W-ADE: Timing Performance Benchmarking in Web of Things

Verena Eileen Schlott[1][0000−0002−7777−6291], Ege Korkan[2][0000−0003−4910−4962], Sebastian Kaebisch[3][0000−0002−0544−4204], and Sebastian Steinhorst[2][0000−0002−4096−2584]

[1] Ludwig Maximilian University of Munich, Munich, Germany
verena.schlott@campus.lmu.de
[2] Technical University of Munich, Munich, Germany
{ege.korkan, sebastian.steinhorst}@tum.de
[3] Siemens AG, Munich, Germany sebastian.kaebisch@siemens.de

Abstract. As the number of devices participating in the Internet of Things (IoT) rapidly grows, the challenge of interoperability across IoT platforms becomes more apparent. In order to limit fragmentation of IoT development and improve compatibility, web mechanisms and technologies can be applied, forming the Web of Things (WoT). The World Wide Web Consortium (W3C) supports the standardization of WoT by providing a platform-independent specification called Thing Description (TD). It is a machine-readable document that semantically describes metadata, interactions and interfaces of a device, indicating its functionality. However, it does not provide any information about timing performance, which is crucial for the design of optimal system compositions. In this paper, we present W-ADE, a development environment for WoT and TD that facilitates manual timing measurements and automated timing performance benchmarking of Thing interactions, merely with a TD available. Timing performance is guaranteed systematically, hence allowing optimization during the design phase of Thing mashups. Our evaluation shows that with 99.9% confidence W-ADE can predict average interaction timing performance within a range of +/- 5%, and is able to provide approximate network-independent static timing performance benchmarks for interaction affordances to 99.93%. To enable the design of heterogeneous IoT applications based upon these timing requirements, a proposal on how to annotate a TD based on the measured performance data is made.

Keywords: Web of Things · Thing Description · Timing Performance · Performance Benchmarking

1 Introduction

The *Internet of Things* (IoT) is a system of physical devices, such as sensors or actuators, which are able to communicate over various IP-level networking interfaces and eventually can be connected to the Internet. These smart things, also referred to as *Things*, enable us to monitor and interact with the physical world in a fine-grained spatial and temporal resolution. [1]

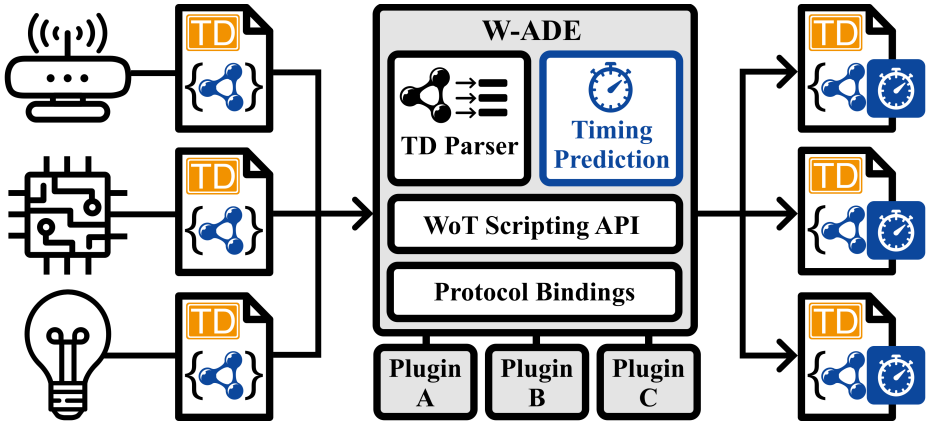

Fig. 1. The Web of Things (WoT) development environment W-ADE is a standalone application based upon the W3C WoT Architecture [6]. Its core includes a WoT runtime, protocol bindings, a Thing Description (TD) parser, and a timing performance benchmarking functionality. It can be extended with further plugins. W-ADE takes a TD as input and is able to return it, annotated with timing performance benchmarks.

However, limitations become visible as soon as multiple Things from diverse vendors are integrated into one system. As universal application protocols and platform-independent standards are missing, companies come up with their own solutions, leading to fragmentation of IoT development. Thus, it requires complex integration work, technical expertise, and it is mostly mandatory to have real devices available to build heterogeneous system compositions. [2]

The World Wide Web and its associated technologies are capable of providing solutions for the fragmented progression of IoT and offer the foundation for the next steps beyond basic network connectivity. Hence, web mechanisms are nowadays used to facilitate communication with IoT platforms - independent from their underlying technologies. This approach of integrating real-world devices into the Web is called the *Web of Things* (WoT). [3, 4]

The World Wide Web Consortium (W3C) is seeking to standardize this web layer for IoT to enable effortless integration of heterogeneous devices. The core concept in this process is the *Thing Description* (TD) specification [5], further discussed in Sec. 2. A TD is an abstraction of a Thing's capabilities: It semantically describes its metadata, interfaces, and available protocols. It acts as exposed interface, facilitating the communication with the described IoT instance.

1.1 Problem Statement

IoT applications are often composed of not only a single but multiple devices, also referred to as *mashups*. The TD facilitates to design them without having the actual Things or other device documents (e.g. system specification) at disposal. To create reliable TD-based mashups, it is inevitable to have data on timing performance of the included Things. Timing is defined as the sum of

the round trip delay time (RTD) plus the required processing time of the client application and Thing. Timing performance can therefore be understood as the required time to interact with a Thing. This is further discussed in Sec. 4. However, TDs currently do not provide such information. To obtain data on timing performance, a time-consuming manual process needs to be accomplished: for each protocol a compatible service has to be started and timing performance has to be measured manually. Although many qualitative and quantitative studies on performance of Things have been conducted, as outlined in Sec. 7, they are rather focusing on protocol or network than general application logic timing performance. Moreover, there is no tool available, which is capable of measuring timing performance, supports diverse IoT protocols (besides common web protocols) and is able to generate timing performance benchmarks at the same time. The question on how timing performance can be measured with only a client-side application available, as it usually is the case when including third party IoT devices, also remains unexplored.

1.2 Contribution

In this paper, we introduce *W-ADE*, the missing foundation block of a development and testing environment for WoT, TDs and mashups, illustrated in Fig. 1. It facilitates the automated generation of timing performance benchmarks as well as annotating a TD with the produced data - with only the associated TD available. In particular, the following contributions are made:

- In Sec. 3, we enable system designers to invoke single interactions of a Thing independent from its protocol, based only on its TD, and allow them to retrieve the associated timing performance.
- In Sec. 4 and 6, we introduce and evaluate a technique to automatically produce static and dynamic timing performance benchmarks for device interactions, giving estimations for worst-, best- and average-case execution with confidence interval limits.
- In Sec. 5, we propose a vocabulary set, aligned with the TD specification, to annotate existing TDs with observed timing performance.

Our approach is then compared to related work in Sec. 7. Finally, conclusions are drawn in Sec. 8.

2 W3C Web of Things

Our concept of providing information on timing performance evolves around the Thing Description standard. TD is one of the building blocks associated with the W3C WoT Architecture [6], which aims to prevent the further fragmentation of IoT development. The main idea is exposing device capabilities as resources in a description-oriented fashion through the WoT interface, that is, network interactions modeled as *Properties*, *Actions*, and *Events* [7]. This information can then be processed and interpreted by a *WoT Consumer*, also referred to as *Consumer*, an entity, for example another device, browser, or web application that is able to understand TDs and interact with Things [6].

Other important building blocks are the *WoT Scripting API* [7] and *WoT Binding Templates* [8]. The Scripting API is the description of a programming interface, representing the WoT Architecture. It allows scripts to discover, operate, and expose Things. The WoT Binding Templates provide guidelines on how to define Protocol Bindings for the description of network-facing interfaces. [8]

2.1 Thing Description

In the WoT context, the TD acts as a defined representation of a Thing and can be considered the entry point for communication. As TDs are encoded in JSON-LD [9] format, they are machine-readable as well as human-understandable. The main goal is to preserve and complement existing IoT standards and solutions [6]. Thus, the TD is not a proposition for a new protocol to replace other standards but a way to represent them through syntactic and semantic information [10].

One of the TD's main parts is the interaction model, a formal definition of mapping application intent to concrete protocol operations [6]. A TD interaction can therefore be understood as the description of a specific capability of a Thing, representing the data structure, access protocol and access link [10]. Consequently, a TD instance comprises a list of a Thing's interactions and how to access them. The interaction affordance is based upon the before-mentioned WoT paradigms:

- **Properties**: Exposed values of a Thing that can be *read* (e.g. sensor data), *written* (e.g. to set configuration parameters) or *observed*.
- **Actions**: Invoking them triggers physical, possibly time consuming processes (e.g. moving a robot arm) or functions inside the Thing.
- **Events**: Used for signaling asynchronous notifications that are triggered by events (e.g. a pressed button alert).

An example TD including a Action, Event, and Property is shown in List. 2.1.

2.2 Thing Description Based Mashups

As IoT systems usually consist of multiple devices, it is important to shift the focus towards mashups. Mashups in WoT are associated to digital mashups in Web 2.0. These describe the technology of composing modularized web applications to create entirely new services [11]. Respectively, creating WoT mashups expresses the process of aggregating WoT-enabled Things[4] to form new applications. This is done by chaining together multiple interactions, whereby the TD provides required information. A smart-home mashup could for example compose a light sensor and window shutters that are opened as the sun rises.

2.3 Importance of Timing Performance Benchmarking for Mashups

In order to build reliable physical mashups, a way to analyze, describe, and generate timing performance benchmarks for included Thing interactions has to be

[4] A Thing, that is accessible via its TD and can be consumed.

found. Knowledge on timing performance is especially important if involved interactions trigger physical processes. As these executions can take a considerable amount of time. Furthermore, when the sum of different interactions influences the total mashup time, it is highly relevant to be able to extract data on their individual timing performance from their description. This becomes even more valuable when a mashup is more complex, due to including many interactions or having dependencies on each other's responses. Timing performance information is also required during the mashup's design phase. As then included Things might not be available and individual interaction request times cannot be measured.

Another example for the importance of the availability of timing performance benchmarks in a TD is, when a client application persistently requests data from a Thing (polling). It could send requests in a shorter time period than the device needs to process. This potentially leads to malfunctions or to a system overload. However, the TD does not yet include any performance related information such as timing behavior, measurement context or precision. As a remedy, this paper introduces a way to integrate timing aspects into TDs.

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "title": "Coffee-Machine",
    "securityDefinitions": { "basic_sc": {"scheme": "basic", "in":"header"},
    "security": ["basic_sc"],
    "base": "coaps://coffee-machine.example.com:5683",
    "properties": {
        "status" : { "forms": [{"href": "properties/state"}]}
    },
    "actions": {
        "brew" : {"forms": [{"href": "actions/brew"}]}
    },
    "events":{
        "error": {
            "data": {"type": "string"}, "forms": [{"href": "events/error"}]
}     }     }
```

Listing 2.1. A Thing Description for a smart coffee machine that exposes the machine status, a brewing action and an error event functionality, together with their URIs.

3 W-ADE: API Development Environment for WoT

To solely measure the time it takes to invoke an interaction, several operations need to be performed. Depending on a Thing's implementation and the choice of protocol, a compatible service which can communicate via this protocol eventually needs to be started on the Consumer. Then, the desired endpoint has to be extracted and a request to execute it has to be sent. Subsequently, the time until the response is received has to be manually measured. To obtain representative results, this process would have to be repeated numerous times. Depending on the number of interactions, the level of their diversity and the quantity of services that need to be utilized, this can become a time-consuming and error-prone process. To minimize the susceptibility to errors and overall lighten this series of actions, we developed *W-ADE*: *Web of Things API Development Environment*. It simplifies the TD-based interoperation with devices, can be expanded with required protocols and facilitates timing measurement of Thing executions.

3.1 Application Features and Implementation

W-ADE possesses WoT specific functionalities, typical features needed for API interaction, and the possibility to measure timing. Its core is based upon the W3C Scripting API reference implementation[5], making it compatible with WoT paradigms. It is able to parse and interpret TDs, enables users to edit them and execute chosen interactions in a specific order. W-ADE acts as Consumer to communicate with virtual or physical entities over various protocols[6]. As it supports diverse protocols existing in IoT ecosystems and web browsers are usually restricted, it is realized as a standalone Electron[7] application[8].

Since a TD is the interface for interactions and accessing API endpoints is a main use-case, functionalities, such as sending requests with optional input values and displaying responses, are implemented. Another substantial feature is entering, storing, and applying required security credentials. Beyond that, W-ADE measures the overall time it takes to send a request, process it on the target application, and finally receive a response. Utilizing the Node.js feature `process.hrtime()`, high-resolution real time measurements in nanoseconds are available. This and the size of the sent or received data, is then displayed. Further, W-ADE's architecture facilitates custom plugins, allowing it to be easily extended by numerous already existing WoT implementations. An overview of its system architecture is presented in Fig. 1.

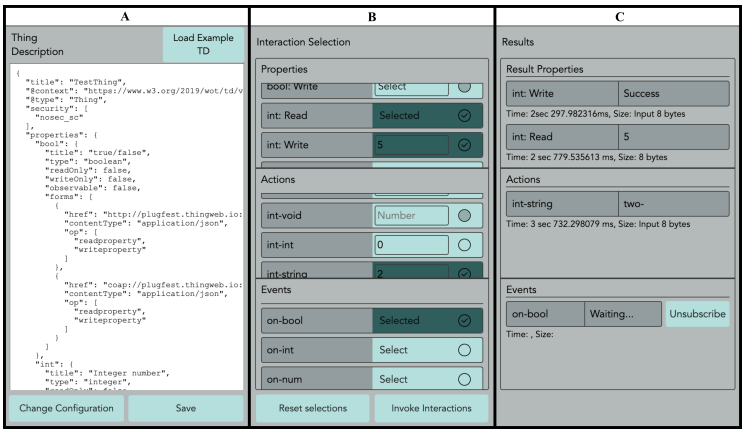

Fig. 2. W-ADE's GUI. Thing Descriptions can be uploaded and edited in the Editor (A). Interactions are parsed and visualized in (B), where input values can be entered and interactions can be selected. Results, including measured communication time and payload size of input or output are displayed in (C).

[5] Node-wot (`https://github.com/eclipse/thingweb.node-wot`) is based upon the JavaScript runtime Node.js (`www.nodejs.org/`).

[6] Embedded Binding Templates [8] enable the incorporation of further protocols including custom ones and thus, facilitate interoperability for diverse vendors.

[7] A JavaScript based framework, which allows to build cross platform applications.

[8] W-ADE is available here: https://github.com/tum-esi/wade

3.2 Workflow

To retrieve timing performance data of a Thing in W-ADE, first a TD has to be inserted: it can be pasted, uploaded, or fetched via a URI. This TD is then parsed and all available Properties, Actions, and Events with according input fields/ drop-downs are generated. Interactions can then be selected and invoked. If applicable, security credentials, e.g. user-password combinations or broker data, can be stored beforehand. Then, a request to the associated endpoint will be sent. Simultaneously, the internal measurement is started and stopped as soon as W-ADE receives the Thing's response. The elapsed time in milliseconds and if available, the size of the received data in bytes will be shown. The according user interface is illustrated in Fig. 2.

W-ADE provides the framework for timing performance testing, eases the work with TDs and due to its plugin architecture, is able to improve tooling around the WoT ecosystem. Nonetheless, to give reliable and reusable assertions on timing performance we introduce a more elaborate timing analysis technique for timing performance benchmarking in Sec. 4.

4 Automated Timing Performance Benchmarking

W-ADE aims to facilitate the automated generation of timing performance benchmarks of a Thing with merely its TD available; Premised that the specific device is WoT-enabled and consumable. We define the scope of timing performance in Sec. 4.1, elaborate our benchmarking technique in Sec. 4.2 and present its implementation in Sec. 4.3.

4.1 Timing Performance Possibilities in W-ADE

The time it takes to communicate with a Thing is dependent on factors like connection throughput, available bandwidth, network workload, loss rate, software characteristics, hardware architecture, latency of outgoing packets, packet size or used communication protocols [12, 13]. To measure timing performance, common approaches are measuring the latency of outgoing packets or the throughput rate, while tweaking network conditions, manipulating bandwidth, or switching protocols [3, 12, 14]. In test scenarios, it is feasible to vary system configurations and retrieve data on the network environment or target devices. However, in real-world scenarios, mashup designers do not necessarily have access to manipulate devices or the opportunity to analyze internal application processes. Things could be connected via gateways and data on network conditions might be missing. As our objective is the facilitation of automated timing performance benchmarking on any capable machine, in any network environment, without knowledge about the target application, and while using any IoT-protocol, network-, protocol- and machine influences are not considered individually.

For this reason, W-ADE's timing performance technique is based on measurements of the overall round trip delay time (RTD), including the required processing time of the Consumer. The RTD is the sum of latency in each direction including the Thing's processing time. Latency indicates the total delay

between endpoints [13]. In our case this relates to the total time elapsed, from the moment the data is sent until the response is received, expressed by Eq. 1.

$$T_{\text{dynamic}}(x) = T_{\text{consumer}}(x) + T_{\text{transfer}}(x) + T_{\text{process}}(x) \tag{1}$$

- $\mathbf{T_{dynamic}}$: Total *dynamic* time needed for transferring a message from the Consumer to a Thing and receiving a response for an interaction x.
- $\mathbf{T_{consumer}}$: Consumer-side (in our case W-ADE's) processing time.
- $\mathbf{T_{transfer}}$: Time needed for sending/ receiving messages over the network.
- $\mathbf{T_{process}}$: Internal process time of a target Thing. It depends on the path of the internal execution and the time spent in the instructions on this path on this particular hardware [15]. Also covers physical interaction times.

Although the influences of $T_{consumer}$ and $T_{transfer}$ cannot be isolated reliably, it is possible under certain conditions to extract the static timing performance of an Action (a time-consuming physicial or virtual internal process). The IoT device must offer a Property-read as well as an Action interaction; Interactions must be implemented synchronously (not asynchronously or queued: responses are only send after a physical or internal process is finished)[9]; The application logic is comparable for all existing interactions; The time for reading a Property value is negligible (e.g. due to retrieving from memory); $\mathrm{T}_{\text{dynamic}}$ of the Action is bigger than $\mathrm{T}_{\text{dynamic}}$ of the Property-read. With these fullfilled prerequisities, we make the following assumption:

$$T_{\text{static}}(x) = T_{\text{dynamic}}(x) - T_{\text{dynamic}}(y) \tag{2}$$

$\mathrm{T}_{\text{static}}(\mathrm{x})$ is the estimated *static* time of an Action x that does not change with network or client alternations. $\mathrm{T}_{\text{dynamic}}(\mathrm{x})$ defines the *dynamic* time of the invoked Action x, $\mathrm{T}_{\text{dynamic}}(\mathrm{y})$ the same for a Property-read y.

4.2 Benchmarking Technique

To provide meaningful timing performance benchmarks, timing constraints on measurements and reliable average timing values need to be determined. For this purpose, bounds on execution times have to be identified. This can be achieved by executing an interaction for several repetitions or a specific amount of time, while simultaneously measuring the elapsed time. Results are combined to estimate execution time bounds. These are specified by deriving the overall maximum and minimum observed execution time. This is commonly called worst-case execution time (WCET) and best-case execution time (BCET) [15]. Generally, the BCET is overestimated and the WCET underestimated, as the actual values are almost impossible to derive. Since we cannot guarantee that Things can be analyzed, they are treated as *black-box* components and estimated WCETs are utilized. Moreover, the average execution time (AET) for all measurements is computed.

[9] It is assumed that interactions of a WoT-enabled Thing are mostly implemented to be synchronous, as conveyed in the WoT TD implementation report (https://w3c.github.io/wot-thing-description/testing/report.html). In future versions of the TD, information on interaction-implementation will be included.

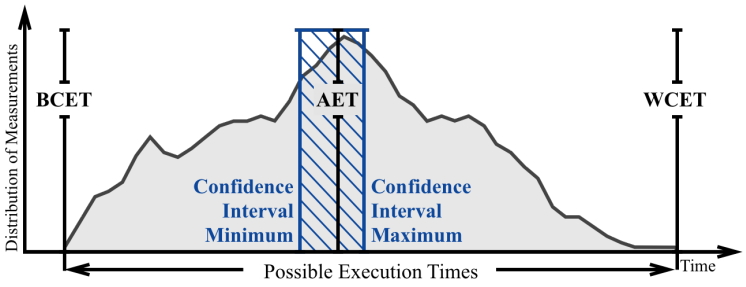

Fig. 3. Basic concept of our timing benchmarking technique. The curve depicts sample values of interaction timing performance. Its minimum indicates the estimated best-case execution time, its maximum the estimated worst-case execution time. The middle line represents the Average Execution Time, surrounded by its confidence interval.

In order to offer reliable benchmarks, we use confidence intervals (CI)[10] to propose a range of plausible values for the AET. The CI is calculated as seen in Eq. 3, where $\overline{\mathrm{x}}$ is the sample mean, σ the standard deviation, n the sample size and z^* represents the appropriate z^*-value from the standard normal distribution of the chosen Confidence Level (CL). The CL indicates the probability that the unknown parameter lies in the stated interval.

$$\overline{\mathrm{x}} \pm z^* \frac{\sigma}{\sqrt{n}} \tag{3}$$

To compute *static* timing, measurements of both a Property and Action are required. The *static* AET is computed from the difference of the *dynamic* AET values of the included Action and Property measurements (see Eq. 2). Its CI is calculated with Eq. 4: Elements have the same meaning as in Eq. 3, whereby subscript a represents Action and subscript p Property values.

$$\overline{\mathrm{x}}_\mathrm{a} - \overline{\mathrm{x}}_\mathrm{p} \pm z^* \sqrt{\frac{{\sigma^2}_\mathrm{a}}{n_\mathrm{a}} + \frac{{\sigma^2}_\mathrm{p}}{n_\mathrm{p}}} \tag{4}$$

Certain factors influence execution times and need to be considered when timing performance is measured and interpreted. One factor is the context dependence of execution times [15]. The execution time of individual instructions may vary by several orders of magnitude depending on the state of the processor. Thus, an execution time B can heavily depend on the state produced by execution A, e.g. for initial connection establishment, regarding memory or caching [15]. A task can also show variations depending on the payload or divergent environment behavior. To minimize this impact, the option of using measurements after a certain time has passed and an indefinite amount of interactions have been executed, is available and measurements can be repeated multiple times to reduce context errors. To remove the impact of initial connection establishment, the

[10] A CI, in statistics, refers to the probability that an unknown value will fall between a specific range of values, calculated from observed data [16].

first measured values are removed. Further, timing anomalies, counter-intuitive influences on the local execution time of one instruction on the global execution time of the entire task [15], need to be considered and optionally be removed. We use the common approach of detecting outliers with the help of the interquartile range (IQR) that represents the width of the box in the box-and-whisker plot [17] and indicates how spread out middle values are, shown in Eq. 5.

$$IQR = Q_3 - Q_1 \tag{5}$$

$$min = Q_1 - 1.5 \times IQR \quad \text{and} \quad max = Q_3 + 1.5 \times IQR \tag{6}$$

The IQR is the difference of the third (Q_3) and the first quartile (Q_1). A Quartile divides the number of data points into four equal parts, or quarters. Q_1 marks where 25%, Q_3 where 75% of the data is below. Eq. 6 defines how the minimum and maximum threshold can be computed to identify outliers. Outliers are defined as values that are more than one and a half times the length of the middle-value box, away. Identified outliers can then be removed.

4.3 Implementation

To put this technique into practice, a timing performance feature is implemented in W-ADE. Users need to select interactions, enter required inputs, and choose performance analysis settings, including the type of measurement - either *Iteration* or *Duration* - with the desired amount. Iteration indicates that measurements are executed a certain number of times, whereas Duration executes them for the entered time-period. Then, a delay before the beginning of overall measurements or before the beginning of each execution can be scheduled. To finally start the execution and computation process, the desired CL to calculate the AET's CI has to be entered. Selectable options are 80%, 85%, 90%, 95%, 99%, 99.5% and 99.9%. To take into account potential environmental impacts, results are subdivided into *Possible* and *Realistic*. They respectively contain computed WCET, BCET, AET, CI limits for the AET, and the first measurement value. *Possible* results are computed considering all measurements except the first measured, whereas for *Realistic* results neither the first measured, nor detected outliers are included. Additionally, settings, general information, and all measured values in chronological order are displayed. To compute *static* timing, one Property, one Action, and the option `static timing` must be selected. As WCET and BCET are not applicable, they are not present in *static* results.

5 Timing Performance Annotation

To be able to apply timing performance benchmarks, results needs to be integrated into the TD, in reference to the appropriate interaction. For this purpose, we propose a vocabulary set, referred to as *InteractionTiming*, that describes timing performance benchmarks. It is compatible with present TD vocabularies, as it is based upon similar semantics and principles. Existing TD vocabularies are independent and extensible. They each define a collection of terms that can be interpreted as objects denoting Things and their Interaction Affordances [5].

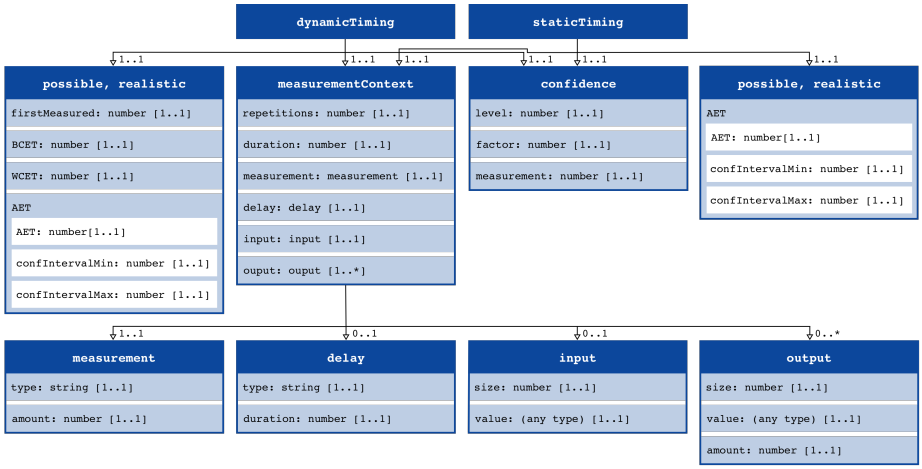

Fig. 4. Our proposed *InteractionTiming* vocabulary with term keys and values, including child elements and their data types. It is intended for annotating a Thing Description, in order to characterize timing performance of its associated Thing.

As not all collected performance data discussed in Sec. 4 is significant for describing timing performance, only specific elements are included in our vocabulary. Fig. 4 gives an overview of the main *InteractionTiming* vocabulary terms. The proposed term names are only suggestions and can be easily adjusted if they conflict with other TD terms. `staticTiming` characterizes *static* timing measurements and is added to the TD on the same level as `forms`. Other measurements are respectively summarized in `dynamicTiming` and are added to the particular `forms` element of the interaction. By default, both of them include `measurementContext` information, `confidence` data, and results categorized in `possible` and `realistic`. As `measurementContext` implies only contextual data, it is linked to by a JSON Pointer [18] and can be stored in another document. This default term set was elaborated with the aim of providing most information without including the entire set of results. An example of an annotated TD comprising *dynamic* and *static* performance data is depicted in List. 5.1. For the sake of readability, only *Possible* measurement data is added and confidence information of the *dynamic* annotation is absent. We provide a complete annotated TD[11] externally[12]. As a TD can be presented as a JSON-LD file, its specification [5] also provides a JSON Schema [19] to syntactically validate TDs. To extend this Schema, we created an *InteractionTiming* JSON Schema, including detailed descriptions[13] and a TD JSON Schema extended with it[14]. An annotated TD can be generated in W-ADE after results are computed. If desired, annotations can be further revised in the TD editor, before saving. An

[11] Available at `www.ei.tum.de/fileadmin/tueifei/esi/WADE-files/td_annotated.json`
[12] The distribution of annotated TDs is not determined in this paper, since not even the W3C WoT working group has fully explored TD distribution possibilities so far.
[13] `www.ei.tum.de/fileadmin/tueifei/esi/WADE-files/Interaction_Timing_schema.json`
[14] `www.ei.tum.de/fileadmin/tueifei/esi/WADE-files/TD_Interaction_Timing_schema.json`

enhanced TD can be interpreted by a human and an automated system. It offers detailed semantics for describing how a Thing behaves regarding timing.

```
"brew": {
  "staticTiming": {
     "possible": {
        "AET": {
           "AET": 40078.83,
           "confIntervalMin": 40072.32, "confIntervalMax": 40085.35
  }  }  },
  "forms": [{
        "href": "actions/brew",
        "dynamicTiming": {
           "measurementContext": "#/measurementContext/
               dynamicTimingContext_action/brew",
           "possible": {
               "firstMeasured": 40221.06,
               "BCET": 40118.66, "WCET": 40962.02,
               "AET": {
                   "AET": 40238.33,
                   "confIntervalMin": 40185.93, "confIntervalMax": 40290.74
           }   },
           "confidence": {
               "level": 99.9, "factor": 3.291, "numMeasurements": 100
}  }  }  ]  }
```

Listing 5.1. A snippet of an annotated Thing Description, including *dynamic* and *static* timing data for the Action `brew`. *Dynamic* annotations are added to the particular `form` element, whereby *static* annotations are located on the same level as `forms`.

6 Evaluation

To prove the correctness and validate the quality of our technique, we conducted three virtual experiments and one use-case scenario with a physical IoT device.

Experiment 1, Validity Test: To validate W-ADE's timing performance functionality and to show that processed results are consistent, a Property-read that returns a 14 byte string of an externally hosted TD and its simulated Things[15] was selected. Then, a HTTP-request was executed with a chosen CL of 99.9% for 1000 iterations, 10 times. Different network conditions applied for the Consumer and simulated Thing. We then examined how results differed, to what percentage the CI fluctuates around the AET and verified whether AETs are in the range of other CIs. Fig. 7 shows that computed AETs and their corresponding CI limits are consistent for *Possible*, as well as *Realistic* results. Only some negligible deviation, with a max. range of 10ms in *Possible* and 4ms in *Realistic* average AET values, were observed. Moreover, CI limits always lied within a range of $+/-5\%$ of AETs and AETs values lied in their associated CI, including all other measured CIs.

Experiment 2, Sanity Check: To evaluate W-ADE's credibility, we matched our measurements with measurements of an ADE[16] called *Postman*[17]. It acted

[15] This TD is provided by the W3C WoT working group for testing purposes.
[16] ADE stands for API Development Environment and describes software that focuses on designing, building, and testing APIs.
[17] Postman (`www.getpostman.com/`) version v7.16.1. was utilized.

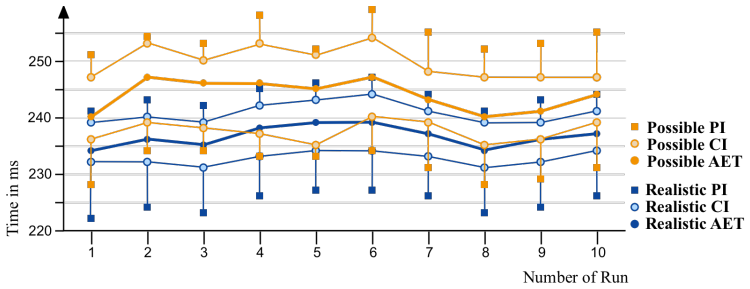

Fig. 5. W-ADE's measured, rounded *Realistic* (blue) and *Possible* (orange) average execution times (AET). Confidence intervals (CI) always lied within a 5% range (PI) around the average values. AETs always lied in their own and all other CIs.

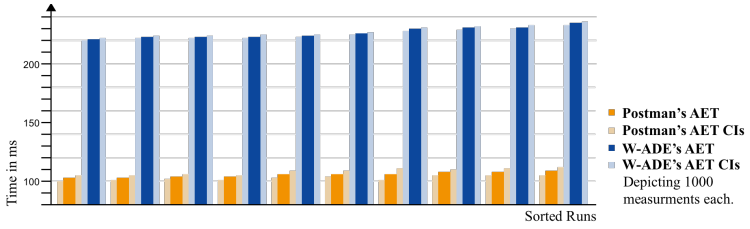

Fig. 6. Measured average execution times of W-ADE (blue) are compared to Postman's (yellow). Bars denote the average values for 1000 measurements each. Enclosing thinner bars respectively indicate confidence interval limits. Both systems produced consistent values, whereas W-ADE's expected overhead to Postman was also consistent.

as a control entity. In comparison to W-ADE, Postman is not able to parse or understand TDs. It can only communicate over HTTP and lacks functionalities to interpret and produce timing performance benchmarks. Using the same test Thing from Exp. 1, we executed an HTTP Property-write request with a 12 byte string-input, no output, and a CL of 99.9% for 1000 times, repeating it 10 times with W-ADE and Postman. To make result sets comparable, we applied W-ADE's technique onto Postman's results to processed the WCET, BCET, and AET with its CI. We used W-ADE's *Possible* results, as outliers were not removed from Postman, and rounded them to integers.

Results, presented in Fig. 6, show that Postman's and W-ADE's results are proportional to each other and consistent within. WADE's AET values showed a max. deviation of $5,98\%$ (min. 221ms, max. 235ms). Postman's values showed a similar max. deviation of $6,42\%$ (min. 103ms, max. 109ms). W-ADE added an average overhead (see $T_{client}(x)$ in Eq. 1) of about 121ms and $111,01\%$ in comparison to Postman. This is expected, as Postman's default behavior keeps socket connections open[18]. W-ADE closes them after each request as keep-alive connections are only usable for polling, which should rather be implemented as event properties. This is the anticipated way of writing a Consumer application.

[18] This behavior cannot be changed in the current version of Postman.

Experiment 3, Static Timing: To confirm the validity of our *static* timing approach, introduced in Sec. 4.1, we created a script that simulates a Thing, providing a Property-read that returns a 14 byte string and an Action, that returns the same variable after a predetermined delay of 1000ms, simulating a physical process of a device. It was then exposed on the same machine running W-ADE, communicating over the same local network. 100 Action and Property HTTP requests were sent to the before-mentioned interaction with a chosen CL of $99,9\%$ and measured with W-ADE, 10 times each. Due to network anomalies and other outliers adding a noticeable effect to timing, we used rounded *Realistic* values for our evaluation. As shown in Fig. 7, the computed average *static* AET of 999,30ms matched the artificial delay of 1000ms to $99,93\%$, whereby *Realistic* AET measurements resulted either 999ms or 1000ms. The min. lower limit of the CI was 998ms and the max. 1001ms. Giving a $-0,2\%/+0,1\%$ range around the actual delay. This confirms that that computed results are able to anticipate actual interaction timing performance.

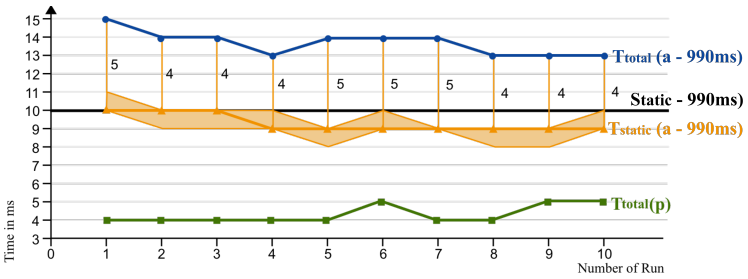

Fig. 7. Computed *static* timing performance of an Action. The upper curve indicates *dynamic* measurements (see Eq. 1) of Action a, the lower measurements of a Property p. The middle line shows the *static* timing with its confidence interval. For the sake of comprehension **990ms are subtracted from both Action values**.

Use-Case with a Physical Device: To demonstrate the practicability of timing performance generation with a real Thing, we conducted *static* measurements analog to Exp. 3. Included components were W-ADE as client and a WoT-enabled[19] Pan-Tilt HAT module[20] (PTH) attached to a Raspberry Pi[21] (RP), both connected to the same network. The evaluated Action `scan` moves the robot arm from the outer-left to the outer-right position and the used Property `panPosition` returns the position of the pan module. Both HTTP interactions were executed 10 times with 1000 iterations each and a $99,9\%$ CL, before the *static* timing of `scan` was calculated. Analog to Exp. 3, the evaluation was based on *Realistic* results. An average *static* AET of 40055ms with a CI of 40053ms - 40057ms was measured for `scan`. Whereby the average *dynamic* AET of all measured Actions was 40063ms, with a CI of 40049ms - 40077ms. Property measurements revealed an average AET of 8ms and CI limits of 7,6ms - 8,8ms.

[19] A WoT-enabled Pan-Tilt HAT: `www.wotify.org/library/Pan-Tilt\%20HAT/general`.
[20] A set of horizontal and vertical motion servos, that can be moved individually.
[21] A small single-board computer, here a model `3B+` running `Raspbian 2019-09-26`.

This proves that W-ADE enables the conduction of *static* timing performance on an actual physical IoT device. To once again validate that the *static* AET matches the actual time of a `scan` movement, manual, possibly error-sensitive chronometer based measurements, would be necessary.

7 Related Work

Numerous approaches on how the Web of Things can look, have been introduced to the world of IoT [20–22]. As the TD standard and associated WoT approaches not only present a well conceived concept, but also actively offer solutions to counteract the fragmentation of IoT, this work is based upon them. Many W3C WoT tools and services have already been contributed, nevertheless, there is little scientific work on them available. This might be due to the TD standard being rather new and hardly distributed. Therefore, no service or approach to enhance a TD with timing information has been released.

While performance evaluation in IoT is the topic of many studies, they often focus on comparison of diverse protocol performance under specific circumstances, whereby the experiment environment is mostly controlled [23, 14, 3]. Other studies target network-performance regarding an IoT device [22, 24], the general performance spectrum, or stress testing of Things [12]. In general, studies in the field of IoT performance do not deal with *dynamic* or *static* timing performance of Things and do not offer solutions on how to generate comparable benchmarks for this purpose. Choosing the best fitting communication protocol and determining network performance makes sense when setting up IoT platforms, but not when third-party Things need to be integrated into mashups or their general timing performance needs to be benchmarked. In contrary to existing studies, our proposed technique enables developers to actively use timing performance data for the design of real-world IoT systems and use cases.

8 Conclusion

Motivated by the problem of missing opportunities to easily measure and compare timing performance of IoT devices based on their TD, we introduced a technique which facilitates timing performance benchmarking, while considering environmental influences. Thereupon, we developed W-ADE, an API development environment and platform for the WoT ecosystem that implements our technique and additionally enables manual timing measurement of device interactions. To validate and demonstrate the applicability in practice, we tested our technique with a physical IoT device and conducted virtual simulations. We proved that W-ADE reliably predicts *static* timing performance of interactions and offers accurate timing performance benchmarks. Combined with our proposed *InteractionTiming* vocabulary to annotate TDs, mashup designers are now able to estimate and compare interaction timing performance; thus, optimize system compositions during design time.

References

1. D. Guinard and V. Trifa. *Building the Web of Things: With Examples in Node.js and Raspberry Pi*. Manning Publications Co., 2016.
2. D. Guinard, V. Trifa, T. Pham, and O. Liechti. Towards Physical Mashups in the Web of Things. In *Proc. of INSS*, volume 9, pages 17–19, 2009.

3. Z. B. Babovic, J. Protic, and V. Milutinovic. Web Performance Evaluation for Internet of Things Applications. *IEEE Access*, 4:6974–6992, 2016.
4. D. Guinard and V. Trifa. Towards the Web of Things: Web Mashups for Embedded Devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in Proc. of WWW*, volume 15, 2009.
5. T. Kamiya, S. Käbisch, M. Kovatsch, M. McCool, and V. Charpenay. WoT Thing Description. Technical report, W3C, 2019. www.w3.org/TR/2019/CR-wot-thing-description-20191106/.
6. R. Matsukura, M. Lagally, M. Kovatsch, K. Toumura, and T. Kawaguchi. WoT Architecture. Technical report, W3C, 2019. www.w3.org/TR/2019/CR-wot-architecture-20191106/.
7. D. Peintner, K. Nimura, Z. Kis, J. Hund, and K. Nimura. WoT Scripting API. Technical report, W3C, 2019. www.w3.org/TR/2019/WD-wot-scripting-api-20191028/.
8. M. Koster. WoT Protocol Binding Templates. Technical report, W3C, 2018. www.w3.org/TR/2018/NOTE-wot-binding-templates-20180405/.
9. P. Champin, G. Kellogg, and D. Longley. JSON-LD 1.1. Technical report, W3C, 2019. https://www.w3.org/TR/2019/CR-json-ld11-20191212/.
10. E. Korkan, S. Kaebisch, M. Kovatsch, and S. Steinhorst. Safe Interoperability for Web of Things Devices and Systems. In *Languages, Design Methods, and Tools for Electronic System Design*, pages 47–69. Springer, 2020.
11. X. Liu, Y. Hui, W. Sun, and H. Liang. Towards Service Composition Based on Mashup. In *2007 IEEE Congress on Services*, pages 332–339. IEEE, 2007.
12. J. Esquiagola, L. C. de Paula Costa, P. Calcina, G. Fedrecheski, and M. Zuffo. Performance Testing of an Internet of Things Platform. In *IoTBDS*, pages 309–314, 2017.
13. G. Huston. Measuring IP Network Performance. *The Internet Protocol Journal*, 6(1):2–19, 2003.
14. Y. Chen and T. Kunz. Performance Evaluation of IoT Protocols under a Constrained Wireless Access Network. In *2016 Int. Conf. on Selected Topics in MoWNeT)*, pages 1–7. IEEE, 2016.
15. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al. The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008.
16. J. Neyman. X—outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 236(767):333–380, 1937.
17. J. W. Tukey. *Exploratory Data Analysis*, volume 2. Reading, Mass., 1977.
18. P. Bryan, K. Zyp, and M. Nottingham. JavaScript Object Notation (JSON) Pointer. *RFC 6901 (Proposed Standard)*, 2013.
19. A. Wright, H. Andrews, and G. Luff. JSON Schema Validation: A Vocabulary for Structural Validation of JSON. *IETF Standard*, 2016.
20. D. Guinard, V. Trifa, M. Friedemann, and E. Wilde. From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. In *Architecting the Internet of Things*, pages 97–129. Springer, 2011.
21. T. Käfer, S. Richard Bader, L. Heling, R. Manke, and A. Harth. Exposing Internet of Things Devices via REST and Linked Data Interfaces. In *Proc. 2nd Workshop Semantic Web Technol. Internet Things*, pages 1–14, 2017.
22. S. Duquennoy, G. Grimaud, and J. Vandewalle. The Web of Things: Interconnecting Devices with High Usability and Performance. In *2009 Int. Conf. on Embedded Software and Systems*, pages 323–330. IEEE, 2009.
23. T. Yokotani and Y. Sasaki. Comparison with HTTP and MQTT on Required Network Resources for IoT. In *2016 Int. Conf. on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pages 1–6. IEEE, 2016.
24. R. Morabito, I. Farris, A. Iera, and T. Taleb. Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge. *IEEE Internet of Things Journal*, 4(4):1019–1030, 2017.