entire set of properties foreseen by the WoT Architecture model of W3C. The comparison and the discussion followed, revealed that the Semantic OpenAPI ontology complies with the W3C recommendation. However, the advantages of the OpenAPI approach are many. Things and Web services co-exist and interact with each other in a Web of Things application [15]. OpenAPI achieves a uniform representation of both types of entities. This has a positive impact on the way applications are designed and implemented. The same ontology can be used to describe both, Web Services and Things.

The two ontology approaches for Web Things by W3C Thing Description and OpenAPI are presented in Sec. II and Sec. III respectively. Their comparison in terms of supported features is discussed in Sec. IV followed by conclusions and issues for future research in Sec. V.

II. WoT ARCHITECTURE OF W3C

The Web of Things (WoT) Architecture of W3C sets the requirements for the interaction of clients (e.g. users or services) with Things on the Web. Things offer particular interaction affordances (i.e. metadata showing how a client can interact with Things) such as Properties, Actions and Events. Properties are used to define the state that Things expose (e.g. humidity value). Actions are used to describe the functions that Things may perform (e.g. a smart window that opens and closes). Events are used to represent the transition of Thing's state (e.g. the state property of a smart window turning to open). Interaction affordances are described using Thing Descriptions (TDs).

An Event describes an event source sending asynchronous notifications (e.g. overheating alerts) to the subscribed clients. Events are closely related to subscriptions. A subscription is a result of subscribing to a specific event related to a Thing. The recommendation defines operations for subscribing and unsubscribing to events (e.g. overheating of a device). Clients can subscribe or unsubscribe to an event and receive asynchronous notifications when the event occurs. A client could subscribe, for example, with a Webhook or Callback URI.

The WoT Architecture of W3C proposes the use of hypermedia controls for the interaction of clients with Things. Two kinds of hypermedia controls are used in the W3C WoT: Web Links and Web forms.

Web Links provide navigation affordances that allow clients to discover linked resources. For example, a Web Link may provide a link target attribute and a link relation type to relate a Thing with another resource that is represented by a hyperlink. Web Links can be followed by both users and machines. A link may include (at least) the URI of a resource (i.e. a target resource) which can be followed to fetch the representation of a resource. The recommendation highlights that Web Links are used in the WoT to discover Things and also to express relations with other Web documents. Links are discovered during the interaction of the Web client with a server.

Web forms allow clients to perform particular operations that may change the state of a Thing (e.g. turn on a device) and not just discover resources. The recommendation defines a number of well-known operation types for the Web of Things. Web forms can specify how these operations can be performed. The operations are related to Thing properties (e.g. an operation to read a property or an operation to update a property), to Thing actions (i.e. an operation to invoke an action), and to events related to Things (e.g. an operation to subscribe to an event). Clients are instructed on how to perform these operations by sending a proper request to their submission target.

A. Thing Descriptions (TDs)

The JSON representation of a TD can be enhanced with a context field (i.e. @context) and be converted to JSON-LD. The WoT Architecture suggests hosting TDs in a directory service (on a gateway or the cloud) providing a Web interface for Thing registration and discovery. The architectural aspects of a Thing (e.g. data schemas, security configuration, etc) are also included in the WoT Architecture recommendation.

Listing 1 is an example TD for a smart door actuator containing (a) a context attribute which extends the definition with additional vocabulary terms, (b) the device identifier, (c) an indicative title, (d) the security configuration (Basic Authentication in this example), (e) interactions supported by the smart door; the state property, the lock and unlock actions, the door open event (i.e. the state property of the door turning to open) and, (f) the forms field (i.e. included in every interaction object) that describes how each interaction can be performed; it specifies the protocol that should be used (e.g. HTTPS) and the operation endpoint. The endpoint to retrieve the last state value of the smart door is defined by the Properties object (i.e. in the forms array). The protocols and the endpoints used to execute the lock and the unlock actions are specified by an Actions object. The protocol, the endpoint, and the sub-protocol (e.g. the exact mechanism used for asynchronous notifications) for subscribing to the open event of the smart door are specified by an Events object.

Listing 1: Thing Description for the smart door.

```
{
  "@context": "http://www.w3.org/ns/td",
  "id": "urn:dev:ops:32473-WoTSmartDoor-1234",
  "title": "MySmartDoor",
  "securityDefinitions":{
    "basic_sc": {"scheme": "basic", "in": "header"}
  }
}
```
According to the specification, there is an alignment of Web Links and Web Forms of WoT Architecture and event notifications from a Thing.

The ontology includes an additional object property, the thing property, that refers to the exact mechanism that should be used by a protocol (i.e. when there are multiple mechanism options) to achieve a particular interaction with a Thing (e.g. how the long polling subprotocol subscribes to the overheating event of a temperature sensor).

b) Thing Description (TD) core vocabulary: it describes information related to Things (e.g. information in TDs) and provides alignments with other ontologies related to WoT (e.g. SOSA [17], [18]). It defines an ActionAffordance, an EventAffordance, an InteractionAffordance, an OperationType (i.e. it lists well-known operation types needed to implement the WoT interaction model), a PropertyAffordance and a Thing class. It includes many object properties, datatype properties and named individuals.

The Hypermedia Controls ontology is capable of describing the hypermedia controls included in the TD documents of W3C. Properties of TDs can be mapped to terms (e.g. object properties) of the Hypermedia Controls ontology using the ontology’s namespace IRI. The JSON-LD Context Usage section in the WoT Thing Description recommendation indicates that the href JSON key of TDs is mapped to the hasTarget object property of the ontology. The href term can be used in TDs (i.e. instead of the mapped IRI) to declare how clients can perform an operation. The forSubProtocol data property can describe the exact mechanism that should be used by a protocol (i.e. when there are multiple mechanism options) to achieve a particular interaction with a Thing (e.g. how the long polling subprotocol subscribes to the overheating event of a temperature sensor).

The Hypermedia Controls ontology is capable of describing the hypermedia controls included in the TD documents of W3C. Properties of TDs can be mapped to terms (e.g. object properties) of the Hypermedia Controls ontology using the ontology’s namespace IRI. The JSON-LD Context Usage section in the WoT Thing Description recommendation indicates that the href JSON key of TDs is mapped to the hasTarget object property of the ontology.

B. Thing Ontologies

The TD ontology [7] is still at an early stage of development. It can be used to describe WoT Things and their interaction affordances. The TD ontology imports the Hypermedia Controls vocabulary (or ontology) for Web Links and forms (i.e. the main hypermedia controls in use on the Web). The TD Information model is based on the TD vocabulary and the Hypermedia Controls vocabulary, and also on vocabularies for data schemas and security configurations.

a) Hypermedia Controls vocabulary: it is an ontology for Web Links and Forms. Web forms are similar to the operation descriptions of Hydra vocabulary [15]. The ontology includes an ExpectedResponse class that defines the primary response of a service request, an AdditionalExpectedResponse class that defines the additional response of a service request, a Form class for Web forms and a Link class for Web Links. The ontology also includes various object properties (e.g. the hasTarget object property which refers to the URI, protocol, and method used to perform an operation) and datatype properties (e.g. forSubProtocol that refers to the exact mechanism of a protocol used to receive asynchronous event notifications from a Thing).

This ontology is an attempt to describe the concepts of Web Links and Web Forms of WoT Architecture and presents alignments with the Hydra RDF vocabulary. According to the specification, there is an alignment between links in the Hypermedia Controls Ontology (htcl:Link) and links in Hydra (hydra:Link), which is a vocabulary intended to describe hypermedia-driven Web APIs. In addition, the specification indicates that there is alignment between forms in the Hypermedia Controls Ontology (htcl:Form) and operations in Hydra (hydra:Operation).
be represented by the TD ontology in the form of RDF triples. The Actions object (i.e. field actions) of the TD, for example, can be represented by the hasActionAffordance object property of the TD ontology.

c) Web of Things (WoT) Security vocabulary: it defines API security mechanisms. It includes a SecurityScheme class that defines security schemes in general, an APIKeySecurityScheme class for API key authentication, a BasicSecurityScheme class for basic authentication and an OAuth2SecurityScheme class for OAuth2.0 protocol, among others. The ontology also includes many object properties and datatype properties. In addition, the specification document presents a usage example of ontology.

The WoT Security ontology is capable of describing the API security mechanisms (i.e. used for Things and their operations) included in Thing descriptions. The JSON-LD Context Usage section of the TD specification indicates that the basic JSON key of TDs (i.e. as in Listing 1) is mapped to the IRI that represents the BasicSecurityScheme class of the ontology in a specific JSON-LD file (among other mappings). The term basic can be used directly in TDs (i.e. instead of the mapped IRI) to declare how clients can perform a specific operation (i.e. using Basic Authentication).

d) Data Schema vocabulary: it is an RDF vocabulary for JSON schema definitions. It builds on the main terms defined by JSON schema (i.e. a vocabulary for the annotation and also the validation of JSON documents) to represent schema definitions in RDF format. It provides namespace IRIs for JSON schema keywords and simple axioms. More specifically, the vocabulary includes an ArraySchema, a BooleanSchema, a DataSchema, an IntegerSchema, a NullSchema, a NumberSchema, an ObjectSchema and a StringSchema class. For example, the ArraySchema class defines the metadata describing data of type array and the NumberSchema class defines the metadata describing data of type number. This is indicated by the value array or the value number assigned to the type attribute in DataSchema instances in TDs. The vocabulary also includes a number of object properties (e.g. allOf, anyOf) and datatype properties (e.g. contentEncoding, contentType) for the description of data schemas. This vocabulary is meant to describe the data schemas that represent Things and their interactions (e.g. properties, actions, events).

III. OPENAPI THING DESCRIPTION AND ONTOLOGY

OpenAPI service descriptions comprise many parts (objects). Each object specifies a list of properties that can be objects as well. Objects and properties defined under the Components unit of an OpenAPI document can be reused by other objects or they can be linked to each other (e.g. using keyword $ref). The Info object provides non-functional information such as the name of the service, service provider, license information, and terms of the service. The Servers object provides information on where the API servers are located. Servers can be defined for different operations (locally declared servers override global servers). The Operation object provides the needed information for expressing HTTP requests to the service. It also provides information regarding the HTTP responses of the service. The Security object lists the security schemes of the service which, among others, lists definitions of responses, parameters, Schemas, and more. These are defined under the Components unit of reusable objects.

The Paths object describes the available operations (i.e. HTTP methods) and contains the relative paths for the service endpoints (which are appended to a server URL to construct the full URL of an operation). The Request and Response objects describe the request and the response of an operation (e.g. a specific HTTP status code), its message content, and HTTP headers that a request or response may contain. The Parameters object describes parameters that operations use (i.e. path, query, header, and cookie parameters). The specification supports HTTP authentication, API keys, OAuth2 common flows or grants (i.e. ways of retrieving an access token), and OpenID Connect.

A tool for generating OpenAPI Thing descriptions is implemented in Python Flask as a RESTful service and is available on Github. It applies to any device as long as its functionality can be exposed using REST. As a use case, the complete OpenAPI Thing descriptions for a smart door and a DHT22 sensor device (along with their corresponding JSON files given as input to the mechanism), can be found in the same Github address.

Listing 2 is a schema description of a smart door. The extension property x-refersTo is used to semantically associate the actuator type of the smart door to the SODA ontology. The x-kindOf extension property is used to semantically annotate the schema properties of the Thing (i.e. id, name) with concepts in www.schema.org vocabulary.

Listing 2: Thing Description example for the smart door.

```json

schemas:
  Webthing:
    required:
      - id
    type: object
  x-refersTo:
    'http://www.w3.org/ns/sosa/Actuator'

properties:
  id:
```
Listing 3 shows a delete operation on a subscription based on its subscription identifier. A human may refer to the description of the operation to understand its intended purpose, but a machine needs additional information which is provided by the `x-operationType` extension property. The value of the property is a URL pointing to the concept that semantically describes the operation type. The `Action` type of the `schema.org` vocabulary provides a detailed hierarchy of Action sub-types (e.g., `DeleteAction`) that can be used by the property.

Listing 3: Semantic annotation of a smart door Operation.

paths: '/subscriptions/{subscriptionID}':
delete: tags:
  - Subscriptions
summary: Delete a subscription
description: reject the request with an appropriate status code or remove (unsubscribe) the subscription and return a 200 OK status code.
operationId: deleteSubscription
x-operationType: 'https://schema.org/DeleteAction'
parameters
  - name: subscriptionID
    in: path
description: The id of the specific subscription
required: true
style: simple
explode: true
schema:
  type: string
example: 5fd23facdde6be05da68bcfb
responses:
  '200':
    description: OK
  '404':
    description: Not found

A. OpenAPI Thing Ontology

The semantic meaning of an OpenAPI service description is captured by the OpenAPI ontology (\[10\], \[11\]). The ontology incorporates features of Hydra \[16\] for modeling service operations. Hydra is a promising technology for understanding and constructing Web services that meet the HATEOAS requirement of REST architectural style. Classes together with constraints on class properties are described using SHACL \[19\]. SHACL is an RDF vocabulary that can be used to define classes with constraints on class properties allowing for service descriptions to be validated against the ontology. A specific method, which is used to map (i.e. instantiate) service descriptions to the OpenAPI ontology, is available as a Web Application\[11\] for testing.

The `Document` class of the ontology provides general information about the service (in `Info` class) and, specifies service paths, the entities and the security schemes that it supports. The `Path` class represents (relative) service paths (in `pathName` property). The `Operation` class provides information for issuing HTTP requests. Request bodies are represented by the `Body` class, while, responses are declared in `Response` class specifying the status code and the data returned. The entire range of HTTP responses is represented. The `MediaType` class describes the format (the most common being JSON, XML) of a request or response body data. Class `Operation` refers to a security scheme in `SecurityRequirement` class. Class `Security` has security schemes as sub-classes. Class OAuth2 has different flows (grants) as sub-classes. If the security scheme is of type OAuth2 or OpenID Connect, then scope names are defined as properties.

Schema objects are expressed as classes, objects, and data properties using SHACL \[19\]. The `NodeShape` class defines the properties of a class and specifies whether a class may contain additional properties (additional-Properties) of a specific type. Additionally, it represents operations related to a class (supportedOperation). Class `PropertyShape` represents the properties of a class, their data, and restrictions (e.g., a maximum value for a numeric property) and indicates whether the supported property is required or read-only.

OpenAPI parameters are represented as separate classes for every parameter type in the OpenAPI ontology. The `Header` class contains all the definitions of header parameters that are used in HTTP requests and responses. The `Cookie` class defines the cookies that are sent through HTTP requests and responses. The `Parameter` class defines all parameters that are attached to the operation’s URL. The class is further organized in the `PathParameter` and the `Query` classes that refer to the corresponding path and query parameters of the specification.

A request or response body (defined using the content property) is used to send and receive data via the REST API respectively (a response also includes a response code, e.g., 200, 400, etc.). `MediaType` is a representation format of request or response body data in different formats - the most common are JSON, XML, text, and images. They are typically defined in the `Paths` object; however, reusable bodies can also be defined in the `Components` object which holds a set of reusable objects (e.g., responses, parameters, schemas, request bodies). Each media type includes a `Schema` property, defining the data type of the message body. Request and response bodies are represented as properties of the `Operation` class. In

\[1\]https://www.intelligence.tuc.gr/semantic-open-api/
particular, request and response bodies are defined as classes, and their media type is also defined in this way. The Encoding class defines keywords denoting serialization rules for media types with primitive properties (e.g. contentType for nested arrays or JSON).

IV. Comparison

This work compares the two approaches (i.e. the W3C approach with the OpenAPI approach) in terms of completeness (i.e. their capability for describing Things and their related concepts). There is no direct correlation between the OpenAPI ontology and the ontologies used by the Thing Description approach of W3C. The OpenAPI Thing description approach benefits from semantic annotations proposed by the Semantic OpenAPI to describeWoT concepts along with other concepts such as identifier, name, description, etc. It maps these concepts to terms in the SOSA ontology (which is part of the SSN ontology) and also, in schema.org. The RDF triples in the translated OpenAPI ontology (i.e. resulting from a Thing description) may include URLs from these semantic models. These concepts are also described in the resulting ontology.

The comparison of the two approaches is based on the following criteria: a) the support of WoT concepts, b) the support of security concepts, c) the hypermedia controls support, d) the support for the descriptions of events and subscriptions, e) the protocol support and f) the data schemas support. Table I summarizes the results of this comparison.

A. WoT concepts support

The TD approach of W3C proposes the W3C TD Ontology for WoT concepts (e.g. Thing, interaction affordance, property affordance, action affordance, operation type). It also defines specific object properties, datatype properties and named individuals. This ontology also presents some alignments with the SOSA ontology and the schema.org vocabulary. The OpenAPI ontology does not represent WoT concepts (e.g. Thing, properties, actions) as it was meant to describe API terms, according to the OpenAPI specification. However, the OpenAPI Thing description approach benefits from semantic annotations (i.e. in OpenAPI Thing descriptions) to semantic models such as the SOSA ontology and the schema.org vocabulary that describe these concepts.

B. Security support

Both approaches are capable of describing the security mechanisms used for authentication and authorization of clients that interact with Things. The TD approach of W3C proposes the WoT Security Ontology for defining security schemes such as the API Key security scheme and the Basic Authentication security scheme. This ontology defines classes that represent security schemes and specific object properties (e.g. authorization) and datatype properties (e.g. flow, scopes). Similarly, the OpenAPI Thing description approach benefits from the OpenAPI ontology and the Security class, in particular, to describe the API security definitions used in OpenAPI Thing descriptions. The Security class of the ontology includes security schemes as subclasses (as shown in [10], [11]). Similar to the JSON description of the two approaches, the ontologies describe the same security schemes, except for the PSK security scheme which is only described in the WoT Security Ontology, and the OpenID Connect security scheme which is only described in the OpenAPI ontology. The combination of multiple security schemes (i.e. Combo) and the use of no security schemes at all (i.e. NoSecurity) in the API exposed by a Thing are also supported in both approaches.

C. Hypermedia Controls support

The TD approach of W3C proposes the Hypermedia Controls ontology for defining hypermedia controls, as defined by the WoT Architecture; Web Links and Web Forms. This ontology defines classes to represent a link, a form, an expected response and an additional expected response, and specific object properties (e.g. hasTarget) and datatype properties (e.g. forSubProtocol). This ontology also presents some alignments with the Hydra core vocabulary.

The OpenAPI ontology does not (yet) represent hypermedia concepts as described in OpenAPI (i.e. links). There is no direct correlation between the links described in the Hypermedia Controls ontology and the links described in the OpenAPI approach. For example, links in the W3C approach can be used in TDs to express any relations of Things with additional resources which may also be retrieved. Concerning forms, the OpenAPI ontology is based on Hydra and it represents service operations, which, according to the Hypermedia Controls ontology specification, are similar to TD forms. Forms represent how the client can perform specific requests to perform with Things. The class Form of the TD ontology could be related to the openApi:Operation class of the OpenAPI ontology, but this would not be absolutely correct (i.e. the two concepts are not equivalent). Finally, links (along with callbacks) might be represented in the ontology in the near future; it is a work in progress by the authors of the OpenAPI ontology.

D. Events and subscriptions support

The TD approach of W3C proposes the TD ontology for describing event affordances and subscriptions to events. These are described using Webhooks and Callbacks in OpenAPI. However, there is not (yet) support for Links, Webhooks and Callbacks in the OpenAPI ontology. In OpenAPI, Links are defined in the service response section to allow values returned by a service call to be
TABLE I: Comparison of W3C and OpenAPI ontology implementations.

<table>
<thead>
<tr>
<th>Comparison Criteria</th>
<th>W3C TD approach</th>
<th>OpenAPI TD approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>WoT Concepts support</td>
<td>Uses the TD Ontology.</td>
<td>Uses semantic annotations to SOSA, SSN, schema.org etc. It describes API concepts.</td>
</tr>
<tr>
<td>Protocol support</td>
<td>Also additional protocols (not limited to HTTP and HTTPS). Extra protocols using separate documents (e.g. a protocol vocabulary).</td>
<td>APIs that use the HTTP(S) protocol.</td>
</tr>
<tr>
<td>Hypermedia Controls support</td>
<td>Uses the Hypermedia Controls Ontology for Links and Forms.</td>
<td>The OpenAPI ontology does not (yet) support hypermedia links as defined by the OpenAPI Specification. OpenAPI ontology operations are similar to TD Forms.</td>
</tr>
<tr>
<td>Events and Subscriptions support</td>
<td>The TD Ontology describes Event affordances, subscriptions to events and Callbacks.</td>
<td>The concepts of events, subscriptions and OpenAPI Callbacks are not (yet) supported in the OpenAPI Ontology. Subscription operations to events are described using the Operation class.</td>
</tr>
<tr>
<td>Data Schemas support</td>
<td>Uses Data Schema Vocabulary.</td>
<td>Uses SHACL.</td>
</tr>
</tbody>
</table>

used as input for the next call. This is an attempt to incorporate HATEOAS functionality into the specification. Callbacks is a feature for defining asynchronous APIs or Webhooks. Callbacks define the requests that the described service will send to another service in response to certain events. The concept of subscriptions is not defined in the OpenAPI ontology, as it is not an OpenAPI specification feature (i.e. it is an API feature). However, subscription operations (i.e. to create, retrieve or delete subscriptions), are supported in OpenAPI Thing descriptions using the Operation object and they can be supported in the OpenAPI ontology as well using the Operation class.

E. Protocols support

The OpenAPI ontology can only describe APIs implemented using the HTTP(S) protocol, while the TD approach is meant to support additional protocols. W3C notes that the number of Protocol Bindings that can be implemented by a Thing is not restricted. Other protocol bindings (e.g. for CoAP, MQTT, or OPC UA) can be described in separate documents (e.g. a protocol vocabulary).

F. Data schemas support

Both approaches use ontologies to describe the data schemas used for the representation of Things and their interactions (e.g. the input and output data schemas for actions). The TD approach proposes an RDF vocabulary (i.e. the Data Schema vocabulary described for JSON data schema definitions). For the OpenAPI ontology, the corresponding mechanism is SHACL.

V. Conclusions

This work suggests that OpenAPI descriptions of Web Things can be mapped to ontologies. The approach is similar (but not identical) to that of the WoT Thing Description of W3C and ontologies respectively. Both approaches are still in progress. Both contribute to the vision of the Semantic Web of Things concept that aspires to unify the world of interconnected Things so that Things can be handled using Semantic Web tools. The advantage of the OpenAPI approach over WoT Thing Description is the uniformity of the representation for both Things and services that co-exist in an application. We are currently extending the OpenAPI ontology to support representation for Links and Callbacks. Future work will also focus on Thing discovery and re-use in composite applications with many Things.

REFERENCES


