Standard Behavior Descriptions for the Web of Things

Victor Charpenay
Chair of Technical Information Systems,
Friedrich-Alexander Universität (FAU) Erlangen-Nürnberg
victor.charpenay@fau.de

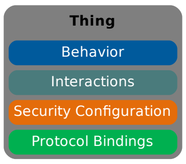
April 22, 2019

Abstract

This article motivates the need for an extensive behavior description framework as part of the collection of Web standards for the Web of Things (WoT). The existing WoT API represents a good starting point to describe the behavior of 'things', as well as more complex automation systems running on WoT. To make a WoT script easy to expose and exchange, high-level constraints on the 'things' it consumes (and exposes) should be exposed as well.

1 INTRODUCTION

This article presents a possible extension to the Web of Things (WoT) framework[1] standardized by the World Wide Web Consortium (W3C). This framework is structured around building blocks like the Thing Description (TD) model[2] and the WoT Scripting API[3]. This modular construction allows for the development of layered runtime environments for WoT as the engine of intelligent 'things'.



Source: W3C

Of all layers, the one that got the least attention in the recent W3C standardization process is the upper one: *behavior*. As the main architecture document puts it, "the behavior aspect of a Thing includes both lifecycle management (...) but also the operational behavior of the Thing"[4]. To some extent, the WoT Scripting API addresses the latter aspect of *operational* behavior, though in a non-normative way. This article motivates the need for an extensive behavior description framework and elaborates a proposal on top of the WoT API, with relation to the TD model.

2 DESCRIBING THE BEHAVIOR OF THINGS

The general architecture of WoT mostly finds application in industrial use cases, where the decentralized nature of the Web allows for robust and adaptive automation systems. The seminal idea of WoT was indeed to facilitate Web mash-ups between sensor inputs and actuator outputs.

1 https://www.w3.org/WoT/WG/
2 http://w3.org/TR/wot-thing-description/
3 https://www.w3.org/TR/wot-scripting-api/
4 https://www.w3.org/TR/wot-architecture/

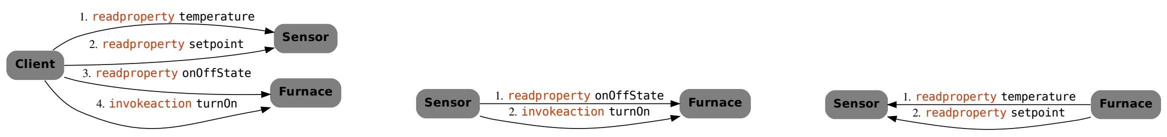


Figure 1: Alternative sequences of interactions among elements of a thermostat system; interactions are either mediated (left) or peer-to-peer (middle & right)

The W3C standard for WoT focuses on individual 'things', the atomic elements of a WoT system, insofar as the whole framework relies on the TD model, which specifies interaction affordances 'things' can expose. However, complex interaction patterns may emerge among WoT agents such that a collection of TD documents would not suffice to explain the general behavior of an automation system. As the complexity of these systems grows, it becomes urgent to not only describe 'things' in a formal way but also to describe their behavior.

2.1 *Challenge: Web of Things Scripts as Behavior Descriptions*

To illustrate the problem of describing behaviors, let us take the traditional WoT example of a temperature sensor interacting with a heating system in a thermostat application. It is mentioned in the following terms in the W3C WoT architecture document:

> The behavior implementation specifies how other behavior of a Thing is implemented, which may or may not be directly related to an interaction. For example, a thermostat may need to execute a control loop to control a furnace based on a sensor reading. This does not directly relate to a specific interaction, although it might be controlled by a property (such as target temperature) that can be modified by an interaction.

Even in such a simple example with two 'things', several interactions can take place to achieve the same goal of maintaining the temperature of some room constant. If we assume the existence of two connected 'things', a temperature sensor and a furnace, three cases can be identified. First, interactions can be *mediated* by a computer agent controlling the whole system by reading the properties of both 'things' and invoking actions on the furnace. Interactions can also take place directly between the sensor and the furnace, in which case they are *peer-to-peer* and can be either driven by the sensor or by the furnace. Figure 1 shows possible sequences in the three cases.

Because interactions are not unique, they cannot be a relevant descriptor of the thermostat behavior. The problem can however be seen from a different perspective, namely in terms of WoT scripting. The same script can generate the interactions on Fig. 1 in all three cases, depending on which WoT runtime it is executed: when interactions are mediated, the script is run on the client while in both peer-to-peer sequences, the script is run on the 'thing' driving the interactions (either the sensor or the furnace). Such a script can be written solely against the WoT API, as demonstrated in Listing 1.

```
var sensor  = wot.consume({...}), // temperature sensor TD
    furnace = wot.consume({...}); // furnace TD

function regulate() {
    var actual  = sensor.properties.temperature,
        desired = furnace.properties.setpoint,
        isOn    = furnace.properties.onOffState;

    if (actual < desired && !isOn) furnace.actions.turnOn.invoke();
    else if (isOn) furnace.actions.turnOff.invoke();
}

setInterval(regulate, 1000);
```

Listing 1: WoT script implementing a thermostat (ECMAScript)

Another level of complexity is added when considering recursive definitions for automation systems. Again, a single call to wot.expose would expose the whole thermostat as a 'thing', e.g. with a writable setpoint property to control temperature. In general, the WoT API is a powerful tool to describe the behavior of WoT systems. The next step in the standardization process may be to facilite the exchange of WoT scripts.

2.2 *Motivation*

As already mentioned, the WoT API is not a normative document. Within the W3C working group, there was the intuition that the TD model would have a higher value if it came with a programmatic interface, per analogy with HTML and the DOM API. This intuition, deserves being followed in more depth as the example above illustrates.

However, none of the use cases the group considered requires exposing WoT scripts themselves on the Web. At the time, the main argument in favor of a WoT API was code reusability: a thermostat script could be registered as a package by a package manager like npm[5] or PyPI[6] for reuse in other scripts. Arguably, there is no need for the W3C to standardize anything like a package manager. In the field of WoT, the Node-RED platform[7] already plays that role.

Yet, we will see in the next section that the current API specification is hardly enough for reusability and there are other good reasons to expose and exchange WoT scripts. Among others: 1. automatic (re)deployment of an application onto a WoT runtime, 2. simulation of the internal behavior of a system in a so-called Digital Twin[8] and 3. interaction replay in case of failure or liability testing as required in some industries.

2.3 *Main Issues*

The thermostat script of Listing 1 makes implicit assumptions about what the input TD documents include. Listings 2 & 3 make these assumptions explicit. However, one should not expect developers willing to reuse the script in their own code to find these assumptions by themselves. There should therefore be a mechanism to express requirements for a script in

5 https://www.npmjs.com/
6 https://pypi.org/
7 https://nodered.org
8 Edward Glaessgen and David Stargel. "The digital twin paradigm for future NASA and US Air Force vehicles." In: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. 2012, p. 1818.

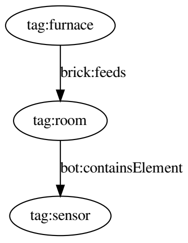
terms of TD structures. The listings below are in fact JSON-LD *frames*, a sort of template TD documents can match[9].

```
{
  "@context": "...",
  "id": "tag:sensor",
  "properties": {
    "temperature": {
      "type": "number"
    },
    "setpoint": {
      "type": "number"
    }
  },
  "actions": {}
}
```

Listing 2: temperature sensor TD

```
{
  "@context": "...",
  "id": "tag:furnace",
  "properties": {
    "onOffState": {
      "type": "boolean"
    }
  },
  "actions": {
    "turnOn": {},
    "turnOff": {}
  }
}
```

Listing 3: furnace TD



In addition to this frame matching problem, the thermostat script lacks semantics: the script should not apply to all temperature sensors and all furnances but only to those that relate to the same room. The preferred way to convey semantics on the Web is to use Web ontologies. The relation between the sensor and the furnace can e.g. be expressed using terms of the Brick schema[10] and the Building Topology Ontology (BOT)[11].

The semantics of individual interactions between 'things' can be given by some ontological expression that relates these 'things' with each other[12]. Technically, such expression could also be expressed as JSON-LD frames. However, another issue arises in this case: the semantic alignment between existing Web ontologies and the TD model. Because the TD model does not link to any of the Web ontologies mentioned above, finding the proper terms for a given application is arduous. As most WoT scripts consume several TD documents, the minimum semantics that would be necessary for on-target development should include formal relations between these TDs.

3 EXISTING TECHNOLOGIES

As already mentioned, Node-RED is a technology with which scripts can be easily exposed and exchanged. Node-RED is a flow-based programming environment designed for non-developers to create mash-up applications. It is similar in spirit to 4diac[13], a development environment based on Eclipse for distributed control systems. 4diac is an implementation of the IEC 61499 standard[14].

3.1 *High-Level Taxonomy*

Both Node-RED and 4diac include the possibility of defining the general behavior of an application (like a thermostat) as the composition of sub-applications (temperature sensor, furnace). They however differ in the way

[9] https://www.w3.org/TR/json-ld-framing/
[10] http://brickschema.org/
[11] https://w3id.org/bot
[12] Victor Charpenay, "Semantics for the Web of Things: Modeling the Physical World as a Collection of Things and Reasoning with their Descriptions," PhD Thesis, Universität Passau, Passau, *submitted* 2019.
[13] https://www.eclipse.org/4diac/
[14] "IEC 61499 Function blocks – Part 1: Architecture, Edition. 2.0," International Electrotechnical Commission (IEC), IEC 61499-1:2012, 2012.

the behavior of base applications (or *nodes* or *function blocks*) are specified. In Node-RED, they are simply pieces of ECMAScript code while IEC 61499 requires to define them in terms of control charts, i.e. state-transition machines.

More generally, behavior descriptions can be of one of the following types (non-exhaustive): 1. *process-oriented* (i.e. state-transition machines), 2. *numeric* (e.g. transfer functions from control theory), 3. *rule-based & knowledge-based* (e.g. when modeling intelligent agents) and 4. *statistical* (mostly the result of some machine learning process, like pre-trained neural networks). By allowing arbitrary scripts, Node-RED theoretically encompasses all categories since ECMAScript is a Turing-complete programming language. It is however hard to optimize to specific problems, e.g. when time constraints must be met.

3.2 *Risks & opportunities*

The reader may find analogies with earlier attempts to standardize the semantics of Web services. OWL-S[15] and WSMO[16] both include something like behavior descriptions. It has become clear though that these attempts have failed given a generally low adoption. An obvious risk for the W3C is to repeat the story with WoT and a potential semantic module. On the other hand, the significant success of Node-RED may provide evidence that there is a need for such an ecosystem.

It immediatly follows that another important risk in the further standardization of WoT scripts is the competition with Node-RED. However, this risk becomes an opportunity when considering some of its well-known shortcomings. One in particular, which originates from a simplistic module identification mechanism, may be elegantly solved with full Web addressing (using IRIs). As a result, W3C standardization may complement (maybe even improve) Node-RED with positive results on both sides.

4 ROADMAP & CONCLUSION

As outlined in Sec. 2.3, two main issues limit the reusability of WoT scripts, which in turn can be both addressed with the concept of JSON-LD framing. The first task to assign a future working group on WoT could consist in studying the integration of framing in the current WoT API.

A second task may be the alignment of the existing TD model with other WoT-related ontologies like Brick and BOT, with focus on usability. This task may run in parallel to the task mentioned above.

Another aspect that may be of interest is the mapping from specialized approaches like state-transition machines or rule-based inference to ECMAScript. Despite the theoretical agnosticism of the current WoT API, practical experience shows that "any application that *can* be written in JavaScript, *will* eventually be written in JavaScript", as Jeff Atwood puts it[17].

In fact, WoT scripts exchanged as behavior descriptions may never be executed as JavaScript code. They may as well be used for pure analytics only (e.g. for liability testing). Yet, the versatility of the language makes it the ideal tool to bring WoT to the front stage of industrial systems.

[15] https://www.w3.org/Submission/OWL-S/
[16] www.wsmo.org/
[17] https://blog.codinghorror.com/the-principle-of-least-power/