Towards a Behavioral Description of Cyber-Physical Systems Using the Thing Description

Fady Salama
Ege Korkan
fady.salama@tum.de
ege.korkan@tum.de
Technische Universität München
Munich, Bavaria, Germany

Sebastian Käbisch
Siemens AG
Munich, Bavaria, Germany
sebastian.kaebisch@siemens.com

Sebastian Steinhorst
sebastian.steinhorst@tum.de
Technische Universität München
Munich, Bavaria, Germany

ABSTRACT

The World Wide Web Consortium (W3C) introduced the Thing Description (TD), a standardized and unified human- and machine-readable semantic description of Internet of Things (IoT) devices that focuses on describing how to interact with the described device using its network-interfaces. However, the TDs lack a way to describe the physical effect of said interactions on the device itself, as well as on the environment around the device, limiting its viability for cyber-physical scenarios. In this paper, we propose an extension for describing the effects of an interaction on the property affordances of a Thing in the TD as a first step towards a TD that is able to fully describe a Cyber-Physical System (CPS). We show this extension permits the generation of accurate Digital Twins, facilitates machine-aided system design and device mashup generation and allows for formal verification of the functionality of CPSs during their deployment and maintenance.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Networks** → *Cyber-physical networks.*

KEYWORDS

Internet of Things, Web of Things, behavioral description

ACM Reference Format:
Fady Salama, Ege Korkan, Sebastian Käbisch, and Sebastian Steinhorst. 2021. Towards a Behavioral Description of Cyber-Physical Systems Using the Thing Description. In *Descriptive Approaches to IoT Security, Network, and Application Configuration (DAI-SNAC '21), December 7, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3488661.3494030

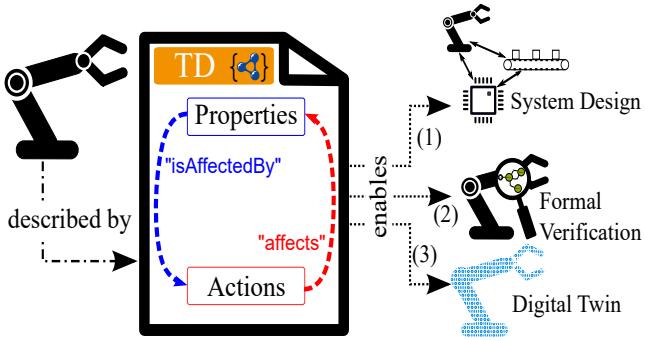

Figure 1: Our extension allows a Thing Description (TD) of a Thing to describe the effect of invoking an action on its own property affordances, as well as the on property affordances of other Things. Using extension facilitates automated device mashup generation (1), formal verification of the behavior of Cyber-Physical Systems (CPSs) (2) and generation of more accurate Digital Twin simulations (3).

1 INTRODUCTION

The advent of the Industrial Internet of Things (IIoT) brings with it a set of challenges that need to be tackled. Manufacturing facilities can no longer be considered as independent silos and the interoperability of devices is no longer insured, because sensor, industrial devices, and whole production systems from different hardware vendors and in different manufacturing facilities may need to operate and communicate together using different IIoT protocols and platforms. Designing such industrial systems so that they could communicate and function in tandem to perform a joint functionality is a non-trivial task given the high fragmentation of Internet of Things (IoT) platforms and technologies.

Therefore, the need for a unified and standardized description of industrial devices arises. As a step towards this goal, the World Wide Web Consortium (W3C) introduced the Thing Description (TD) [2], a standardized and unified human- and machine-readable semantic description of IoT devices that focuses on describing the interaction affordance that are used to interact with the described device and the network-interfaces of such affordances. The TD does not only make it easier for a system developer to design any system, including industrial systems, but also allows for machine-aided design of such systems.

The TD also facilitates interoperability of different devices across different protocols based on semantic descriptions of device interactions. Using the information in a TD, it is possible to generate a software simulation of the device, a Digital Twin, that can emulate cyber behavior of devices as seen in [5]. Such Digital Twins can aid the design process of systems before their physical deployment.

DAI-SNAC '21, December 7, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-9136-8/21/12…$15.00
https://doi.org/10.1145/3488661.3494030

1.1 Problem Statement

While the TD already provides a standardized way for describing network-interfaces of device interactions, it lacks a way to describe the physical effect of these interactions on the device itself and on the system it operates in. This limits the usefulness of the TD in describing CPS and, in turn, the ability to generate accurate cyber-physical Digital Twins.

1.2 Contribution

In this paper, we propose an extension for the TD to link the effect of an interaction on the properties of the described Thing in Section 3, which can be considered as a first step towards a full cyber-physical description of a Thing. In Section 4, we show that this addition can be used to automate system design and device mashup generation, automatically verify a system during its design phase, deployment and maintenance, and help generate more accurate Digital Twins. The rest of this paper is structured as follows: Section 2 introduces background knowledge about the technologies used in this paper, and Section 5 presents related work. Section 6 concludes this paper.

2 BACKGROUND INFORMATION

In this section, we present a brief overview of the Thing Description (TD) specification, as well as its extension, System Description (SD).

2.1 Thing Description

The TD [2] is a JSON-LD document [8] that is both human- and machine-readable and semantically describes any entity on the Web of Things (WoT). Such entities can be physical devices, or virtual entities, such as Digital Twins, and are called Things in the context of this paper. The TD focuses on describing the interface that a Thing exposes using standardized interaction affordances, which are divided into:

(1) **Property affordances**. They represent states that a Thing exposes and can be read, written to or observed.
(2) **Action affordances**. They represent either state manipulations or physical processes that can be invoked.
(3) **Event affordances**. They represent notifications and data streams that can be subscribed to.

2.2 System Description

The System Description (SD) [3] extends the TD to allow for describing network-interfaces of systems of Things, called mashups in the context of this paper. In such systems, the sequence in which the interactions from different Things are executed is essential to the correctness of the performed functionality [4]. To describe such sequences, the SD introduces the Atomic Mashup (AM) abstraction, which describes small mashups that perform a specific functionality and these are used as building blocks for bigger mashups. AMs always have the same structure: A mashup controller executes a series of unordered input interactions to get data or data streams from input devices, followed by a series of unordered output interactions/actuation to output devices.

```
1  {"title": "Conveyor Belt",
2  "properties": {
3      "speed": {
4          "type": "number",
5          "forms": [{"href": "..."}]
6      },
7      "acceleration": {
8          "type": "number",
9          "forms": [{"href": "..."}]
10     }},
11 "actions": {
12     "start": {"forms": [{"href": "..."}]},
13     "changeSpeed": {"forms": [{"href": "..."}]}}}
```

Listing 1: A Thing Description (TD) is a JSON-LD document that describes the network-interfaces of the interactions as well as metadata of a Thing. Interaction affordances are divided into property (lines 2-10), action (11-13) and event affordances. TDs do not provide a way to describe the effect of invoking an action on a property affordance.

2.3 Shortcomings

Both the TD and the SD lack a way to describe how the executed interactions affect the properties of a Thing or variables of a system. For some cases, a human agent may easily deduce such effects, i.e., a `"writeproperty"` operation will update a `"lightState"` property or an invoking an action named `"changeSpeed"` will change a numeric property affordance named `"speed"`, as seen in Listing 1. However, this is not feasible in poorly named TDs or for more complex Things in which one interaction may affect multiple properties or variables. Furthermore, a machine cannot accurately deduce the effects of interaction based only on natural language.

3 THE PROPOSAL

To remedy the shortcomings of the TD and SD as discussed in Section 2, we first formally define the coupling between interactions and properties to extract the requirements that our proposed extension needs to fulfill. We then present our proposed solution in the form of two additional keywords.

3.1 Formal Definition

We can formally describe a TD as a set of interaction affordances I that include a subset of property affordances P and a subset of action affordances A with both P and A being mutually exclusive, i.e:

$$P \subseteq I \ \wedge \ A \subseteq I \ \wedge \ P \cap A = \emptyset \tag{1}$$

Each interaction affordance exposes a set of operations as described in 2.1. The two operations that can directly alter properties of a Thing are `"writeproperty"` and `"invokeaction"`. The `"writeproperty"` operation implicitly defines which property will be altered and, but `"invokeaction"` operations lacks such an implicit definition
Hence, there is the need to describe the coupling of action affordances with one or more property affordances. We can consider this coupling as a mapping a that is defined as:

$$a : A \longrightarrow 2^{P_A} \tag{2}$$

where:

- 2^{P_A} is the power set of P_A.
- P_A is the set of properties affected by actions.

Furthermore, we can define the inverse mapping a^{-1} of Equation 2 as:

$$a^{-1} : P \longrightarrow 2^{A_P} \tag{3}$$

where:

- 2^{A_P} is the power set of A_P.
- A_P is the set of actions that affect properties.

These mappings represent the transfer function of the action in regard to the properties it affects and its inverse transfer function respectively

3.2 Requirements

Based on the formal definitions we introduced in Section 3.1, we can define the requirements that our extension should comply to.

- Our extension should be able to describe the mapping of actions to affected properties as per Equation 2.
- Our extension should be able to describe the inverse mapping as per Equation 3.
- Our extension should permit deeper description of the coupling using additional keywords and/or annotations. This would allow for describing the transfer function of the coupling.

3.3 The "affects"-keyword

Based on the first and third requirements in Section 3.2, we propose the "affects"-keyword for action affordances. This keyword accepts an object or array of objects, with each object corresponding to a property that is affected the action invocation. These objects must at least contain the ID in form of a JSON Pointer [1] of said property but can be extended to include additional keywords or semantic annotations, permitting the description of the mapping as a transfer function[1]. This implements the coupling defined in Equation 2.

3.4 The "isAffectedBy"-keyword

Complying with the second and third requirements in Section 3.2, we propose adding the "isAffectedBy"-keyword for property affordances. The keyword accepts an object or array of objects, with each object corresponding to an action that affects the property when invoked. These objects must at least contain the ID in form of a JSON Pointer [1] of said action but can be extended to include additional keywords or semantic annotations, permitting the description of the mapping as an inverse transfer function[1]. This implements the coupling defined in Equation 3.
An example of a TD using both keywords can be viewed under Listing 2.

4 EFFECT ON CPS DEVELOPMENT CYCLE

Adding the "affects" and "isAffectedBy"-keywords formally defines which properties are affected by which actions, removing ambiguity not only for human agents, but also for machines. This opens the possibility for machine-aided design, verification, and maintenance of CPS, which we will explore in this section.

[1]There are myriad methods for describing transfer functions and their inverses and comparing them is out of scope of this paper.

```
1  {"title": "Conveyor Belt",
2  "properties": {
3      "speed": {
4          "isAffectedBy": [
5          {"id": "/actions/starts"},
6          {"id": "/actions/changeSpeed"}]},
7      "acceleration": {
8          "isAffectedBy":
9          {"id": "/actions/changeSpeed"}}},
10 "actions": {
11     "start": {"affects":
12         {"id": "/properties/speed"}},
13     "changeSpeed": {"affects":
14         [{"id": "/actions/starts"},
15         {"id": "/actions/changeSpeed"}]}}}
```

Listing 2: Using our proposed extension on the TD in Listing 1 results in this extended TD, which clearly specifies the coupling between the action affordances and property affordances (Line 5 with Line 11 and Line 13, Line 8 with Line 13). The other keywords were removed for brevity.

4.1 System Design and Mashup Generation

When designing a mashup in CPSs, a human agent needs to consider which interactions need to be executed, but also the correct sequence in which to invoke these interactions.
While the SD offers the means to describe such mashups, the need to explore the design space of mashups in a system remains. A manual exploration of such a design space in industrial scenarios is arduous, error-prone, and infeasible given the exponential increase in possible mashups with increasing number of devices/interactions. Thus, machine-aided exploration of the design space is essential. Adding the proposed keywords ensures that a machine can understand the coupling between interactions affordances in the same TD as well as between different Things in an SD and can take advantage of these keywords to automatically generate mashups using algorithms.

As an example, we can use an algorithm to find all the actions that can be invoked to alter a specific property using the "isAffectedBy"-keyword. Such an algorithm can be used to find alternative execution paths that could lead to the same result, and can be useful for designing fault-tolerant distributed systems.

4.2 Formal Verification of CPS

Explicitly specifying which properties are altered by invoking an action permits a machine to automatically and formally verify the operation of CPSs during design, deployment, and maintenance phases. To showcase this, we look at the example of a TD that contains 2 action affordances A_1 and A_2 and three property affordances P_1, P_2, and P_3. A_1 can alter P_1 and P_2, A_2 can alter P_2, and P_3 is not altered by either A_1 or A_2. We can formally describe this setting using Equations 2 and 3 as the following:

$$a(A_1) = \{P_1, P_2\} \tag{4}$$

$$a(A_2) = \{P_2\} \tag{5}$$

$$a^{-1}(P_3) = \emptyset \tag{6}$$

We also define a set of atomic propositions:

- $i(x)$: action x is invoked.
- $c(y)$: property y changed.
- $e(z)$: property z changed due to sensing the environment

and, based on these atomic propositions, we define some ground truths using propositional logic and LTL-formulas[7]:

- *Liveness*: Invoking an action A_k should change the properties it affects in the next state.

$$i(A_k) \longrightarrow \mathbf{X}\, c(P_l) \qquad \forall P_l \in a(A_k) \tag{7}$$

- *Safety 1*: A property can only be altered when an action that alters it is invoked or the environmental property it senses changes.

$$\neg c(P_k) \longrightarrow \neg c(P_k)\ \mathbf{W}\ (i(A_l) \vee e(P_k))\ \forall A_l \in a^{-1}(P_k) \tag{8}$$

- *Safety 2*: Invoking an action does not alter any properties other than the properties specified in its mapping.

$$\Big(i(A_k) \wedge \neg\mathbf{X}\, e(P_l)\Big) \longrightarrow \neg\mathbf{X}\, c(P_l) \qquad \forall P_l \notin a(A_k) \tag{9}$$

Based on these ground truths, we can formally verify the following properties of our example system:

- From Formula 7:

Liveness of Action A_1:

$$\mathbf{G}\Big(i(A_1) \longrightarrow \mathbf{X}(c(P_1)\ \wedge\ c(P_2))\Big)$$

Liveness of Action A_2:

$$\mathbf{G}(i(A_2) \longrightarrow \mathbf{X}\, c(P_2))$$

- From Formula 8:

Safety of property P_1:

$$\mathbf{G}\Big(\neg c(P_1) \longrightarrow \neg c(P_1)\ \mathbf{W}\ (i(A_1) \vee e(P_1))\Big)$$

Safety of property P_2:

$$\mathbf{G}\Big(\neg c(P_2) \longrightarrow \neg c(P_2)\ \mathbf{W}\ (i(A_1) \vee i(A_2) \vee e(P_2))\Big)$$

Safety of property P_3:

$$\mathbf{G}(\neg c(P_3) \longrightarrow \neg c(P_3)\ \mathbf{W}\ e(P_3))$$

- From Formula 9:

Safety of action A_1:

$$\mathbf{G}((i(A_1) \wedge \neg\mathbf{X}\, e(P_3)) \longrightarrow \neg\mathbf{X}\, c(P_3))$$

Safety of action A_2:

$$\mathbf{G}\Big(\Big(i(A_2) \wedge \neg\mathbf{X}\ (e(P_1) \wedge e(P_3))\Big) \longrightarrow \neg\mathbf{X}\ (c(P_1) \wedge c(P_3))\Big)$$

Such formulas can be automatically constructed based on the keywords without any human intervention and can be used by a machine to validate a system during its simulation, deployment, or maintenance to automatically detect errors such as P_3 changing when A_1 is invoked.

4.3 Digital Twin Generation

Using our proposed extension, the TD is able to capture the physical behavior of a Thing by describing the transfer functions of its actions. A TD or SD can, therefore, be considered as a sufficient description document for automatic generation of Digital Twins, eliminating the need for reading the specifications of a Thing or the need for to physically access a deployed system to test its properties.

5 RELATED WORK

There have been other ontologies in literature that aim to describe the physical behavior of CPS. [6] proposes an ontology that describes stateful WoT environments by describing the effect of actuators on environmental parameters that impact the state of the system and outlining which sensors measure these parameters. Its main goal is to facilitate automatic system composition. However, in contrast to our paper, it does not describe the effect of actuation on the devices.
[5] implemented Digital Twin generation based on TDs. However, a faithful representation of the device required a human agent to manually specify the real-life behavior of the device. Our paper here tries to be a first step towards solving that shortcoming.

6 CONCLUSION

In this paper, we proposed an extension for the TD specification that allows for describing the coupling between action invocations and property affordances in a TD. We have formally defined this coupling as a mapping between action affordances and property affordances, and used this formulation to define a JSON-LD compliant implementation in the form of two additional keywords. We showed that, using our extension, it is possible to automatically generate device mashups, verify the correctness of CPSs properties using LTL-formulas and generate more accurate Digital Twins.

REFERENCES

[1] Paul C. Bryan, Kris Zyp, and Mark Nottingham. 2013. JavaScript Object Notation (JSON) Pointer. RFC 6901. https://doi.org/10.17487/RFC6901

[2] Sebastian Kaebisch, Takuki Kamiya, Michael McCool, Victor Charpenay, and Matthias Kovatsch. 2020. Web of Things (WoT) Thing Description. https://www.w3.org/TR/2020/WD-wot-thing-description11-20201124/

[3] Adrian Kast, Ege Korkan, Sebastian Käbisch, and Sebastian Steinhorst. 2020. Web of Things System Description for Representation of Mashups. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. 1–8. https://doi.org/10.1109/COINS49042.2020.9191677

[4] Ege Korkan, Sebastian Kaebisch, Matthias Kovatsch, and Sebastian Steinhorst. 2018. Sequential Behavioral Modeling for Scalable IoT Devices and Systems. In *2018 Forum on Specification Design Languages (FDL)*. 5–16. https://doi.org/10.1109/FDL.2018.8524065

[5] Ege Korkan, Emanuel Regnath, Sebastian Kaebisch, and Sebastian Steinhorst. 2020. No-Code Shadow Things Deployment for the IoT. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. 1–6. https://doi.org/10.1109/WF-IoT48130.2020.9221368

[6] Mahda Noura and Martin Gaedke. 2019. WoTDL: Web of Things Description Language for Automatic Composition. In *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. 413–417.

[7] Salomon Sickert. 2016. Linear Temporal Logic. *Archive of Formal Proofs* (March 2016). https://isa-afp.org/entries/LTL.html, Formal proof development.

[8] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin, and Niklas Lindström. 2020. JSON-LD 1.1. https://www.w3.org/TR/2020/REC-json-ld11-20200716/