Bridging the Gap to the Web of Things

On the Conversion between WoT Data Models and the Semantic Definition Format

Jan Romann[1]

[1]University of Applied Sciences Emden/Leer, Constantiaplatz 4, 26723 Emden, Germany

Abstract

In this paper, we present our work on the conversion between Web of Things (WoT) data models (Thing Descriptions and Thing Models) and the Semantic Definition Format (SDF) that aims at providing a bridge between different data modeling ecosystems for the Internet of Things. We present a comprehensive mapping of the current 1.1 version of the WoT TD specification to SDF and discuss possible improvements that could be integrated into the standardization process of both specifications. Our results are illustrated by a conversion tool written in Python that can be integrated into many (less constrained) devices and environments, such as gateways or code generation tools, and can also serve as a reference implementation for other programming languages.

Keywords

Web of Things, Semantic Definition Format, Interoperability

1. Introduction

Semantic descriptions of devices in the Internet of Things (IoT) are an important building block for enabling both machine-to-machine and human-to-machine interaction: IoT devices need indications for the proper protocols, data formats, and security mechanisms to use, while human-readable meta-data is essential for rendering information in user interfaces. As of today, the IoT faces two kinds of interoperability problems related to semantic descriptions: (a) incompatible data models and interaction patterns at the device or *instance* level, and (b) the lack of a common data and interaction model at the ecosystem level, which makes it difficult to universally describe *classes* of devices.

There are two open standards that promise solutions for these problems: On the one hand, the W3C Web of Things (WoT) and its Thing Description (TD) [1] try to adapt principles from the World Wide Web (WWW) to the IoT. The TD, a JSON-LD [2] document with a specialized vocabulary, serves as the fundamental building block of the WoT architecture and is supposed to be the entry point of a Thing, making it easier for its peers and other *consumers* to interact with it. A different approach is pursued by the One Data Model liaison organization (OneDM) together with the Internet Engineering Task Force (IETF): The so-called Semantic Definition Format (SDF) [3] strives to be a universal format for converting data models between different

SWoCoT 2023: First International Workshop on Semantic Web on Constrained Things at ESWC 2023, May 28, 2023, Hersonissos, Crete, Greece

jan.romann@hs-emden-leer.de (J. Romann)

0000-0002-9021-3821 (J. Romann)

CEUR Workshop Proceedings (CEUR-WS.org)

ecosystems, with the goal of being an intermediary between them and thus solving the second aspect of the interoperability problem.

While bearing structural similarities—both approaches use properties, actions, and events to describe the *interaction affordances* of a Thing, and borrow vocabulary from JSON Schema [4]—there are also important differences between the two specifications, such as the reliance on plain JSON (and not JSON-LD) in the case of SDF models and thus a fundamentally different approach to semantically describing IoT devices. At the same time, new features such as the recently introduced WoT Thing Model (TM)—which also serves the purpose of providing a reusable format for device *classes*—indicate a convergence of the two specifications to a certain extent. Both specifications also have different philosophies when it comes to describing a *taxonomy* of devices or individual device components: where WoT relies on a linking approach between TDs or TMs, SDF also allows for the creation of nested components within a single model.

In the past, structural differences like these have prevented a full conversion between the two formats. Moreover, there is not yet a formal specification of how to map WoT data models to SDF and vice versa. Since both formats are dealing with two important sides of the interoperability problem in the IoT, there is a strong motivation to develop (1) a mapping between WoT data models and SDF, and (2) corresponding conversion tools. As SDF strives to be a universal format for converting between different information models, creating such a tool set would potentially pave the way for the integration of any ecosystem supported by SDF into the Web of Things.

In this paper, we present the results we have obtained in our approach for a systematic conversion between WoT data models and SDF. This includes a flexible converter written in Python, which is usable both as a library, a command line tool, and a simple web application. Based on our results, we discuss the current limitations of the two specifications as well as possible improvements. Furthermore, to explore the potential of a combination of the two formats for bridging the gap between constrained environments and their corresponding ecosystems—where device capabilities might only be described by a class identifier, as is the case for IPSO Smart Objects—, and the Web of Things, where TDs are the common way of describing a Thing. Lastly, we will discuss if WoT and SDF in general, and our mapping approach in particular can bridge the gap between the Web of Things and constrained devices or if a fundamentally different approach is required to achieve this goal.

The rest of this paper is structured as follows: After introducing relevant foundations in section 2 and defining the requirements for our endeavor in section 3, we will present our work on mappings between the two formats in section 4 and outlining its design and implementation in section 5. Lastly, in section 6, we will draw a conclusion and discuss possible limitations and future work.

2. Foundations

In this section, we will give a brief overview of normative documents (subsection 2.1) and other literature (subsection 2.2) relevant to this paper.

2.1. Relevant Specifications and Standards

While both WoT and SDF have developed into families of specifications, the two approaches are in different stages of development: The Web of Things Architecture [5] and the Thing Description [6] specification have both reached the status of W3C Recommendations, with updated 1.1 versions of both Web Standards [1, 7] and additional specifications for Discovery [8] and a Profile [9] mechanism on the way. The specification body also defines a mechanism [10] for binding interaction affordances to resources accessible via application-layer protocols, such as HTTP [11] or CoAP [12]. However, this binding mechanism is currently still underdeveloped and will be fleshed out during the Working Group's next charter period.

The updated TD specification also introduces the new concept of Thing Models (TMs), which share most of their vocabulary with TDs, but are less constrained when it comes to mandatory fields while providing additional mechanisms for referencing and extending existing (external) definitions. These features make them compatible with SDF and designate them as intermediaries between TDs and SDF models, which we will see later in more detail.

SDF on the other hand has not reached the status of an RFC yet at the time of writing but is about to be finished by its IETF working group. Besides the SDF core specifications, there are a number of additional drafts being worked on that are dealing with extensions to SDF, e.g., by adding support for modeling complex relations [13], or for so-called mapping files [14] that can be used for augmenting an SDF model and are very important for this paper.

Examples of a TD (Listing 1) and a corresponding TM, (Listing 2) as well as an SDF model (Listing 3) and an SDF mapping file (Listing 4) can be found in the appendix. These examples also illustrate our conversion process for simple, i.e. non-nested data models.

2.2. Related Work

There are a number of pre-existing projects which also deal with the conversion between SDF and other formats. The one most closely related to this paper is a converter by Roman Kravtsov[1] written in JavaScript (for the Node.js ecosystem), which is able to convert SDF models to WoT Thing Models. Kravtsov dealt with the conversion between the two formats in the context of his master's thesis [15], comparing, besides SDF, multiple formats for semantic descriptions of IoT devices with the newly added Thing Model feature. Besides his SDF converter, he also provides implementations for Oracle Device Models[2], Eclipse Vorto models[3], and Microsoft's Digital Twin Definition Language (DTDL)[4].

While working in general, Kravtsov's converter between SDF and WoT has a number of limitations: It does not support the backward conversion of WoT documents to SDF, can only convert SDF models to TM (there is no support for TD), and only accepts a single `sdfObject` within a model as an input. This also means that Kravtsov's converter does not support roundtripping, i.e., translating a conversion result back into its original format. Furthermore, there is no validation of both inputs and outputs, while the actual mapping of SDF affordances is

[1] https://github.com/roman-kravtsov/sdf-object-converter (retrieved: May 10, 2023).
[2] https://github.com/roman-kravtsov/oracle-device-model-converter (retrieved: May 10, 2023).
[3] https://github.com/roman-kravtsov/vorto-model-converter (retrieved: May 10, 2023).
[4] https://github.com/roman-kravtsov/digital-twins-converter (retrieved: May 10, 2023).

a simple copy operation, potentially resulting in WoT TMs containing definitions only specified for SDF. Besides these limitations for the actual conversion, the converter has a number of usability issues, neither providing a library API nor a command line interface that allows for specifying input and output file names.

Another relevant SDF converter[5] covers the conversion between SDF and the data modelling language YANG (Yet Another Next Generation, [16]). Written by Jana Kiesewalter in C++, her converter can be used as a standalone command line application, which also allows integration into web applications.[6] Her converter resembles a more sophisticated approach to SDF conversion than Kravtsov's, also supporting bidirectional conversion and roundtripping. Kiesewalter's converter, which is part of her Master's thesis [17], also features a comprehensive mapping between the two data modeling approaches, which she also codified in an Internet-Draft [18]. However, there are a number of small issues with the converter which are related to the flexible nature of the JSON Schema-inspired vocabulary and the implementation of its validation, the latter of which has a negative impact on the converter's performance since the JSON Schema document currently needs to be read in from the file system for every validation process.

Furthermore, there are two converters by Ericsson Research: The first one[7] allows the conversion between SDF and IPSO data models, which is the data modeling approach of OMA SpecWorks. The second converter[8] also allows for a conversion between SDF and DTDL, which is used for the company's Azure Digital Twins models and is, like WoT TD, based on JSON-LD. Finally, OCF provides tools[9] to convert between SDF and OpenAPI, a popular format for describing web APIs.

Through a growing number of converter implementations, we can observe an integration between different ecosystems and data modeling approaches fostered by SDF as a common description framework. However, both comprehensive support for a conversion of the WoT document formats and the inclusion of instance-specific information had been underdeveloped so far.

3. Requirements

In this section, we will describe the requirements we defined both for our mappings between WoT and SDF, and our converter implementation.

Our most important requirement was to provide a comprehensive mapping between SDF and WoT TD. That means that our converter should be able to translate every field that is contained in an SDF model or a WoT definition into an equivalent definition in the other format. This should also include definitions that are either not defined in one of the two specifications or originate from vocabulary extensions. In the case of SDF, these additional definitions should be included in mapping files.

[5] https://github.com/jkiesewalter/sdf-yang-converter (retrieved: May 10, 2023).
[6] See http://sdf-yang-converter.org/ (retrieved: May 10, 2023).
[7] https://github.com/EricssonResearch/ipso-odm (retrieved: May 10, 2023).
[8] Although the converter is accessible through a web interface hosted by Ericsson http://wishi.nomadiclab.com/sdf-converter/ (retrieved: May 10, 2023), alongside a number of other SDF converters, its source code and/or project description seems unavailable to the public at the moment.
[9] https://github.com/openconnectivityfoundation/SDFtooling (retrieved: May 10, 2023).

An important criterion for evaluating both the comprehensiveness and the accuracy of a mapping is the ability to roundtrip the conversion process, i.e., to convert a document from either of the two specifications into the other one and receive the same document when applying a conversion in the other direction. The roundtrip potential should therefore be considered both during the development of the mappings themselves and as additional test cases for the actual converter implementation, ensuring that as many mappings as possible are reversible. Besides the conversions between SDF and WoT, the converter should also support the conversion of WoT TMs into WoT TDs, and vice versa. This way, users are able to first create reusable WoT documents from SDF models, which they can then instantiate on demand. Conversely, they can create generalizations in the form of TMs from TDs, although the resulting TMs will contain instance-specific information that might have to be removed manually by the user after the conversion process.

Both specifications have mechanisms for referencing and (in the case of WoT TMs) extending models, which should also be able to be resolved in order to create consolidated documents before conversion, as documents conforming to the other specification cannot be directly referenced. However, this might be at odds with the roundtripping mentioned above, as resolved references cannot be converted back into a reference without making explicit which definitions originated from another document. Moreover, the converter should be able to validate both conversion inputs and outputs on the basis of the pre-defined schemas (using either CDDL [19] or JSON Schema [20]) that are included in both specifications.

Our second most important requirement was that the converter should be usable in as many deployment scenarios as possible. Similar to Kiesewalter's converter, our implementation should be usable both as a CLI tool and as a web application. The CLI tool should enable users to both call the converter from a terminal and integrate it in scripts, while the web application should offer a simple interface for converting between SDF and WoT. However, the converter should also expose its conversion logic as a library for the ecosystem/programming language of our choice, making it possible to reuse it in other implementations of the same or a neighboring ecosystem, such as code generators.

Moreover, the converter as a whole should work in a wide variety of environments, supporting as many operating systems (e.g., Linux, macOS, and Windows) and types of devices as possible. This should include single-board computers like Raspberry Pis, but not necessarily constrained devices, as JSON—the serialization format of both SDF and WoT—is not very well suited for constrained environments.

Lastly, the converter should provide a high degree of usability and should be easy to install and add to third-party projects, e.g., by being included in a package repository associated with the programming language/ecosystem chosen.

4. Mappings between SDF and WoT

In this section, we will briefly outline our mapping between the WoT data and interaction models on the one hand and SDF on the other hand. A high-level view of our mapping process can be seen in Figure 1.

At the center of our mapping between WoT and SDF are WoT Thing Models, which proved to

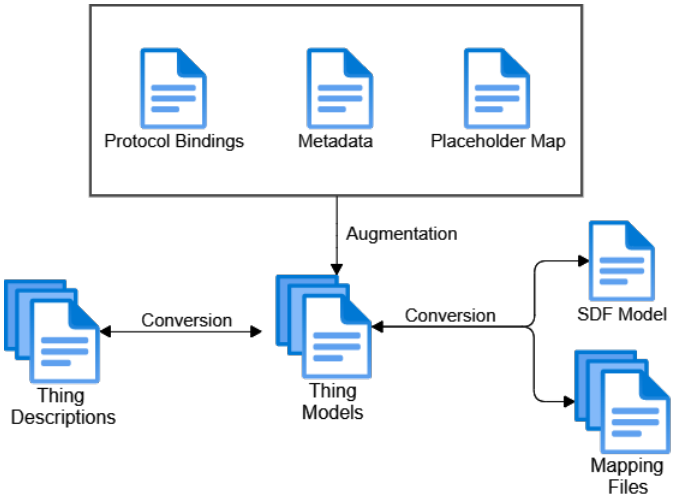

Figure 1: High-level view on the conversion process for WoT TMs, WoT TDs, and SDF models and mapping files.

be versatile enough to become an intermediary between the instance-specific Thing Descriptions and the abstract SDF models, the latter of which might also be accompanied (and potentially augmented) by additional mapping files providing instance- or ecosystem-specific information. Overviews of our mappings of the most important keywords from both specifications can be seen in Table 1 (mapping from SDF to WoT) and Table 2 (mapping from WoT to SDF).

The central role of Thing Models in our mapping process arises from their ability to, similar to SDF models, describe interaction affordances in a protocol-agnostic way. In contrast to TDs, they do not need to include protocol bindings (in hypermedia forms) and can also omit the otherwise mandatory security definitions. Furthermore, they can also reproduce and preserve SDF's import mechanism (which uses the `sdfRef` keyword) when it comes to same-document references.

However, it is therefore not only the abstract nature of WoT Thing Models that makes it possible to convert SDF models almost directly to them but also the similarities of both formats' vocabulary: For instance, an SDF field like `sdfOutputData` becomes `output` when converted to WoT, while also allowing for simply copying over many of the vocabulary terms inspired by JSON Schema. However, since not all SDF vocabulary terms have a (direct) WoT equivalent, we have to differentiate two cases if direct mapping is not possible.

In the first case, we can map the field to a helper field defined in addition to the regular WoT vocabulary. One example of this approach is the `copyright` field from the SDF info block, which does not have a direct equivalent in the WoT information model. Therefore, we introduce a prefixed field named `sdf:copyright`, which can be used to integrate copyright information into a WoT data model and enable roundtripping, since unmapped information would otherwise be lost when reversing the conversion process. It is noticeable, though, that the info block in SDF is describing the whole model and therefore all of its components, i.e. every `sdfObject` and `sdfThing`. Since every `sdfObject` and `sdfThing` becomes its own TD or TM, this information

Table 1
Overview of mappings of the most important SDF keywords to WoT.

SDF Keyword	WoT Class/Keyword
`sdfThing`	TM with `tm:submodel` links
`sdfObject`	TM without `tm:submodel` links
`sdfProperty`	`PropertyAffordance`
`writable`	`readOnly` (negated)
`readable`	`writeOnly` (negated)
`sdfAction`	`ActionAffordance`
`sdfOutputData`	`output`
`sdfInputData`	`input`
`sdfEvent`	`EventAffordance`
`sdfOutputData`	`output`
`sdfData`	`schemaDefinitions` (at the TM level)
`sdfRef`	`tm:ref`
`sdfChoice`	Enum of JSON objects with `sdf:choiceName`
`sdfRequired`	`tm:optional` (by including all non-required interaction affordance keys)
`namespaces`	`@context`
`defaultNamespace`	`sdf:defaultNamespace`
`info`	`Multiple targets:`
`version`	`model` field in Version class
`title`	`sdf:title`
`copyright`	`sdf:copyright`
`license`	If URL: link with relation-type `license` Else: `sdf:license`

needs to be included in every resulting JSON-LD document in order to prevent information loss.

In the second case, we can map SDF definitions without a direct counterpart to a semantically equivalent construction in WoT. One example of this approach is the `license` field from the SDF information block, given that its value is a URL pointing to a license text. In this case, the license is included as an entry in the `links` field, specifying a `license` link relation type. However, if the license is not a URL, the first strategy (using an `sdf:license` field) needs to be applied. Another example for this second case is the mapping of `sdfChoice`, an enumeration where data schemas are associated with a name. Since WoT only has `enums` as a semantically similar construct, we currently use them as a conversion target, adding an additional `sdf:choiceName` member to each enum entry to enable roundtripping. We decided to use this approach since WoT is currently missing an equivalent to `sdfChoice`, which is semantically similar to JSON Schema's `anyOf`. However, `anyOf` is currently not part of the vocabulary WoT borrows from JSON Schema, but is considered for the next major version of the TD specification. This is one example where the alignment of SDF and WoT is not yet complete, and where the WoT data models need to become more expressive to fully cover SDF's semantics.

After a Thing Model has been created from SDF inputs, which might include mapping files that add missing information needed by TDs, it can be transformed into a Thing Description using a well-defined algorithm from the WoT TD specification [1, section 9.4]. In the other direction,

Table 2
Overview of mappings of the most important WoT classes and keywords [1, section 5] to SDF.

WoT Class/Keyword	SDF Keyword
`Thing`	`sdfThing` (TM has `tm:submodel` links), `sdfObject`
`title`	`label`
`description`	`description`
`schemaDefinitions`	`sdfData`
`@context`	namespaces of the SDF model (with exceptions)
`DataSchema`	`dataqualities`
`readOnly`	Mapping file
`writeOnly`	Mapping file
`InteractionAffordance`[a]	—
`title`	`label`
`description`	`description`
`PropertyAffordance`	`sdfProperty`
`readOnly`	`writable` (negated)
`writeOnly`	`readable` (negated)
`observable`	`observable`
`ActionAffordance`	`sdfAction`
`input`	`sdfInputData`
`output`	`sdfOutputData`
`EventAffordance`	`sdfEvent`
`tm:ref`	`sdfRef`
`tm:optional`	`sdfRequired` (by including all non-optional interaction affordance keys)
`Link`	Mapping file, except for special link types (e.g., `license`, `tm:extends`, `tm:submodel`)

[a] This is the base class of the three affordance types.

using TDs as an input, TMs can be seen as near supersets of TDs, making it possible to convert a single TD with little changes into a TM since—in contrast to SDF models—they are capable of also containing instance-specific information such as protocol bindings. When converting a Thing Model containing instance-specific information to SDF, the unmappable fields are put into an SDF mapping file, which is another JSON document where the keys of the conversion targets represent JSON Pointers [21] (such as `#/sdfObject/bar/sdfProperty/bar/security`) that reference the place within the corresponding SDF model that is supposed to be augmented by the mapping file definition. This approach also enables roundtripping in the other direction, making it possible to store information such as the original JSON-LD `@context` that cannot be recreated from SDF's namespace concept alone.

A crucial difference between SDF and WoT that became apparent during the development of the mappings is how the two specifications approach hierarchical or nested models that describe complex devices (such as a combination of a freezer and a refrigerator in a single apparatus that is capable of controlling both of its components). SDF allows for expressing hierarchical relationships via its `sdfThing` class which itself can contain an arbitrary number of `sdfThings` and `sdfObjects`, the latter of which can be seen as "leaf nodes" within an SDF model. WoT on

the other hand strongly prefers the use of web links that point to subordinate TMs or TDs for describing hierarchical relationships between Things (using either the `tm:submodel` or `item` link relation type). In the process of creating our mappings, we identified needs for alignment in both specifications, which led to an adjustment of the role `sdfThing`s play within an SDF model.[10] We also saw the need to specify a concept in WoT that allows for containing multiple TMs or TDs in a single document, as, otherwise, it becomes very difficult to work with nested conversion results due to the need to point to submodels or subthings via a URI. In our mapping, we introduced a workaround called TD/TM *Collections*, which are JSON objects where each value represents a TD or TM, while link URIs pointing to subthings or submodels within the same Collection are JSON Pointers (see Listing 5 for a TM Collection example). This approach showed potential for achieving a full mapping between hierarchical SDF and WoT data models, enabling the conversion of nested relationships between the two formats (see Listing 6 for an SDF model created from the TM collection). However, we also experienced some difficulties when covering edge cases, such as TM/TD Collections that contain TMs or TDs that point to documents outside the Collection, which could potentially lead to naming conflicts within the TM/TD Collection during the reference resolution. Therefore, we arrived at the conclusion that there is still more specification work needed to be able to reliably describe complex devices via a single WoT document and to enable the conversion of nested descriptions from and to other formats.

5. Converter Design and Implementation

We designed our converter to be as flexible as possible and usable in a wide variety of deployment scenarios. Therefore, we located the core conversion logic into a reusable library component, which we built upon to create both a command-line interface (CLI) and a web application.

Due to the fact that we could use Thing Models as a mediator during the conversion process, the core logic only needed to cover the conversion between TDs and TMs on one hand, and between TMs and SDF on the other. The conversion between TDs and SDF models could then be derived by chaining the two existing types of conversion. In total, our library component exposes six functions (named, e.g., `convert_sdf_to_wot_td` or `convert_wot_td_to_wot_tm`) as its external API, offering a number of additional parameters to be included to enable additional features or, e.g., pass additional augmenting files alongside an SDF model or a TM. The available parameters for the six exposed functions are also illustrated by the CLI API, which can be seen in Table 3.

Our third component, a web application, is designed to demonstrate the converter's capabilities in an easily accessible manner and encourage experimentation by offering a basic graphical user interface (GUI) as well as a simple REST API. A screenshot illustrating the web application's composition and functionality can be seen in Figure 2. Users are able to insert models that are supposed to be converted in one of the two text areas and then select the appropriate source

[10] In an earlier version of SDF, `sdfThing`s were not allowed to contain interaction affordances as `sdfObject`s can. This made it very difficult to map a Thing Description or Model that both contains an interaction affordance and a link to a subthing or submodel to SDF. However, in recent versions of the SDF specification, this problem has been addressed and is no longer present.

Table 3
Overview of the available parameters for the sub-commands of our CLI.

Parameter	Arguments	Sub-Commands	Default
`--input`, `-i`*	File path(s) or URL(s)[a]	all	—
`--output`, `-o`	File path	all	—
`--suppress-roundtripping`	—	all	False
`--indent`	Natural number	all	4
`--origin-url`	URL	sdf-to-tm, sdf-to-td	—
`--mapping-files`	Zero or more file paths	sdf-to-tm, sdf-to-td	—
`--title`	String	sdf-to-tm, sdf-to-td	—
`--version`	String	sdf-to-tm, sdf-to-td	—
`--copyright`	String	sdf-to-tm, sdf-to-td	—
`--license`	String	sdf-to-tm, sdf-to-td	—
`--meta-data`	File path or URL	tm-to-sdf, tm-to-td	—
`--bindings`	File path or URL	tm-to-sdf, tm-to-td	—
`--placeholder-map`	File path or URL	tm-to-sdf, tm-to-td	—
`--mapping-file-output`	File path	tm-to-sdf, td-to-sdf	—
`--remove-not-required-affordances`	—	tm-to-td	False

* Mandatory Parameter.
[a] Can be multiple paths/URLs when converting from WoT TM/TD – the imported TMs/TDs are then treated as Collections.

and target formats via a dropdown menu. Helper functionality (e.g., clearing or formatting the inputs) and options can be enabled at the bottom of the GUI.

The actual implementation was done in Python, which offered not only a rich package ecosystem we could build upon, especially for handling command line arguments (via the argparse module included in the standard library) and for building our web application (using the framework Flask); we were also able to reuse libraries for handling and applying JSON Schema [4], JSON Pointers [21], and the JSON Merge Patch algorithm [22], making it possible to perform input and output validation as well as resolving the different reference and extension mechanisms WoT and SDF define.

The converter library and CLI tool can be installed as a bundle via the Python package repository PyPI[11], while their source code is publicly available on GitHub[12]. After the installation, users can either use the converter CLI from a terminal of their choice (by using the command `sdf-wot-converter`) or depend on the library in their own Python project.

The web application is continuously running on a web server reachable under https://sdfwotconverter.pythonanywhere.com/ (retrieved: May 10, 2023). Its source code is also publicly available on GitHub[13] and can be used to run the web application locally. As mentioned in section 2.2, our converter implementation is also featured in a collection of SDF conversion

[11] https://pypi.org/project/sdf-wot-converter/ (retrieved: May 10, 2023).
[12] https://github.com/JKRhb/sdf-wot-converter-py (retrieved: May 10, 2023).
[13] https://github.com/JKRhb/sdf-wot-converter-py-demo (retrieved: May 10, 2023).

Figure 2: Screenshot of our converter's web interface.

tools[14] that illustrates how SDF models can be translated into different target formats, allowing for a direct comparison via a web interface.

6. Conclusion and Future Work

In this paper, we dealt with the conversion between the Semantic Definition Format [3] and the Web of Things Thing Description (WoT TD) [1]—two specifications that aim at improving the interoperability in the Internet of Things in terms of data models and device interactions, respectively. On the basis of a detailed mapping between the data models of both specifications, we developed a converter written in Python which can be used as a library and a command line interface (CLI). Building upon our library, we also created a web application, which not only allows for easy access to the converter's logic via a graphical user interface but also via a simple REST API.

Corresponding with our mappings, our converter supports both WoT TDs and TMs (Thing Models), the latter being a new addition that allows for describing not only *instances*, but also *classes* of devices, enabling the re-use of definitions via import and extension features. For our mappings, we discovered that TMs can act as an intermediary for the conversion between SDF and Thing Descriptions, allowing us to only define two types of mappings (TM and SDF on the one hand and TM and TD on the other) in order to cover all three data formats. Besides SDF models, we use the newly defined concept of SDF mapping files [14], allowing us to consider instance- and ecosystem-specific information during the conversion process. This aspect closes

[14] http://wishi.nomadiclab.com/sdf-converter/ (retrieved: May 10, 2023).

an important gap between the two specifications, making it possible to convert from SDF to WoT Thing Descriptions, provided that a mapping file is present that contains the necessary mandatory protocol bindings and security definitions.

In a real-world deployment, however, the requirement of such a mapping file may become a problem: For example, an intermediary that is performing conversions from an ecosystem-specific format to WoT TD, using SDF as a translation medium, needs to somehow provide the instance-specific information that is required to create a valid TD. This information must either be derived from the input format or from the context of the conversion (e.g., by using the Thing's IP address to fill the TD's `base` field). However, this indicates that SDF is not yet as universal as it needs to be to fully bridge the gap between WoT and other information models, since the accompanying mapping files still include ecosystem-specific terms most of the time which are not necessarily compatible with each other. Therefore, more research and specification work is required to further develop the concept of mapping files and to make SDF truly viable for data models that go beyond abstract descriptions of device classes.

In WoT, we identified the need for being able to include multiple TMs or TDs in a single document in order to be able to work with nested data structures, as SDF allows for describing a hierarchy of Things in a single model, while the WoT TD specification does not, as the preferred way for creating a hierarchy is by linking between documents. By introducing an experimental concept called TM/TD Collections, we have been able to recreate a nested structure using JSON Pointers [21] with same-document references. These new concepts also enabled us to support roundtripping—i.e., the reproduction of original inputs from a conversion result—in most cases, with references to external documents (which we also resolve before the conversion) being the greatest exception in this regard.

While we have been able to close a number of gaps between the two specifications and were able to make a few contributions along this way, there is still room for improvement. In particular, there are parts of the WoT vocabulary which we currently need to map to a mapping file that could potentially also be converted to SDF model vocabulary. The most prominent example for this at the moment is probably the addition of SDF relations [13], which could be used for mapping generic WoT links to SDF. Other additions could involve security definitions or more concrete protocol-specific vocabulary. Furthermore, we noticed that the concept of SDF namespaces could be refined in order to make the relationship to JSON-LD features used in WoT TD clearer. Therefore, the proposed mappings in this paper are still only a starting point for a comprehensive alignment of the two specifications and should also be codified in a specification or standard as soon as both WoT TD is published as version 1.1 and SDF reaches RFC status.

Future work also needs to better evaluate and compare this and future implementations based on quantitative and qualitative criteria. Moreover, a better grounding of our research in the theoretical literature is required to provide not only a better foundation for the mapping between the two formats but also for enabling a more sophisticated approach to the implementation of these mappings.

Lastly, for future versions of our converter, we consider choosing a compiled, statically typed language like Rust for the implementation instead, potentially improving its performance and making it more robust, while also making it possible to use the converter in environments where a Python interpreter is not available. This would potentially also contribute to further

bridging the gap between constrained environments and the Web of Things.

Acknowledgments

This research has been partially funded by the German Bundesministerium für Wirtschaft und Klimaschutz (BMWK), Förderkennzeichen 01MC22008C. We would like to thank Carsten Bormann and Olaf Bergmann for their valuable input and continuous advice, and also thank the reviewers of the SWoCoT 2023 Workshop for their very helpful and constructive feedback.

References

[1] S. Käbisch, T. Kamiya, M. McCool, V. Charpenay, Web of Things (WoT) Thing Description 1.1, W3C Candidate Recommendation Snapshot, W3C, 2023. URL: https://www.w3.org/TR/2023/CR-wot-thing-description11-20230119/.

[2] D. Longley, P.-A. Champin, G. Kellogg, JSON-LD 1.1, W3C Recommendation, W3C, 2020. URL: https://www.w3.org/TR/json-ld11/.

[3] M. Koster, C. Bormann, Semantic Definition Format (SDF) for Data and Interactions of Things, Internet-Draft draft-ietf-asdf-sdf-13, IETF, 2023. URL: https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-13, Work in Progress.

[4] A. Wright, H. Andrews, B. Hutton, JSON Schema Validation: A Vocabulary for Structural Validation of JSON, Internet-Draft draft-bhutton-json-schema-validation-01, IETF, 2020. URL: https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-validation-01, Work in Progress.

[5] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, K. Kajimoto, Web of Things (WoT) Architecture, W3C Recommendation, W3C, 2020. URL: https://www.w3.org/TR/2020/REC-wot-architecture-20200409/.

[6] S. Käbisch, T. Kamiya, M. McCool, V. Charpenay, M. Kovatsch, Web of Things (WoT) Thing Description, W3C Recommendation, W3C, 2020. URL: https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/.

[7] M. Lagally, R. Matsukura, M. McCool, K. Toumura, Web of Things (WoT) Architecture 1.1, W3C Candidate Recommendation Snapshot, W3C, 2023. URL: https://www.w3.org/TR/2023/CR-wot-architecture11-20230119/.

[8] A. Cimmino, M. McCool, F. Tavakolizadeh, K. Toumura, Web of Things (WoT) Discovery, W3C Candidate Recommendation Snapshot, W3C, 2023. URL: https://www.w3.org/TR/2023/CR-wot-discovery-20230119/.

[9] M. Lagally, B. Francis, M. McCool, R. Matsukura, S. Käbisch, T. Mizushima, Web of Things (WoT) Profile, W3C Working Draft, W3C, 2023. URL: https://www.w3.org/TR/2023/WD-wot-profile-20230118/.

[10] M. Koster, E. Korkan, Web of Things (WoT) Binding Templates, W3C Working Group Note, W3C, 2020. URL: https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/.

[11] R. T. Fielding, M. Nottingham, J. Reschke, HTTP Semantics, Technical Report 9110, IETF, 2022. doi:10.17487/RFC9110.

[12] Z. Shelby, K. Hartke, C. Bormann, The Constrained Application Protocol (CoAP), RFC 7252, IETF, 2014. doi:10.17487/RFC7252.

[13] P. Laari, Extended Relation Information for Semantic Definition Format (SDF), Internet-Draft draft-laari-asdf-relations-01, IETF, 2022. URL: https://datatracker.ietf.org/doc/html/draft-laari-asdf-relations-01, Work in Progress.

[14] C. Bormann, J. Romann, Semantic Definition Format (SDF): Mapping files, Internet-Draft draft-bormann-asdf-sdf-mapping-02, IETF, 2023. URL: https://datatracker.ietf.org/doc/html/draft-bormann-asdf-sdf-mapping-02, Work in Progress.

[15] R. Kravtsov, Thing Model for the Web of Things, Master's thesis, University of Passau, 2021. Unpublished.

[16] M. Bjorklund, The YANG 1.1 Data Modeling Language, RFC 7950, IETF, 2016. doi:10.17487/RFC7950.

[17] J. Kiesewalter, Design and Implementation of an SDF/YANG Converter in the Context of Standardization, Master's thesis, University of Bremen, 2021. Unpublished.

[18] J. Kiesewalter, C. Bormann, Mapping between YANG and SDF, Internet-Draft draft-kiesewalter-asdf-yang-sdf-01, IETF, 2021. URL: https://datatracker.ietf.org/doc/html/draft-kiesewalter-asdf-yang-sdf-01, Work in Progress.

[19] H. Birkholz, C. Vigano, C. Bormann, Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures, RFC 8610, IETF, 2019. doi:10.17487/RFC8610.

[20] A. Wright, H. Andrews, G. Luff, JSON Schema Validation: A Vocabulary for Structural Validation of JSON, Internet-Draft draft-handrews-json-schema-validation-01, IETF, 2018. URL: https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-validation-01, Work in Progress.

[21] P. C. Bryan, K. Zyp, M. Nottingham, JavaScript Object Notation (JSON) Pointer, RFC 6901, IETF, 2013. doi:10.17487/RFC6901.

[22] P. E. Hoffman, J. M. Snell, JSON Merge Patch, RFC 7396, IETF, 2014. doi:10.17487/RFC7396.

A. Appendix

```
{
    "@context": "https://www.w3.org/2022/wot/td/v1.1",
    "id": "urn:uuid:0804d572-cce8-422a-bb7c-4412fcd56f06",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in": "header"}
    },
    "security": "basic_sc",
    "properties": {
        "status": {
            "type": "string",
            "forms": [{"href": "https://mylamp.example.com/status"}]
        }
    },
    "actions": {
        "toggle": {
            "forms": [{"href": "https://mylamp.example.com/toggle"}]
        }
    },
    "events": {
        "overheating": {
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

Listing 1: Example WoT Thing Description [taken from 1, Example 1].

```
{
  "@context": "https://www.w3.org/2022/wot/td/v1.1",
  "@type": "tm:ThingModel",
  "id": "urn:uuid:0804d572-cce8-422a-bb7c-4412fcd56f06",
  "title": "MyLampThing",
  "securityDefinitions": {
    "basic_sc": {"scheme": "basic", "in": "header"}
  },
  "security": "basic_sc",
  "properties": {
    "status": {
      "type": "string",
      "forms": [{"href": "https://mylamp.example.com/status"}]
    }
  },
  "actions": {
    "toggle": {
      "forms": [{"href": "https://mylamp.example.com/toggle"}]}
  },
  "events": {
    "overheating": {
      "data": {"type": "string"},
      "forms": [{
          "href": "https://mylamp.example.com/oh",
          "subprotocol": "longpoll"
        }]
    }
  }
}
```

Listing 2: Example WoT TM, created from the TD in Listing 1. Note that, in contrast to the TD, the top-level `title`, `securityDefinitions` and `security` fields as well as the affordance-level `forms` could also be omitted here without making the TM invalid.

```
{
  "sdfObject": {
    "sdfObject0": {
      "label": "MyLampThing",
      "sdfProperty": {
        "status": {
          "observable": false,
          "type": "string"
        }
      },
      "sdfAction": {
        "toggle": {}
      },
      "sdfEvent": {
        "overheating": {
          "sdfOutputData": {
            "type": "string"
          }
        }
      }
    }
  }
}
```

Listing 3: Example SDF model, based on the Thing Model from Listing 2.

```
{
  "map": {
    "#/sdfObject/sdfObject0/sdfProperty/status": {
      "forms": [
        {
          "href": "https://mylamp.example.com/status"
        }
      ]
    },
    "#/sdfObject/sdfObject0/sdfAction/toggle": {
      "forms": [
        {
          "href": "https://mylamp.example.com/toggle"
        }
      ]
    },
    "#/sdfObject/sdfObject0/sdfEvent/overheating": {
      "forms": [
        {
          "href": "https://mylamp.example.com/oh",
          "subprotocol": "longpoll"
        }
      ]
    },
    "#/sdfObject/sdfObject0": {
      "@context": "https://www.w3.org/2022/wot/td/v1.1",
      "id": "urn:uuid:0804d572-cce8-422a-bb7c-4412fcd56f06",
      "securityDefinitions": {
        "basic_sc": {
          "scheme": "basic",
          "in": "header"
        }
      },
      "security": "basic_sc"
    }
  }
}
```

Listing 4: Example SDF mapping file, containing the instance-specific information from Listings 1 and 2 not present in the SDF model in Listing 3.

```
{
  "LampThing": {
    "@context": ["https://www.w3.org/2022/wot/td/v1.1"],
    "@type": "tm:ThingModel",
    "properties": {
        "status": {
            "type": "string"
        }
    },
    "links": [
      {
        "rel": "tm:submodel",
        "href": "#/Switch"
      }
    ]
  },
  "Switch": {
    "@context": ["https://www.w3.org/2022/wot/td/v1.1"],
    "@type": "tm:ThingModel",
    "actions": {
        "toggle": {}
    }
  }
}
```

Listing 5: Example for a Thing Model Collection with one top-level TM (`LampThing`) pointing to a submodel TM (`Switch`) using a JSON Pointer (`#/Switch`).

```
{
  "sdfThing": {
    "LampThing": {
      "sdfProperty": {
        "status": {
          "observable": false,
          "type": "string"
        }
      },
      "sdfObject": {
        "Switch": {
          "sdfAction": {
            "toggle": {}
          }
        }
      }
    }
  }
}
```

Listing 6: Example for a nested SDF model based on the TM Collection from Listing 5.