No-Code Shadow Things Deployment for the IoT

Ege Korkan[1], Emanuel Regnath[1], Sebastian Kaebisch[2], Sebastian Steinhorst[1]
[1] *Technical University of Munich,* Germany, Email: {ege.korkan, emanuel.regnath, sebastian.steinhorst}@tum.de
[2] *Siemens AG,* Germany, Email: {sebastian.kaebisch}@siemens.com

***Abstract*—The Internet of Things (IoT) has brought Internet connectivity to devices from different domains and with different constraints. This, however, requires system developers to have a deep understanding of the individual devices' functionalities to achieve a successful integration. The Thing Description (TD) standard from the World Wide Web Consortium (W3C) is able to describe such device capabilities in a machine and human-readable format. While this standardization effort already provides many benefits to developers, the lack of easy-to-deploy virtual device instances introduces issues such as an increased risk for errors during the integration phase of IoT systems.**

In this paper, we introduce a novel method to create virtual instances of IoT devices based on their TD that act like the real device, which we call the Shadow Thing Method (STM). In the development and deployment stages of IoT systems, STM can be applied to deploy an instance to act as a smart proxy to address different problems, while requiring no programming experience or effort. We show for three different use cases that in addition to the simulation benefits, STM brings scalability, reliability, and safety to the existing IoT systems, making the adoption of IoT feasible for a wider audience.

***Index Terms*—Internet of Things, Web of Things Thing Description, CPS, System Design, Simulation**

I. Introduction

One of the principal goals of the Internet of Things (IoT) is bringing Internet-layer connectivity to electronic devices. The already existing variety of electronic devices, thus, results in a large variety of IoT devices. In return, one can expect IoT devices to have different properties such as power consumption, processing power, supported protocols etc. This is commonly referred to as fragmentation and discourages building systems of IoT devices that originate from different domains, even though the integration of heterogeneous devices is another principal goal of the IoT.

The Thing Description (TD) standard [1] was introduced recently as an open description format to solve the fragmentation and interoperability problem. TD can be used to describe IoT devices with communication protocols of any kind and is human-readable as well as machine-understandable. The TD is not a standard to replace other IoT standards but it makes it possible to describe them through syntactic and semantic information. TD abstracts capabilities of devices as interactions, such as reading a temperature or moving a robot, and dictates how a user of this device should execute these interactions. This way, one can communicate and, thus, use an IoT device only by reading its TD.

Problem Statement. TD enables an easier system development in case the physical devices can be trusted and are available. However, this assumption can often not be met when we observe the life cycle of an IoT device at different stages of a system. More specifically:

With the support of the Technische Universität München – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763.

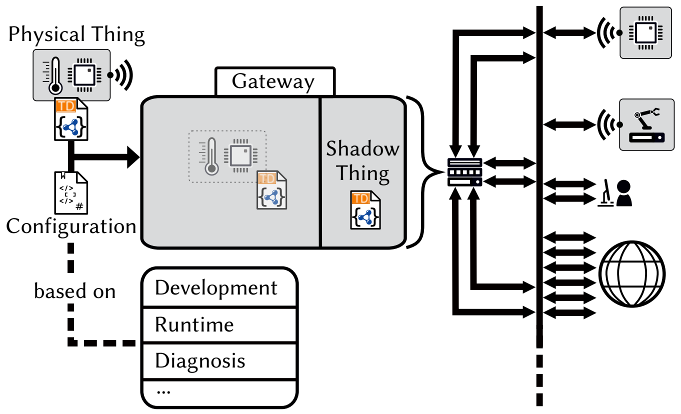

Figure 1: The Shadow Thing Method (STM) enables no-code deployment of a virtual instance of an IoT device based on the Thing Description of the device and a configuration file. The configuration file allows the STM to be used in different stages of the life cycle of an IoT device.

- System developers are facing challenges when the physical device is not available during the development phase, making it difficult to develop the control application that heavily relies on the communication with the device. Additionally, until the system-level application code is stable, it would be unsafe to interact with the physical device even if it is available.
- TD is used to describe existing IoT devices which can have resource and network constraints or can exhibit unsafe behavior in different cases. Since the device code is not always modifiable, there is a need to handle this behavior while not introducing additional implementation effort for the system developers.

Contributions. The aforementioned problems share the common need for a Shadow Thing as shown in Figure 1, an easy-to-deploy virtual device instance that acts like a real one for the system developer in different stages of the system life cycle. In this paper we introduce the No-Code Shadow Things Deployment Method (STM) that allows to deploy such Shadow Things as a smart proxy from a TD with different configurations according to their use in the life cycle of a system while not requiring any programming effort. Consequently, this paper has the following contributions:

- We introduce a method to deploy virtual IoT devices with various protocols and properties, explained in Section III as the Shadow Thing Method and its corresponding publicly available implementation, introduced in Section IV.
- Variations of the STM are presented to fulfill different purposes, according to the specific use case such as simulation, support, or adaptation, introduced in Sections III-A, III-B and III-C.
- Demonstration and evaluation of three different use cases of the STM; to provide a simulation environment for IoT developers and researchers, to support the device

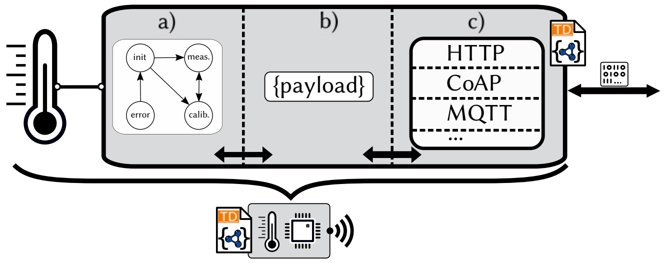

Figure 2: Thing Description abstracts the Thing logic and hardware (a) while describing payload and protocol related information to the Consumers. Here, one can see how an interaction handler would work, from protocol layer (c) to the logic (a).

in runtime by increasing scalability and to adapt the device to different applications by changing the protocol or security mechanisms, introduced in Section V.

Section VI discusses related work and Section VII concludes.

II. Thing Description

The first version of TD has been introduced in September 2017 by the Web of Things (WoT) Working Group of the World Wide Web Consortium (W3C) after years of research on how to bring Internet and Web technologies to the device level. In this section, we shortly explain this standard which sits at the core of the Shadow Thing Method (STM), by focusing on its relevant details.

TD abstracts individual capabilities into interaction affordances, where an Exposed Thing[2] is offering interactions for Consumers such as another Thing, Web browser etc. These interaction affordances are formally defined in [2] and [3], and grouped into three categories:

- Property: Exposed state of the Thing that can be read (like sensor data) or written to (like a configuration).
- Action: Functions of the Thing that can be invoked in order to trigger a physical process (like moving a robot).
- Event: Event source that can send asynchronous messages to the Consumers (like a button pressed alert).

These interaction affordances are shown in the TD of Listing 1, which describes the capabilities of a ventilator. A Consumer of this TD is able to read the rotation speed by reading the property `rotationSpeed` (lines 7-12), request the fans to rotate with a given speed (lines 16-21) and listen to an `emergencyButton` event (lines 24-26). As seen in this example, the interactions have the `forms` field which contains all of the protocol-specific information such as methods or Uniform Resource Identifiers (URIs) whereas the remaining parts give the relevant information for the application, such as the payload description.

An interaction handler uses a specific protocol, as shown in Figure 2.c, and is the actual code that responds to requests. Depending on the programming language, hardware platform or framework, the interaction handlers can be implemented in different ways which are, however, abstracted with a TD using the protocol bindings explained in [4]. In Figure 2, any communication request to the Exposed Thing will go through the protocol layer (c), deliver a payload (b), if necessary, to the Thing's logic layer (a) and the Exposed Thing will respond with a payload that follows the same path back. TD can describe the layers b and c, however the Thing logic (layer a) is not exposed to the Consumers. STM relies on the principle of layers and uses the interaction descriptions of a TD in order to create the interaction handlers.

[2]When the word Thing is used with a capital letter, a Thing means an object, either virtual or physical, that can be communicated with.

```
1{
2  "@context":"https://www.w3.org/2019/wot/td/v1",
3  "title":"ConnectedVentilator",
4  "base":"http://192.168.0.2:8080/",
5  "properties":{
6    "rotationSpeed":{
7      "type":"integer",
8      "minimum":0,"maximum":1200,
9      "readOnly":true,
10     "forms":[{"href":"props/rotationSpeed",
11               "htv:methodName":"GET"}]
12   }
13 },
14 "actions":{
15   "rotate":{
16     "input":{"type":"integer",
17              "minimum":0,"maximum":1200}
18     "forms":[{"href":"actions/rotate"}]
19   }
20 },
21 "events":{
22   "emergencyButton":{
23     "forms":[{"href":"coap://192.168.0.2:5683/button"}]
24   }
25}}
```

Listing 1: Example of a simple Thing Description. It describes a ventilator that exposes the rotation speed as a property, rotate command as an action and the press of the emergency button as an event.

III. Our Shadow-Thing Method

As mentioned in Section I, IoT system developers need an easy-to-use but flexible way of deploying virtual instances of IoT devices that have the same capabilities as a physical device according to their TD. STM addresses this problem from the Consumer point of view and creates a virtual Exposed Thing with the three layers shown in Figure 2 based on a given TD. According to the use case of the developer, the Thing logic and protocols (Figure 2, layers a and c), which are hidden behind the TD, are created differently and give our method its flexibility to be applied to different use cases. This flexibility makes it different from our previous contribution [5], where the layer a and c were used for the single purpose of creating a virtual instance of the Exposed Thing. In this section, we detail how the core of the STM works and how this core principle can be extended for different use cases.

The STM's core principle is the automatic creation of interaction handlers based on a TD. For example, an interaction handler for the property `rotationSpeed` of Listing 1 would return an integer between 0 and 1200 for an HTTP GET request to the URI in the value of `href` of line 11. As shown in Listing 1, the capabilities of these devices are bound specifically to them via URIs in the `base` and `forms` fields of the TDs. However, as recommended in [6], Consumer applications can and should be written without hardcoded URIs, protocols or security configuration, which allows the Consumer application code to be programmed purely with the application logic. Consequently, with the STM we can generate the interaction handlers that are described the same way as in the original TD but with different URIs, protocols or security mechanisms.

A. *Device Simulation in the Development Phase*

The STM can be applied to cases where the physical device is not available but its TD can be acquired in advance. We can imagine a system developer who can interact with one physical Exposed Thing but not with another one which is still being delivered. In this case, the system developer can deploy a virtual Exposed Thing from its TD without requiring any programming effort. This Exposed Thing would respond to interaction requests with data corresponding to the structural schemas and value ranges specified by its TD and support the described protocols.

As the layer a of Figure 2 cannot be described in a TD, the data would appear random by default. Optionally, the interaction handlers created by the STM can be overridden with device behavior by a developer to obtain a more reliable representation of the physical device, such as simulating how the temperature revolves over time with a certain mathematical function. This is commonly referred to as Digital Twin and adding more information into the TD standard - such that a Digital Twin model can be described directly in the device TD - is currently under discussion in the standardization group[3]. This would enable the generation of a more real-life behavior by the STM.

B. *Smart Proxy for Supporting the Physical Device*

During the runtime of systems, resource-constrained IoT devices are unable to scale up like the dedicated servers we are familiar with on the Internet. That is why gateways are proposed to handle the load of these devices. STM can be automatically deployed in such gateways to bring scalability and reliability from the point of view of a Consumer application, where a high number of requests can be handled and intermittent dropouts of the physical Exposed Thing can be hidden. Compared to the previous case, here, the STM works with the presence of a physical device.

The interaction handlers are created according to the core principle but the actual data of the physical Exposed Things are cached and relayed to the Consumers. The action affordances of physical Exposed Things generally interact with the physical world, such as rotating the ventilator. Thus, in the smart proxy, we do not cache the action affordance inputs or outputs since doing so would lead to consequences in the physical world, unknown to the Consumers.

Our approach is different from the traditional caching approaches, where generally safe and idempotent requests from the protocol point of view (such as HTTP GET) are cached. Differently, STM uses the interaction affordance semantics rather than the protocol semantics, where an HTTP GET request that invokes an action would not be cached. Furthermore, this allows us to have a protocol independent caching mechanism where one can cache for an HTTP based Exposed Thing as well as one with MQTT, CoAP, or any protocol described by the WoT Binding Templates [4]. In Figure 3, layer a2, we illustrate this use case.

[3]https://github.com/w3c/wot/blob/master/charters/wot-wg-charter-draft-2019.html

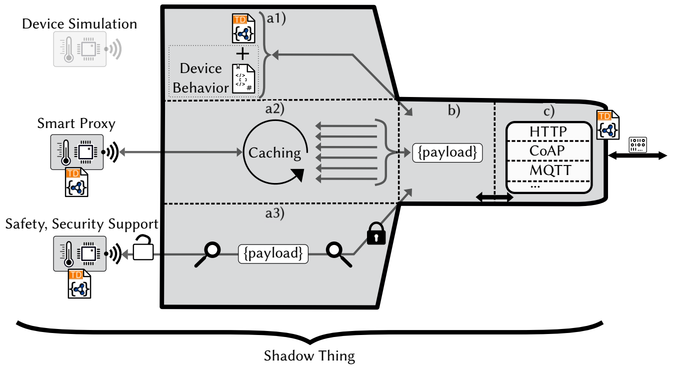

Figure 3: Shadow Thing Method allows the automatic creation of interaction handlers from a given Thing Description. Parts a1, a2 and a3 constitute the different use cases of the method, which can be chosen according to the needs of the system developer.

C. *Adapting Physical Device with Protocol, Safety, and Security Features*

Similar to Section III-B, here we explain another use case of the STM together with a physical Exposed Thing. Differently, STM can be used to automatically add features to a physical Exposed Thing or change its features without changing its source code. More precisely, by deploying a Shadow Thing between the physical Exposed Thing and the Consumer, one is able to add input/output sanitization, protocol translation, or to add/remove security mechanisms.

Input/Output Sanitization: As noted in the TD specification via multiple assertions[4], the payload structures of the data input and output should be respected by the Exposed Thing and the Consumers. However, this does not imply that there is always the verification of these data in the implementation or that a verification will work in all states of the physical device, such as in the presence of an error.

This safety issue can be mitigated by deploying the STM on a gateway where interaction handlers can be automatically supported with input and output validation. For example, rejecting unauthorized data inputs sent to the physical Exposed Thing or detecting invalid data sent by the physical Exposed Thing and informing the Consumer. This allows automatic addition of a safety layer to the existing IoT devices.

Protocol Translation: TD abstracts the interaction affordances from the underlying protocol. This means that a developer who would write a Consumer application with the Scripting API, would not need to write protocol specific code. However, not every Scripting API implementation (or any Consumer application) necessarily comes with the protocol stack the Exposed Thing requires.

By using this abstraction layer, we can use STM to translate the protocol of an Exposed Thing to another protocol that the Consumer application supports. For example, if the Exposed Thing requires a CoAP GET request to read a property affordance, we can automatically translate this protocol requirement to an HTTP GET request that can be then used with regular Web Browsers. Similarly, if the Exposed Thing requires an HTTP long polling mechanism to observe events, which is not suitable for resource-constrained devices or which can be seen as a Denial of Service attack

[4]Assertion IDs are server-data-schema and client-data-schema in https://w3c.github.io/wot-thing-description/testing/report.html

Configuration	Number of Exposed Things	Average(s)	Maximum (s)	Std Deviation (s)
4CPU Threads 8GB RAM	4 000 (4k)	0.76	0.80	0.03
	40 000 (40k)	3.88	4.02	0.08
	400 000 (400k)	40.32	41.52	0.61
	1 000 000 (1M)	111.03	112.63	0.90
32CPU Threads 64GB RAM	1 000 000 (1M)	32.64	32.92	0.17
	2 000 000 (2M)	69.51	70.01	0.31
	16 000 000 (16M)	544.42	560.62	13.63
	32 000 000 (32M)	595.21	600.61	3.67

Table I: Times taken to instantiate a given number of Exposed Things is recorded and presented in this table by average, standard deviation and maximum.

from the network provider's point of view, we can translate it to an MQTT subscription where even the broker can be started in a gateway for the given device.

Modifying Security Mechanisms: Existing IoT devices do not always come with security features or might get outdated and expose vulnerabilities. STM makes it possible to automatically add a security layer, customizable per device that can be updated without interfering with the source code of the devices. The security requirements are described in a TD with the `securityDefinitions` keyword, where it is possible to describe mechanisms such as Basic Authentication, Bearer Tokens or more. These mechanisms are applied in the Consumer application in an initialization stage and not appear in the application logic, meaning that modifying the security mechanism of a physical Exposed Thing (with the Shadow Thing) should not require any change in the application logic.

With STM, we can add a security mechanism to an Exposed Thing that comes without one. This is a common use case where a device meant for local networks coming without any security mechanisms can be secured through a gateway where the STM is applied. For the given id of the physical Exposed Thing, the configuration of STM defines the desired security mechanism. This way, STM brings security support in a granular way that can be automatically deployed.

IV. Implementation

STM is implemented as a software package in node.js runtime under the name `shadow-thing`[5,6]. We are using the `thingweb.node-wot` library[7] of the Eclipse Thingweb project [7] which provides the support for different protocols as well as for processing a TD. This library comes with the support for HTTP(S), CoAP and MQTT protocol bindings, with the development of more bindings in progress.

Our implementation comes with the core contribution of STM and the extensions for the cases mentioned in Sections III-A, III-B and III-C. For its different use cases, shadow-thing can be run in a single or in a multi-threaded mode, the latter being recommended for deploying a large amount of virtual instances. When deploying multiple instances, it is possible to create the virtual instances in Docker containers, which allows easy deployment with different resources given for each container.

In addition to creating an Exposed Thing, we have also implemented a scalable client pool that can be used to stress test any Exposed Thing with a TD. This is also supplied with the shadow-thing package.

[5]https://www.npmjs.com/package/shadow-thing

[6]We would like to acknowledge the work done by Hassib Belhaj Hassine and Teck Wan Wong.

[7]https://github.com/eclipse/thingweb.node-wot

V. Case Study

In this section, we are presenting three different case studies for the use cases mentioned in Section III.

A. Device Simulation

As mentioned in Section III-A, the basic use case of the STM is the deployment of virtual Exposed Things from a TD. After briefly demonstrating the simulation of a single device, we will show how our method is scalable to simulate thousands of devices.

1) Single Device Simulation To demonstrate basic simulation, we are taking the TD shown in Listing 1 which has multiple protocols as well as all of the different interaction affordances. We are taking the point of view of a Consumer application developer who needs to experiment with different Exposed Things and verify whether it fulfills the assertions of the TD specification. After installing the shadow-thing package simply by `npm install` command, it can be started via `npm start {tdlocation}`. The developer is met with a default configuration that can be modified to adjust the event emission intervals or the protocol.

From the Consumer application, once its TD is consumed, shadow-thing provides the same application layer logic where all of the interactions and their corresponding operation types are present. As mentioned in Section IV, the values emitted by events, read from properties and action outputs are respecting the data schema but their values appear random. Thus, one can test if the Consumer application is robust against all data types and can execute all interaction affordances with required protocols.

2) Multi-device Simulation More interestingly, the shadow-thing is very lightweight and can be scaled up to thousands of virtual instances. We see an increasing amount of research on the WoT as well as the semantic technologies around it. With our shadow-thing package, we want to provide the academic community a simulation environment where they can evaluate their methodologies such as searching for a specific interaction in a group of TDs or composing and deploying a system of Exposed Things. In such cases, the validity of the method needs to be supported with how well it can scale and therefore requires large number of Exposed Things. Thus, here we are evaluating shadow-thing on its capability to scale by focusing on the number of Exposed Things that can be deployed.

In order to reduce the amount of variables for evaluation, we are excluding the MQTT protocol since it is highly dependant on the broker, where the actual message delivery is happening. We are varying the RAM and amount of CPUs that can be allocated in order to see how many devices can be simulated by a typical developer's laptop as well as a dedicated server. Our evaluation is running on a computer with the following specifications:

- Hardware: Intel Xeon Gold 6130 with 16 cores and 128GB RAM
- Operating System: Ubuntu 18.04 LTS
- Software: Docker 18.09.7, with node:10.16.0-alpine

Evaluation Method: We give the Docker container and the node.js instance the same amount of RAM. We note the amount of time taken to generate all Exposed Things for different configurations of RAM and CPU count. We repeat the experiment 10 times for each configuration. It is important to precise that the generated instances are fully functional Exposed Things that can accept requests from Consumers and that a new Exposed Thing does not replace the previous ones (i.e. no recycling). Since the multi-threaded version of shadow-thing provides better results, we are running all of our experiments in this mode but it is possible to create all Exposed Things in a single thread. The results of our evaluation is grouped in Table I.

Observations. We observe that our method scales to a high number of instances created as long as the resources are not fully used. This means that we do not have an inherent scalability limitation in the shadow-thing architecture. More importantly, we can generate a high number (such as 1 million) of Exposed Things in under 2 minutes, in a container configuration with 4 CPU threads and 8 GB RAM, which can be considered as a common configuration among researchers and developers. For this configuration, our experiments with more than 1 million Exposed Things could not be completed due to insufficient RAM, which gives a practical upper limit of deployable Exposed Things.

To deploy a higher number of Exposed Things, one would need to dedicate more resources. We can achieve 32 million Exposed Things with a 32 thread, 64 GB RAM configuration, where the deployment took 10 minutes in the worst case. Additionally, we observe that when the number of allocated CPUs is low or if the RAM starts getting fully occupied, the generation slows down, with RAM having a more significant impact. This can be seen in cases when we try to deploy 16 000 Exposed Things, where a 15 times slower instantiation occurs when using 100 MB of RAM instead of 1 GB.

B. Device Support in Runtime

In this use-case, we deploy shadow-thing as a smart proxy as mentioned in III-B and illustrated in layer a2 of Figure 3 to help with scalability issues of IoT devices. The setup is with the following devices in the same WiFi network:

- An ESP8266 based light sensor that has a property affordance for light intensity in its TD. Implementation of this Exposed Thing is publicly available[8].
- A Raspberry Pi 4 with 1 GB of RAM where shadow-thing runs in the single-threaded mode. This is deployed automatically only using the TD of the light sensor and a caching interval.
- A computer[9] running an adjustable amount of Consumers that can communicate either with the physical light sensor or its Shadow Thing, which is set up by choosing the TD to consume. This computer simulates the load that an IoT device can be exposed to.

[8]https://wotify.org/library/ESP%20Light%20Sensor/general

[9]Hardware: i7-7500U CPU with 16GB RAM. Software: Arch Linux with Linux Kernel 5.3.12 and node.js version 13.2.0

Time to Read	Average (ms)				Std. Deviation (ms)			
Amount of Clients	1	5	10	20	1	5	10	20
Physical Device	38	71	711	n/a	3	66	2846	n/a
Shadow Thing	42	88	122	253	14	177	252	1026

Table II: The times taken to get a response from physical Exposed Thing and its Shadow Thing for a readproperty request are recorded and presented in this table by average time, standard deviation and maximum.

Evaluation Method: From the computer, we deploy an increasing number of Consumers that simultaneously read the intensity property. We have deployed Consumers ranging from 1 to 20 that read in 1s intervals for 20 consecutive iterations. Without changing anything in the Consumers, the same procedure is applied to the physical Exposed Thing directly and to its Shadow Thing running in the Raspberry Pi. We have recorded the time taken for each read request and compiled the results in Table II after removing the outliers.

Observations: Most importantly, we observe that the physical Exposed Thing is unable to serve more than around 15 clients at the same time, resulting in the `n\a` fields in the table. This requires a restart of the Exposed Thing. For a small number of Consumers (1-5), the response times from the physical Exposed Thing are better than shadow-thing in average and also deviate less. However, its performance quickly degrades after 10 Consumers, resulting in 10 times worse average reading times than 5 Consumers. shadow-thing initially performs worse, which is due to the extra communication needed with the physical Exposed Thing. However, this disadvantage becomes less apparent with a higher number of Consumers. Additionally, shadow-thing exhibits increasingly more deviation for a higher number of Consumer whereas the averages increases more linearly.

Our results can be further improved when shadow-thing is used in a multi-threaded version where the Shadow Things representing the physical device can be multiplied to keep the number of Consumers per virtual Exposed Thing low.

C. Device Adaptation

In this last use case, we demonstrate how STM can be used to adapt the physical Exposed Thing for the Consumers while requiring no programming. Specifically, we will take the following Exposed Thing[10] with the following attributes:

- Using CoAP protocol for all interactions
- No security mechanism
- No input validation

We configure the shadow-thing to consume the TD of this device with its CoAP client binding. Since it has no security mechanism, all interactions are executable by the shadow-thing. Then, we enable the HTTP server binding and re-expose the TD with HTTP protocol. Additionally, we add the BasicAuth security mechanism by configuring an id and password. shadow-thing comes with input validation enabled which results in the data sent for invoking actions and writing to properties being verified. In the case of invalid inputs, shadow-thing responds with an error message to

[10]https://wotify.org/library/Motor%20Controller%20(Pololu%20TB9051FTG%20Dual)/thingdescription

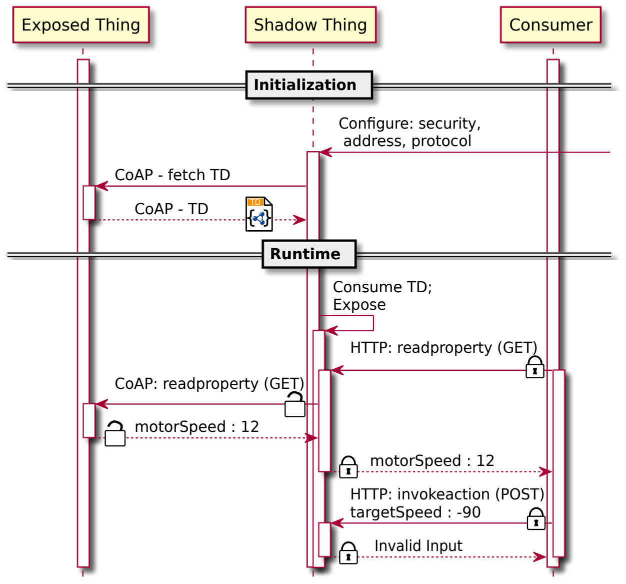

Figure 4: shadow-thing can be set up to translate protocols, change security mechanisms and to verify the inputs sent by the Consumers. Here, an Exposed Thing using CoAP protocol can be automatically connected to a Consumer using HTTP protocol by passing through the Shadow Thing.

the Consumer. This case study is fully illustrated in the UML Sequence Diagram in Figure 4. The left side represents communication with the physical Exposed Thing through CoAP protocol and no security, whereas the right-hand-side of Shadow Thing communicates with HTTP, with security mechanism and input validation.

Observations: We observe that the deployed Shadow Thing adds a similar delay in communication as in the single Consumer case of Table II. Since the mapping of one protocol to the other happens through the in the initialization stage, the protocol conversion does not add any additional delay. The differences we see are due to the security mechanism that requires a certificate to be transmitted as well. This delay comes with the advantage of more secure access to the device where only Consumers with the correct security credentials can interact with the physical Exposed Thing. Finally, we note that such adaptation requires no programming effort and that it can be used also for adapting legacy devices to work with new systems. One can even think of a use case in an industrial context where older devices with protocols like Modbus can be adapted via the STM to newer protocols and architectures, such as OPC-UA. Similar to this use-case, the shadow-thing can accomplish this with no programming effort if the protocol bindings for Modbus and OPC-UA are implemented.

VI. Related Work

Our STM can be seen as a modern variation of the well established Proxy Pattern from [8]. Compared to this original pattern, we rely heavily on the TD standard and assume IoT devices. However, in recent years, IoT device virtualization has sparked interest from industry and academia.

Amazon's Device Shadow Service [9] has been used also in academia in the context of smart cities in [10]. That method, however, relies on a software that has to run on the cloud and that is accessible only to users with a connection to the cloud. Additionally, it is currently constrained to MQTT and HTTP protocols and uses a JSON document that is less flexible than the TD standard to define a device shadow.

[11] introduces DPWSim, a simulation toolkit based on the DPWS standard. In addition to using a different standard, it is a pure simulation toolkit. While the STM can be used also as a pure simulator, it can act as a proxy in the presence of a real device.

Mozilla's Virtual Things Adapter [12] can also simulate devices and uses a description format similar to the TD. It provides a visual interface for configuration and integration to the Web Things ecosystem. However, it is bound to run on the Mozilla WebThings Gateway, whereas our implementation can run on any device that can run the node.js runtime environment, such as every computer and server as well as on many single-board computers and smartphones.

VII. Conclusion

In this paper, we have introduced a novel approach called Shadow Thing Method (STM) to create virtual instances of IoT devices in the WoT context. It is easy to deploy these instances based on the TD of the device with no programming effort. We have applied our method with its publicly available implementation to the development and runtime phases of IoT systems where we have showed how to use STM to create a simulation, a smart-proxy and a device adaptor. In three case studies, we have illustrated how STM enables scalable, reliable, and safe IoT devices by adding a layer between the Thing and its Consumer.

References

[1] S. Kaebisch, T. Kamiya, M. McCool, V. Charpenay, M. Kovatsch, "Web of Things Thing Description", W3C, Tech. Rep., 2020. [Online]. Available: https://www.w3.org/TR/2020/PR-wot-thing-description-20200130/

[2] E. Korkan, S. Kaebisch, M. Kovatsch, S. Steinhorst, "Sequential Behavioral Modeling for Scalable IoT Devices and Systems", in *2018 Forum on Specification & Design Languages (FDL)*, 2018, pp. 5–16.

[3] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura and K. Kajimoto, "Web of Things Architecture", W3C, Tech. Rep., 2020. [Online]. Available: https://www.w3.org/TR/2020/PR-wot-architecture-20200130/

[4] M. Koster and E. Korkan, "WoT Binding Templates", Tech. Rep., 2020. [Online]. Available: https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/

[5] H. B. Hassine, E. Korkan, and S. Steinhorst, "Virtual-Thing: Thing Description based Virtualization", in *Second W3C Workshop on Web of Things*, June 2019.

[6] Z. Kis, D. Peintner, J. Hund, K. Nimura, "Web of Things (WoT) Scripting API", W3C, Tech. Rep., October 2019. [Online]. Available: https://www.w3.org/TR/2018/WD-wot-scripting-api-20191028/

[7] D. Peintner, M. Kovatsch, C. Glomb, J. Hund, S. Kaebisch, V. Charpenay, "Eclipse Thingweb Project", 2018, [Online; accessed April 21, 2019]. [Online]. Available: https://projects.eclipse.org/projects/iot.thingweb

[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design patterns: Elements of reusable object-oriented software", *Reading, MA*, p. 207, 1995.

[9] Amazon Web Services, Inc. (2020) Device Shadow Service for AWS IoT. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html

[10] W. Tärneberg and V. Chandrasekaran, M. Humphrey, "Experiences Creating a Framework for Smart Traffic Control Using AWS IOT", in *Proc. of UCC '16*. New York, NY, USA: ACM, 2016, pp. 63–69.

[11] S. N. Han, G. M. Lee, N. Crespi, K. Heo, N. Van Luong, M. Brut, and P. Gatellier, "DPWSim: A simulation toolkit for IoT applications using devices profile for web services", in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, March 2014, pp. 544–547.

[12] Mozilla Foundation. (2020) Mozilla WebThings Gateway Virtual Things Adapter. [Online]. Available: https://github.com/mozilla-iot/virtual-things-adapter