Article

A WoT-based method for creating Digital Sentinel Twins of IoT devices

Ivan Lopez-Arevalo [1,‡], J.L. Gonzalez-Compean [1,*,‡], Mariana Hinojosa-Tijerina [1,‡], Cristhian Martinez-Rendon [2,‡], Raffaele Montella [3,‡], Jose L. Martinez-Rodriguez [4,‡]

1 Cinvestav Tamaulipas, Mexico; {ilopez, joseluis.gonzalez, mariana.hinojosa}@cinvestav.mx
2 University Carlos III of Madrid, Spain; cristhma@pa.uc3m.es
3 University of Naples, Italy; raffaele.montella@uniparthenope.it
4 Reynosa Rodhe Multidisciplinary Academic Unit, Autonomous University of Tamaulipas, Mexico; lazaro.martinez@uat.edu.mx
* Correspondence: joseluis.gonzalez@cinvestav.mx
‡ These authors contributed equally to this work.

Abstract: The data produced by sensors of IoT devices are becoming keystones for organizations to conduct critical decision-making processes. However, delivering information to these processes in real-time represents two challenges for the organizations: the first one is achieving a constant dataflow from IoT to the cloud and the second one is enabling decision-making processes to retrieve data from dataflows in real-time. This paper presents a cloud-based Web of Things method for creating digital twins of IoT devices (named *sentinels*). The novelty of the proposed approach is that sentinels create an abstract window for decision-making processes to: a) get data (e.g. properties, events, and data from sensors of IoT devices) or b) invoke functions (e.g. actions and tasks) from physical devices (PD) as well as from virtual devices (VD). In this approach, the applications/services of decision-making processes deal with sentinels instead of managing complex details associated with the PDs, VDs, and cloud computing infrastructures. A prototype based on the proposed method was implemented to conduct a case study based on a blockchain system for verifying contract violation in sensors used in product transportation logistics. The evaluation showed the effectiveness of sentinels enabling organizations to get data from IoT sensors and the dataflows used by decision-making processes to convert these data into useful information.

Keywords: Digital twins; IoT data; Microservices; Cloud Computing; Web of Things; Virtual Containers.

Citation: Lopez-Arevalo, I.; J.L. Gonzalez-Compean; M. Hinojosa-Tijerina; C. Martinez-Rendon; R. Montella, and J.L. Martinez-Rodriguez A WoT-based method for creating Digital Sentinel Twins of IoT devices. *Sensors* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

IoT devices are becoming a key element in decision-making processes [1], [2], [3]. These devices are quite common in multiple infrastructures such as Industry 4.0 [4], healthcare domain [5], and supply chains [6], to name a few. The data produced by these devices follow a lifecycle from the sensors to the edge [7], to the fog [4] and to the cloud [8]. In this lifecycle, data are acquired (mainly at the edge [9]), prepared and analyzed (typically at the fog and/or the cloud [10]), and finally converted into information for human consumption to use it in decision-making processes (mainly at the cloud [8] through end-user devices). In these types of infrastructures (any combination of edge, fog or cloud), the virtual containers (VC) are key to deploy services on each infrastructure [11–13]. These services provide dataflows from the IoT to the cloud that produce different types of data and information, which proves to be key for organizations to conduct critical decision-making processes [14–16].

However, extracting data/information from these dataflows to deliver it to decision-making processes in real-time represents a huge challenge in two directions: the first one is verifying the accomplishment of a constant dataflow from IoT to the cloud; and the second one is enabling decision-making processes to retrieve, in real-time, data/information from different points of dataflows. These data acquisition tasks through dataflows are not straightforward because of the heterogeneity of the components participating in a dataflow (applications, types of sensors, data formats, infrastructures [17], to name a few). It is desirable a manner not just to acquire data/information from dataflows, but also to invoke actions and tasks on the dataflow components. That could facilitate tasks on decision-making analysis.

We propose to create *digital twins* of the IoT data acquirers (hardware, physical machine, or virtual container -application/microservice-) by using Web of Things cards (WoT)[1] for decision-making process to retrieve, in real time, data/information or invoke actions/tasks. A *digital twin* is an abstract representation commonly used in Industry 4.0 for IoT device monitoring [18]; that is, a virtual replica of objects/processes that simulate the behavior of their real counterparts. WoT is an initiative for representing and managing definitions of IoT artifacts (devices, components, applications, etc.), which suggests using a set of well-accepted protocols from the Semantic Web for any IoT artifact from the physical world to be available into the World Wide Web by creating a net of WoT definitions [19].

In this paper, we present the design, implementation and evaluation of a cloud-based WoT method for creating *Digital Sentinel Twins* (*DST*) of IoT devices. A DST creates an abstract window for decision-making processes to get information/data such as properties, events, and data produced by sensors, and to invoke actions/tasks from IoT devices. An IoT device is a *physical device* (*PD*) with sensors and tasks that can be accessed directly or through a *virtual device* (VD[2]). Figure 1 shows an example of the process used by a DST to create a window for decision-making processes consumption (by either a human, application, or VD). As it can be seen, in this approach, the applications/services used in decision-making processes deal with DSTs instead of managing the complex details associated with the PDs, VDs, or cloud computing infrastructures.

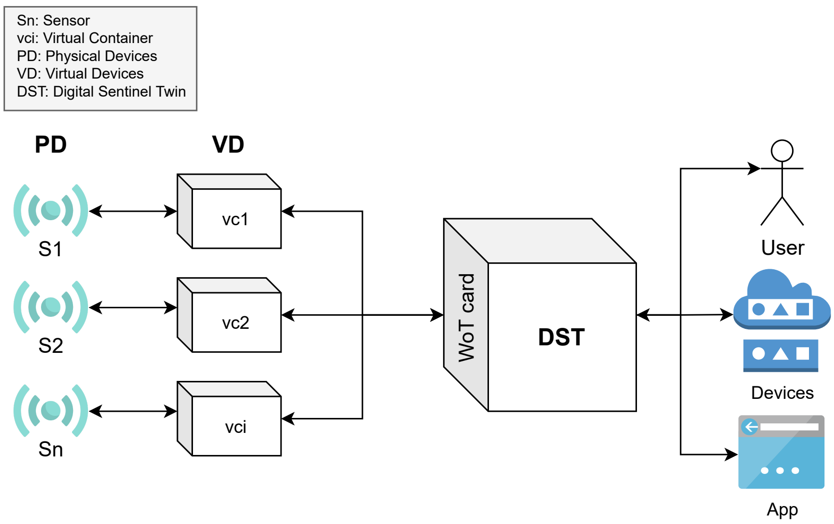

Figure 1. Conceptual view of a DST.

We implemented a prototype based on this method to perform case studies supported by GPS, temperature, and speed sensors. Additionally, using a blockchain system, the compliance of contracts to which these sensors are subject in the transportation logistics of products is continuously verified. The evaluation revealed the effectiveness of the DSTs for organizations to get data/information about both IoT devices and the

[1] www.w3.org/WoT [All web pages in this paper were visited on June 22, 2021].
[2] A VD is an application/microservice encapsulated into a virtual container for acquiring, extracting, processing, monitoring, and analyzing data from PDs.

whole processes converting IoT data into useful information required in decision-making processes.

The contributions of this work are:

- The design, implementation, and evaluation of a cloud-based WoT method for creating Digital Sentinel Twins of IoT devices.
- The definition of the Digital Sentinel Twin concept as a mean for accessing data/information and for invoking tasks from IoT devices.

The rest of the paper is organized as follows: Section 2 describes the state of the art of works related to the topics of the proposed method; Section 3 describes the design and construction of a method to create DSTs for interacting with IoT devices; Section 4 describes the implementation of a prototype for the creation of DSTs; Section 5 presents the results of the prototype in two phases of experiments; the discussion of the obtained results is described in Section 6. Finally, Section 7 is presented with conclusions and future work.

2. Related Work

In the literature, there are some works about Digital Twins that are relevant to our approach, and these are next described.

In the context of Digital Twins, there are different works focused on its use for simulation, monitoring, risk prevention, etc., for IoT devices. Some are [20], [21] and [22]. In [20] Assad *et al.* proposed a Web-based Digital Twin (WDT) architecture, with the purpose of improving the sustainability of industrial cyber-physical systems. In [21] Bevilacqua *et al.* proposed a Digital Twin reference model for risk prediction and prevention. The difference between our work and these two proposals is that we establish the use of virtual containers in a middle layer to access, acquire, extract, transform, etc. the information of the IoT devices; in this way, through a *DST*, we are able to represent both the physical (the IoT artifact) and virtual (software applications accessing the IoT artifact) device. In [22] *Gao et al.* proposed a method of simulation and modeling in real time for the production line of digital twins. The effectiveness of the proposed method is verified by taking an assembly line as an example.

In the context of Digital Twins using virtual containers for the acquisition of information from IoT devices, the proposals [23], [24] and [25] are interesting. In [23] Alaasam *et al.* proposed a study on live stateful stream processing migration of Digital Twins. The authors emphasized the importance of two factors that influence the construction of stateful stream processing in systems as complex as Digital Twin: Stateful virtualization infrastructure and the stateful data. In [24] Tingyu *et al.* proposed a methodology of container virtualization based on simulation as a service (CVSimaaS), the authors use virtual containers to implement a Digital Twins system, obtaining a lower consumption of resources with high efficiency. Like our proposal, these two works include the concept of virtual containers together with Digital Twins for IoT devices. However, these two proposals do not add a standardized representation to the Digital Twin. Moreover, in our proposal, we follow the WoT guidelines for the creation of the *DST* as universal accessible entities. In [25] Borodulin *et al.* proposed a model for simulation and prediction of industrial processes using Digital Twins in Digital Twin-as-a-Service (DTaaS), which is a way to implement an orchestration of a set of independent services and provide scalability for simulation.

In the context of virtual container modeling, two proposals stand out [26,27]. In [26] Paraiso *et al.* presented an approach to model-driven management of Docker containers, which enables verification of the virtual container system architecture at design time. In [27] Piraghaj *et al.* proposed a simulation architecture called *ContainerCloudSim*, which was used to evaluate resource management techniques in virtual containers from cloud environments. Unlike these proposals, whose focus is only on virtual containers modeling, our proposal additionally models the environment of the IoT devices, adding WoT recommendations for representing them, which produces a *DST* flexible for consumption of the virtual containers and IoT devices data. In [28] Medel *et al.* proposed a performance model for Kubernetes-based deployment using Docker containers. Such a model can be used as a basis to support resource management and application design.

In the context of the use of virtual containers for the monitoring, simulation, and orchestration of IoT devices, there are two proposals [29] and [30]. In [29] Alam *et al.* proposed a modular and scalable architecture for IoT based on lightweight virtualization. Thus, the modularity provided, combined with the orchestration provided by Docker, simplifies management and enables distributed deployments, creating a highly dynamic system. In [30] Muralidharan *et al.* proposed a distributed monitoring system based on virtual containers for IoT applications for the management of a smart city environment. They achieved low latency, reliable and secure communication between large-scale deployment of IoT devices, with a focus on horizontal interoperability between various IoT applications. Both works do not use the Digital Twin concept, unlike our work (DST), which allows us to create a reflection with the properties and characteristics of the IoT device.

Muralidharan *et al.* in [31] proposed a semantic Digital Twin model for interacting with IoT devices. The authors used virtual containers to mimic IoT devices. This is the most similar approach to our proposal. However, they only focus on modeling the physical devices (PD) and not virtual devices (VD). Instead, through the DST, we can represent both the physical and virtual devices.

3. On the building of Digital Sentinel Twins for IoT devices

A *Digital Sentinel Twin* (DST) is a software object produced from a data structure named $WoTcard$, which is created from data of *Physical Devices* (PD) or *Virtual Devices* (VD) interacting with surrounding elements for accomplishing some task involving sensors.

The conceptualization of a DST is illustrated in Figure 2, which is composed of the concepts next described.

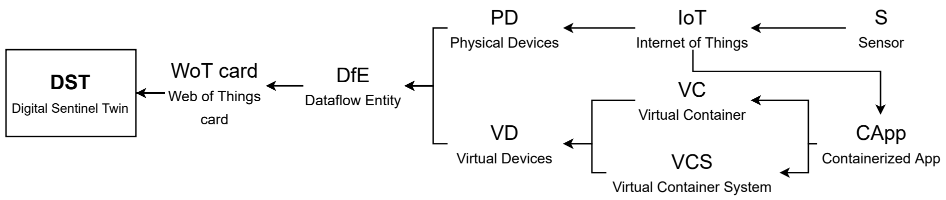

Figure 2. Conceptualization of a DST.

A PD represents an IoT device interacting with sensors. A VD represents the software components required for creating a dataflow from an IoT device to a decision-making process. This means that a VD comprises components such as *Virtual Containers* (vc) or a *Virtual Container System* (VCS). A vc is a mechanism for logical encapsulation of software applications that creates environment independent applications required to create a dataflow. A VCS represents a set of vc_i built as a single solution (service) to perform a task into the dataflow. A *Containerized Application* ($CApp$) is in charge of interacting with IoT devices, and it is encapsulated into either vc or VCS.

Thus, a DST is a versatile object for interacting in an easy manner with the complex and detailed structure of PD or VD. This is due to the flexibility of the $WoTcard$, which fulfills the recommendations of the $W3C$[3]. This information comes from a *Dataflow Entity* (DfE), which captures information of each internal component (any of $CApp \in vc$, $vc_i \in VCS$, or PD) as well as relationships of these components with the PD. The DfE is basically a data structure including information about the structure, behavior, and

[3] www.w3.org

function of VD or PD. The structure, behavior, and function are used to model the dataflow from the IoT device to the decision-making processes (as it captures these features of all entities participating in such a dataflow). The context of generation and usage of a DST is illustrated in Figure 3.

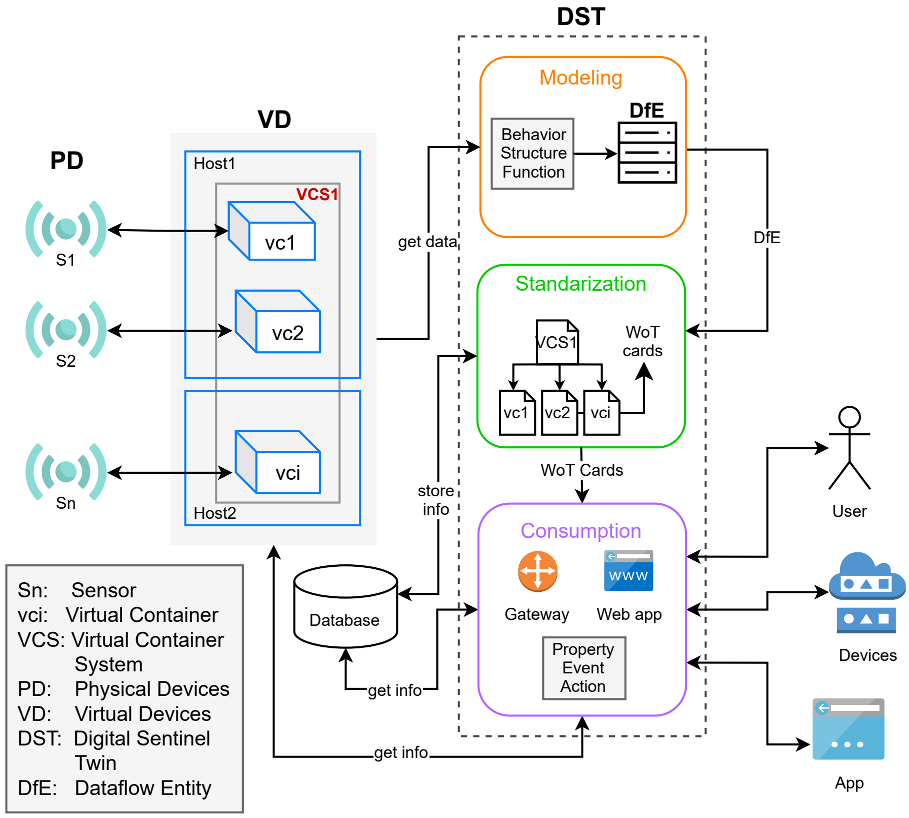

Figure 3. Context of a DST.

We considered an additional layer for standardizing the representation of a DfE by using WoT guidelines; this produces a $WoTcard$. That means, a $WoTcard$ represents the information of DfE through standardized concepts about virtual containers. These concepts come from an ontology based on the ISO norm ISO/IEC JTC 1/SC 38[4]. By following these WoT standards, a $WoTcard$ can represent, in a well-defined manner and unique identity, a VD or PD, without any further adaption on DfE.

We propose a three-phase method to create a DST for a dataflow from IoT devices to decision-making processes. The Figure 3 also shows the conceptual view of the stages of the methodology: *Modeling* (phase 1), where the data of the VD is acquired and its elements modeled; then, in the *Standardization* (phase 2), these elements are depicted into WoT cards, which are ready to be used in the *Consumption* (phase 3). Next, each stage is described more in detail.

3.1. Phase 1: On the usage of functional modeling for building DfE

A VD or PD can be modeled as a process to achieve a goal. The *functional modeling* [32] [33] is quite suitable for creating a representation of its structure, behavior, and function. This modeling has been used, over the past years, for successfully representing processes in multiple scenarios [34] [35] [36].

In the proposed method, all the dataflow participants are modeled as objects composed of low-level parts; the object has an objective, and its components contribute to achieve together such an objective by performing *tasks*, such as acquiring, manufacturing,

[4] www.iso.org/committee/601355.html

preparing, or analyzing data produced by IoT sensors. The functional modeling is quite suitable for the IoT context where it is important not only to model the IoT devices but also the dataflow participants to describe the properties, events, and actions performed from the IoT devices to the decision-making processes (either at the fog or cloud). This approach also allowed us to model all the participants in the production of these dataflows (any of vc_i, VCS, or $CApps$), which, in fact, are having a behavior of chained processes. This model is captured into DfE, which describes the behavior (properties and events), function (tasks), and structure (interconnections) of all participants in the dataflow.

As a preparation step of this method, we assume the existence of a vc_i (see VD in Figure 3) executing a transformation of data (*task*); independently of the number of internal vc_i in a dataflow, this are modeled as one DST. Lets us consider the simpler case, where one vc_i is decomposed into its function, structure, and behavior and stored in a DfE. This decomposition is represented by means of $WoTcards$, making the DfE as a DST ready for consumption. For the case of a VCS, occurs the same process by each individual vc_i, integrating individual functions as the overall function of the DST.

The objective of this phase is to obtain the three main modeling elements of a vc:

- *Structure*, where the components of the vc and its relationships are specified,
- *Behavior*, where the values of the attributes of components are specified, according to the function of the vc,
- *Function*, where the main goal of the vc and the tasks required to achieve it are specified.

This phase starts by receiving the configuration file of a vc, in YML or $YAML$ format. From this file all the data required to represent the vc is acquired.

Next the main elements are described following a decomposition approach.

Function

The function is the goal description of the vc. If the input file is of a VCS, the function is modeled as a composition of functions of the internal vc_i. The function makes reference to the task executed (*transformation*) on the dataflow. There are six base function for a vc:

- *source*, the capability to act as an infinite reservoir of data,
- *transport*, the capability to transfer data from one point to another, including from one medium to another,
- *barrier*, the capability to prevent the transfer data from one point to another, including from one medium to another,
- *storage*, the capability to accumulate data,
- *balance*, the capability to provide a balance between the total rates of incoming and outgoing dataflows,
- *sink*, the capability to act as an infinite drain of data.

Specialized functions can be derived from these base functions, such as *produce-data, acquire-data, integrate-data, consume*, to mention a few. All the functions may be connected to each other into flow paths or flow structures forming software structures.

Thus, each vc_i has at least one application (App_j) performing some *transformation* (tr_k); defined as follows.

$$VC = \{vc_1, vc_2, \ldots, vc_i\} \tag{1}$$

$$App = \{App_1, App_2, \ldots, App_j\} \tag{2}$$

$$Tr = \{tr_1, tr_2, \ldots, tr_k\} \tag{3}$$

$$\forall vc_i \in VC : vc_i \supset App_j \tag{4}$$

$$\forall App_j \in App : App_j \supset tr_k \tag{5}$$

The tr_k is the key element for representing the *function* of a vc_i.

A containerized application ($CApp$) represents one or a set of applications App_l, $l < j$, encapsulated into a vc_i.

$$CApp = \{App_1, App_2, \ldots, App_l\} \tag{6}$$

Structure
The internal *structure* of a vc is commonly organized as software structures (*e.g.* patterns, pipelines, parallel schema, dataflow, etc.). The model of the vc must reflect this kind of organization. Thus, the *structure* of the vc is defined as a logical directed acyclic graph DAG, where nodes (N) represent the components ($comp_i$) that compose the vc, while the interconnections between nodes ($comp_q \rightarrow comp_r$) are established by edges (E), which are defined as follows.

$$N = \{comp_1, comp_2, comp_3, comp_i\} \tag{7}$$

$$E = \{comp_1 \rightarrow comp_2, comp_2 \rightarrow comp_3, comp_{i-1} \rightarrow comp_i\} \tag{8}$$

$$DAG = \{N, E\} \tag{9}$$

The DAG is the key element for representing the *structure* of a vc_i.

Behavior
The *behavior* of the vc is established by assigning values to its properties, that is, by associating the function of the vc with the infrastructure (H) defined in the configuration file. The vc_i are deployed on $H \in I$, being I the whole infrastructure (e.g. a cloud). The consumption of a set of resources (R) of the specified infrastructure (processor -*CPU*-, memory -*MEM*-, file system -*FS*-, and network -*NET*-) is denoted as $R \in H$ per each vc_i, which are observed by a set of metrics (M).

$$R = \{CPU, MEM, FS, NET\} \tag{10}$$

$$M = \{total\text{–}usage, per\text{–}core\text{–}usage, ..., m_{n-1}, m_n\} \tag{11}$$

H, R, and M follow a hierarchy of elements defined as:

$$\forall h \in H : h = \{r, r \subseteq R\} \tag{12}$$

$$\forall r \in R : r \supset value, value \in \mathbb{R}, m(value) \tag{13}$$

Equation 12 specifies that each physical computer h (where a vc_i runs) has a subset of physical resources r. Likewise, Equation 13 specifies that each physical resource r has a $value$ denoting the performance of r for vc_i, and a metric m observes that $value$ for performance analysis.

Each resource r produces several $values$ in the continuous numerical space. Thus, a huge set of values is generated per resource r. These values are used for computing *Utilization Factors* (*UF*), which inform about the status performance of a resource r. Although the resources produce a lot of values and data, we are interested in such values of UF that could initiate a *risk situation*. Then, according to the ISO 31000 standard[5] for risk management [37], the values of UF are discretized in scales: $low \in [0, 0.33)$, $medium \in [0.33, 0.66)$ and $high \in [0.66, 1]$. These thresholds indicate the level of performance (*_lvl*) of each resource r_i, as indicates Equation 14.

5 www.iso.org/iso-31000-risk-management.html

$$UF = \{CPU\_lvl, MEM\_lvl, FS\_lvl, NET\_lvl\} \tag{14}$$

The UF of CPU in an instant of time t is defined by (15).

$$U_{CPU} = 1 - \left[\frac{T_{CPU} - C_{CPU}}{T_{CPU}}\right] \tag{15}$$

where, T_{CPU} is the total processing capacity of the physical computer, given by the sum of the capacity of each of the cores and C_{CPU} is the CPU usage at the current time.

The UF of the file system in an instant of time t is calculated by (16).

$$U_{FS} = 1 - \left[\sum_{i=1}^{f}\left(\frac{T_{FS_i} - C_{FS_i}}{T_{FS_i}}\right)\right] \tag{16}$$

where, f is the number of partitions available on the physical computer, T_{FS_i} is the total capacity of the current partition on the physical computer, and C_{FS_i} is the consumption of the current partition at a given moment. As shown, the multiple storage partitions associated to a studied object are considered in Equation (16).

The UF of memory is calculated by (17).

$$U_{MEM} = 1 - \left[\frac{T_{MEM} - C_{MEM}}{T_{MEM}}\right] \tag{17}$$

where, T_{MEM} is the total memory on the physical computer, and C_{MEM} is the memory consumption at time t.

The UF of network is calculated by (18).

$$U_{NET} = 1 - \left[\frac{T_{NET} - (TX_{NET} + RX_{NET})}{T_{NET}}\right] \tag{18}$$

where, T_{NET} is the total capacity of the network in bytes, TX_{NET} is the number of bytes transmitted, and RX_{NET} is the number of bytes received.

The set UF is the key element for representing the *behavior*.

As a result of this stage, a DfE is obtained, conformed by the three elements before described (structure, behavior, and function). The second stage of the method operates on this data structure.

$$DfE = \{DAG, UF, Tr\} \tag{19}$$

3.2. Phase 2: Standardized access to DST by means of WoT

At this point, a DfE provides a representation of the necessary data of the vc. However, we require a helpful representation to interact with the vc; such an interaction may be machine to machine or human to machine. For achieving this flexibility, this representation is based on the Web of Things (WoT) principles [38]. This standardized representation of a vc is named *WoT card*. In addition to the information captured by DfE, *metadata* of the vc are also added to the *WoT card*. These metadata are: IP addresses, volumes, ports, namespaces, etc. A *WoT card* is defined as shows Equation 20.

$$WoTcard = \{DfE, metadata\} \tag{20}$$

In the case of a VCS, such elements are defined recursively to capture data about structures and transformations used and performed by the whole VCS and its components respectively.

According to the WoT recommendations, the generation of the WoT cards must be based on ontologies. In this sense, we defined and created an ontology (named *VC Docker FU Ontology*[6]), which can be adapted to the context of any WoT card in several scenarios. The *VC Docker FU Ontology* is used as a reference in the whole generation of WoT cards during the representation of *vc*. This ontology comes from two more ontologies, it extends from the *VC Docker Ontology*[7], which extends from the *VC ISO Ontology*. The latter ontology was created from scratch according to the norm ISO/IEC JTC 1/SC 38[8], it defines all the concepts and constraints of the norm in an abstract manner. The *VC Docker Ontology*, in its original version, already defines concepts and constraints of virtual containers into the Docker environment, it was adapted in line with the *VC ISO Ontology*; some additional concepts and restrictions were included to fulfill with the ISO norm. The *VC Docker FU Ontology* adds concepts about the behavior related to infrastructure resources -CPU, MEM, FS, and NET- (such as levels of utilization and properties of such values), and function of virtual containers (such as base functions and tasks).

Technically a WoT card is based on an abstract class named *Thing*, which is the base object for modeling in the WoT approach. It is based on the representation structure of *Thing Description (TD)*[9]. Thus, a WoT card is composed of three elements: *i) metadata* (of Thing), which contains interactions (how *Thing* can be used); *ii) vocabulary*, which contains concept definitions used into the *Thing Description* structure, useful for interactions; and *iii) URIs*, which are useful to identify resources into *Thing Description*, these are Internet links denoting relationships between *Thing* and other resources on the WoT.

The WoT card was designed so that an external user can interact with it by asking about: *properties*, *actions* and *events*. Properties contain information about the *Thing*, such as behavior (*UF*), structure (*nodes* and *edges* of the DAG), and metadata of the *VC*. Actions refer to the functions of the *Thing*, including tasks (*Trs*) executed by its components. Events refer to alerts on behavior changes, such as defined by the *utilization levels* (*CPU_lvl, MEM_lvl, FS_lvl, NET_lvl*).

Then, a WoT card is represented as a file following the format and structure of JSON-LD[10]. Listing 1 illustrates a portion of an example of WoT card.

Listing 1: Thing Description (TD) structure following the JSON-LD format.

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "996ba6e...aec5f14",
    "@type": "Thing",
    "td:title": { "@value": "..." },
    "td:description": { "@value": "..." },
    "properties": {
        "ctv:metadata": { data{} },
        "ctv:structure": { data{} }
    },
    "actions": { "ctv:functions": {input{}, output{}} },
    "events": { "ctv:behavior": {} }
}
```

3.3. Phase 3: Consumption

After the WoT card has been generated and its data stored, it is ready for consumption by means of a *DST*. For the *DST* to be accessible and consumed, it must become

[6] Available at github.com/adaptivez/VirtualContainerOntology

[7] github.com/langens-jonathan/docker-vocab/blob/master/docker.md#config

[8] www.iso.org/committee/601355.html

[9] It is the base model for describing any IoT Thing in the W3C Web of Things approach. *Thing Description* describes the metadata and interfaces of Things. www.w3.org/TR/wot-thing-description

[10] *JavaScript Object Notation for Linked Data*. www.w3.org/TR/json-ld11

an intermediary between the modeled object (*vc*) and the consumer. This is possible by using a RESTful system, which can process requests with the most common HTTP actions: GET, POST, PUT, DELETE. In this way, any artifact making REST type requests can consume the DST. The consumption can be on properties, actions, or events, which are defined as follows.

$$ConsumProperty = \{WoTcard, \texttt{property}\} \tag{21}$$

$$ConsumEvent = \{WoTcard, \texttt{event}\} \tag{22}$$

$$ConsumAction = \{WoTcard, \texttt{action[input]}\} \tag{23}$$

Each element of the WoT card is universally identified and accepted by other physical and/or abstract entities (e.g. other *vc*, VCS, applications, devices, human-requests, etc.) by means of a URI[11].

For the consumption of DST properties (21), it is necessary to give the URI of the DST and the specific `property` to access. Also, in the event consumption (22), the URI of the DST and the `event` to be accessed must be given. For invoking actions (23), it is necessary to give the URI of the DST, the `action` to be performed and the `input` required for that action as parameter. In the three types of consumption, a JSON object is obtained as a response indicating a *value* if a `property` or `event` were requested, or a *value* or *resultant flag* if an `action` was invoked. Next, an example of consumption of the property "`platform`" and the function "`sum`" are given.

Request (property):

```
https://www.example.com/wotmodel/containers/123456789/platform
```

Response:

```
{ "platform" = "Docker" }
```

Request (function):

```
https://www.example.com/wotmodel/containers/123456789/sum/2/3
```

Response:

```
{ "result" = "5" }
```

4. DST Prototype

This section describes the implementation of a prototype for building *DSTs* based on the proposed method. The components of this prototype and its interactions are depicted in Figure 4. The components were implemented as microservices (encapsulated into virtual containers), coded by using Python 3.0, except for the *Observation* component, which was implemented by using JavaScript and PHP because of the nature of observation tasks. Next, each component is described.

[11] A URI (*Universal Resource Identifier*) identify, over the Internet, a resource (webpage, image, audio, video, file, IoT thing, WoT thing, etc.) by means of a unique and universal manner.

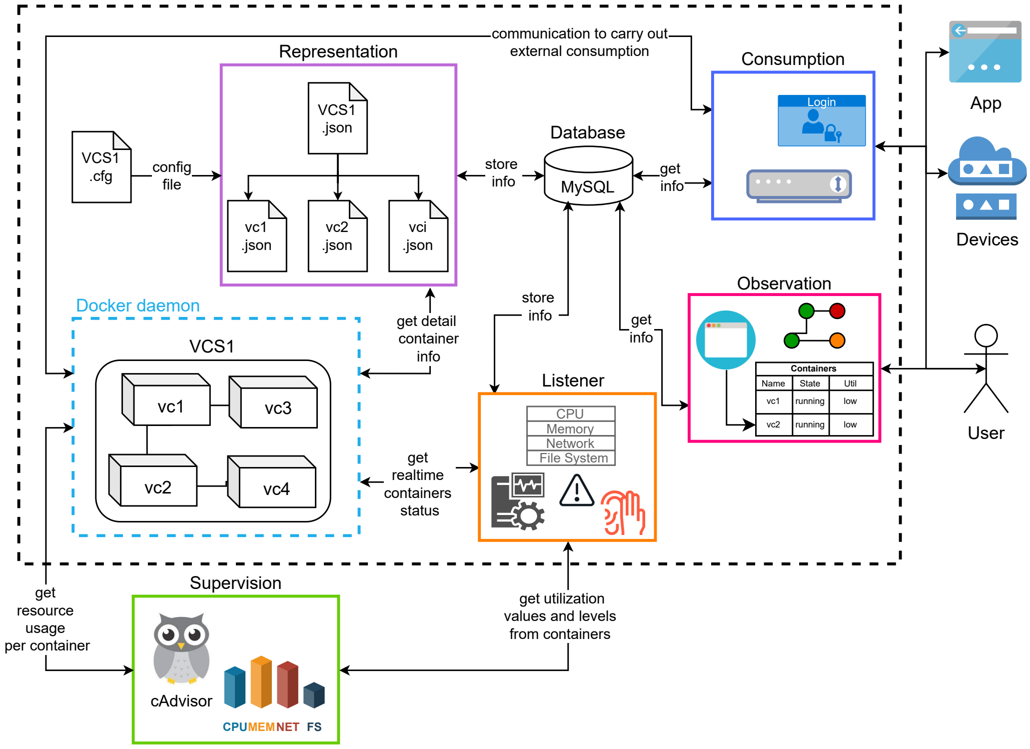

Figure 4. Components of DST prototype.

The prototype was deployed on the Docker platform, but *DSTs* may be created from another platform, such as LXC[12], Hyper-V[13], or rkt[14], where a *vc* can be represented by a *YML* or *YAML* file.

4.1. Representation

In this service, the configuration file (*YML*) of the VD is parsed to build the DfE, capturing structure, behavior, function and metadata of the participants in a dataflow from an IoT device to the decision-making process. After the creation of DfE, the WoT cards are generated and its corresponding URIs defined. In this way, a decision-making process can consume the *WoT card* information (properties, events, and actions). The URIs must follow a defined *namespace*, as shows the Expression 24:

$$\texttt{http://www.example.com/wotmodel/containers/} \\ \texttt{container\_id/\{property,event,action\}} \quad (24)$$

The WoT cards along with the DfE are stored in a MySQL database.

4.2. Listener

This service monitors the state (behavior) of a given VD (any of *vc*, *VCS* or $CApp$). It is in charge of storing and keeping updated, in real-time, all the captured information by requesting status information from the Docker daemon and registering, in the database, each event producing a change on the VD. It also keeps a communication with the *Supervision* service to reflect any change on VD utilization levels, which are also stored in the database.

---

12 https://linuxcontainers.org/lxc
13 https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows
14 https://www.openshift.com/learn/topics/rkt

4.3. Supervision

This service supervises the VD and performs the acquisition of metrics through an external agent, called *cAdvisor*. This is an API that provides information about the metrics of the *vc* and the physical computers on which it runs. When acquiring the values of the metrics, it calculates the behavior values of VD (utilization levels of resources used by VD). It also responds to requests from the *Listener*, which is monitoring the VD and returns values of utilization levels of resources (high, medium, low) about CPU, MEM, FS, or NET, as well as the timestamp when values were captured.

4.4. Observation

This service offers options for observing the VD (structure, behavior, and function). It is a web application with intuitive interface designed for human consumption. Three tasks can be performed: *1) discovering of* VDs, for searching the *vc*s or a specific $CApp$ by using its properties (name, description, type, creator, owner, etc.); *2) monitoring* VDs, to know easily the behavior of the resources used by a VD by means of warning color signs (red for critical, yellow for intermediate, and green for normal) and its utilization level values; *3) observing risk levels*, to know the risk of failure of the applications by means of a graph denoting virtual containers in nodes and its relationships in edges.

4.5. Consumption

This service acts as a gateway, is in charge of attending and processing requests from external users (human users, software applications, virtual containers, etc.) trying to consume or interact with the given VD. This is performed by using an API REST for GET, POST, PUT, and DELETE requests. Three types of consumption are considered: *properties, events, and actions* depending on the desired consumption/invocation. For properties and events, this service queries the *WoTcard* of the VD, then gets the corresponding data from the database to send it to the requester. For actions, the service queries the *WoTcard* of the VD, then establishes a connection to the corresponding VD, which executes the action and returns the result to the requester. All responses are into a JSON file. This is illustrated by invoking the clustering task *kmeans* with the parameters *k* and a dataset named *data*.

Preparation:

```
URI = https://www.example.com/wotmodel/containers/123456789/kmeans
input = {"k"=2,"data":[{"col1":1,"col2":0,"col3":2},
{"col1":2,"col2":1,"col3":1},{"col1":0,"col2":0,"col3":2}]}
```

Request:

```
request.post(URI,input)
```

Response:

```
{"result" :
{"cluster1":[{"col1":1,"col2":0,"col3":2},{"col1":0,"col2":0,"col3":2}],
"cluster2":[{"col1":2,"col2":1,"col3":1}]}}
```

5. Results

The evaluation of the prototype for DST creation was conducted in two phases of experiments. In the first phase, the prototype was evaluated in a controlled manner to measure the response and service times in the construction of the DST and in its consumption. In the second phase, a case study is presented based on the creation of DST for a platform for continuous verification of contracts using a blockchain network.

Table 1 shows the infrastructure used by the VCS created for both cases of study.

Table 1: IT Infrastructure used in the experiments.

ID	Cores	Processor	MEM	HDD	OS
Server1	4	Intel(R) Core i5	16 GB	256 GB	macOS BigSur
Server2	12	Intel(R) Xeon(R) E7-4830	128 GB	1 TB	CentOS 7

5.1. Metrics

The performance of the prototype was evaluated by capturing the following metrics.

- *Service time* (ST): The time required by a microservice (VD) to complete a given task.
- *Response time* (RT): The time observed by an end-user or a decision-making process to complete a given task. This time considers the initial time to get data, create the representation and store it in the database when an end-user builds a DST. This metric also measures the initial time when an end-user sends a request to the prototype and the time spent by DST to process it plus the time spent by it to deliver the results to the end-user.

5.2. Controlled evaluation

To conduct the evaluation of the prototype, a containerized application ($CApp$) was deployed on the previously described infrastructure, one instance of the $CApp$ running on one virtual container vc. This $CApp$ extracts data form real traces produced by ECG medical devices[15] [11], and builds workloads at a given rate time, following a synthetic distribution. An input parameter defines the amount of data to be include in the workload.

By using the $CApp$, several experiments were carried out by varying the number of vc and IoT data sources (ECG sensors), as well as the timing when the DST captures the behavior of the $CApp$; this latter is we call *slot*.

We captured the ST and RT metrics for each experiment, each one was performed 31 times (according to the Central Limit Theorem [39]) to capture the median value of both ST and RT.

Different combinations of virtual containers (vc) and DSTs (dst) were tested, these combinations were defined in the form $vcW - dstZ$, where W is the desired number of virtual containers (vc) in the combination, and Z is the desired number of DSTs. That means 1 (of Z) DST watches W virtual containers. For example, Expression 25 means 1 DST watching 5 virtual containers, this results in a total $vc = 5$. Expression 26 means 5 DST watching 5 virtual containers, this results in a total $vc = 25$.

$$vc5 - dst1 \tag{25}$$

$$vc5 - dst5 \tag{26}$$

These combinations also was executed by varying the *slot* parameter from 1, 10, to 100 seconds. Each combination of these parameters produces a median value of ST and RT, which are evaluated to show the behavior of the DST costs. The total time of ECGs extraction was 10 minutes.

Analysis of results

Figure 5 shows, in vertical axis, the ST and RT by two key operations related to the building of a DST ($GetData$ and $StoreData$ tasks) produced by the different number of virtual containers, evaluated in these experiments. This experiment only shows the ST and RT observed by either end-users or a decision-making application. As it can be seen,

[15] IoT devices for acquiring electrocardiogram (ECG) signals.

the prototype can build in just seconds $DSTs$ for multiple VCS (17,5 secs for creating $DSTs$ for 100 applications, each connected to an IoT data source). This time is only spent by the prototype once, which means that this is affordable for many environments. Moreover, the $GetData$ task (parsing YML files and creating the DfE), as it was expected, was the more significant task in the building of a DST, whereas $StoreData$ task (indexing the DfE in a database) results were not significant for the DST building RT.

Figure 6 shows, in vertical axis, the ST (for the $Representation$ task) spent by the building of the $DSTs$ according to the sequences of DSTs and virtual containers evaluated in these experiments (horizontal axis). As expected, the more the number of DSTs, the more the ST spent by the prototype to create the representation of these $DSTs$.

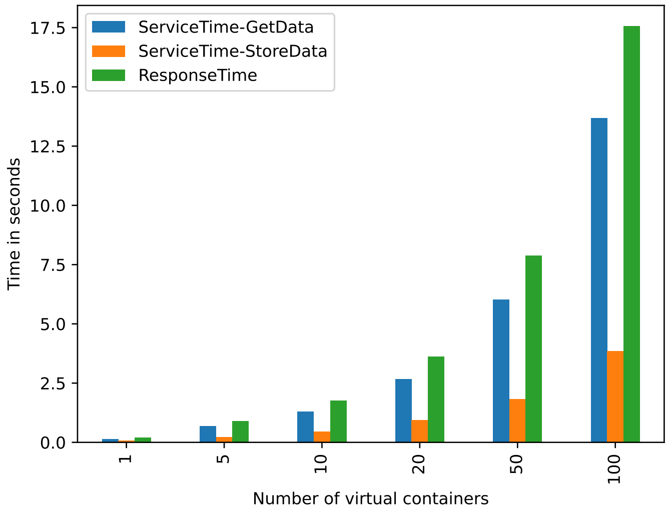

Figure 5. Service and Response times produced by the tasks *GetData* and *StoreData*.

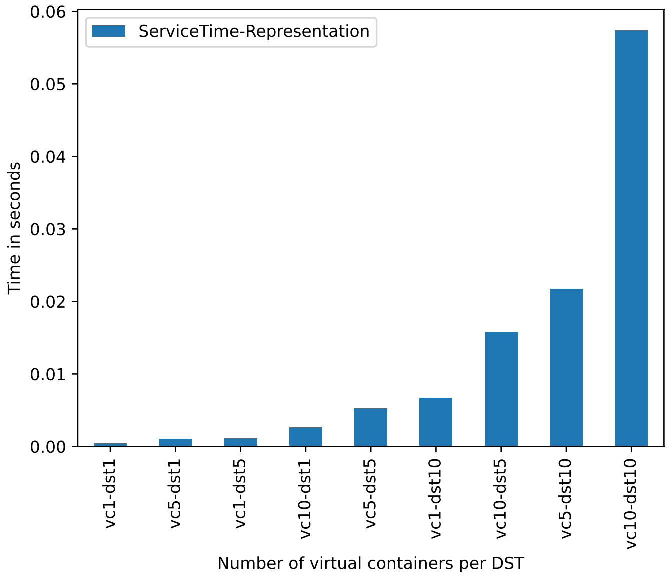

Figure 6. Service time produced by *Representation* component.

Figure 7 shows, in vertical axis, the RT spent by the $DSTs$ to retrieve information about VDs and PDs to the end-user (in this case a DST client application) per different sequences of DST and virtual containers (horizontal axis) for different time $slot$. It can be observed that increasing the number of vcs per DST also increases the number of requests performed by the *DSTs* per slot, increasing *RT*. The *RT* obtained is acceptable as soft real time [40].

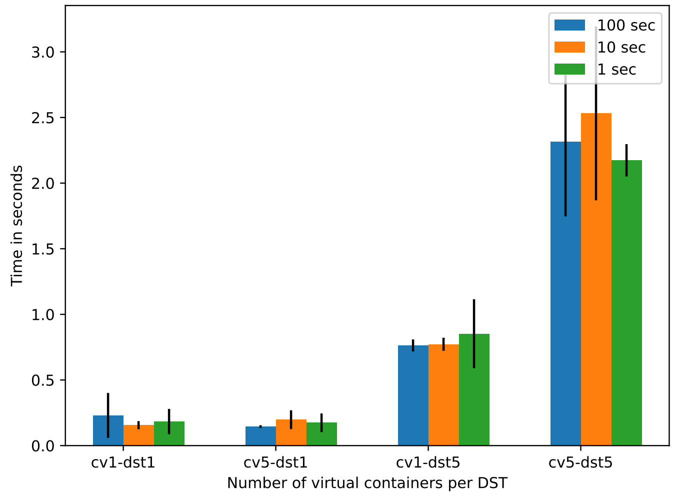

Figure 7. Response time in *DST* consumption

5.3. A case study: blockchain network for continuous contract verification

The previous evaluation showed the costs in time associated to create *DSTs* for decision-making process to get IoT data (by using simple REST API) without dealing with technology elements from IoT and cloud, just invoking tasks on DSTs.

We also conducted a case study to show the flexibility of *DSTs* into a dataflow composed by an end-user (human, device or application), *DSTs*, virtual containers (*VDs*), and IoT devices with sensors attached (*PDs*). This dataflow was emulated from a real trace of a logistic transportation of a supply chain of food, which is used by a *VCS* implementing a blockchain service for the verification of contract violations by monitoring GPS, temperature, and speed sensors of a set of transportation trucks [41].

Figure 8 shows the conceptual representation of this case study. As it can be seen, two *DSTs* were created for two *VCS* (including three virtual containers). The *DSTs* deliver to end-users or applications (decision-making processes) information about *VDs* (the system) and *PDs* (physical devices).

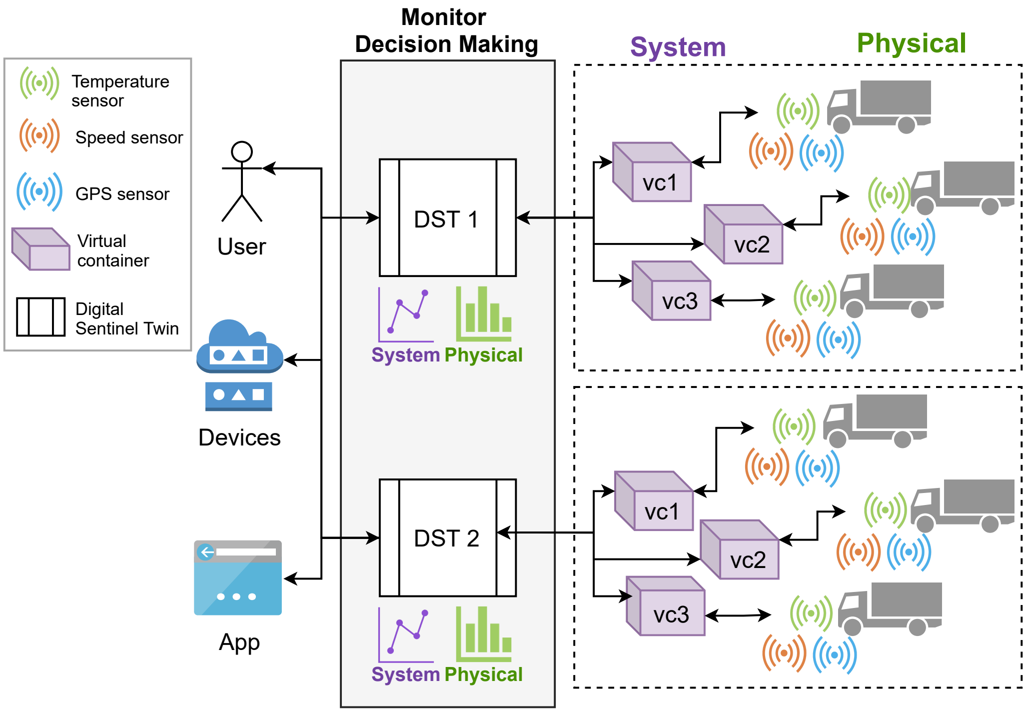

Figure 8. Conceptual representation of the scenario for the case study.

Figure 9 shows, in the horizontal axis, a timeline of the tasks performed by participants on the dataflow (vertical axis) of verifying contract violations: *Build* (tsk1), *Data Acquisition of Temperature* (tsk2), *Data Acquisition of Speed* (tsk3), *Data Acquisiton of GPS* (tsk4), *Send Request* (tsk5), *Get Data* (tsk6), and *Deliver Request* (tsk7). The timeline for this case study was 10 minutes. In *tsk1* the prototype builds two *DSTs*. Then, the data acquisition was carried out from IoT sensors (*tsk2*, *tsk3*, and *tsk4*) by the virtual containers, which were stored on the blockchain network. Also during the timeline, every 10 seconds, the virtual containers verified, registered, and reported contract violations on the blockchain network: first the consumer requests to *DST* (*tsk5*), then the blockchain is queried by the corresponding virtual container (*tsk6*), and finally the *DST* responses to the consumer (*tsk7*). As it can be seen, the impact of the *DST* creation (*tsk1*) and communications (*tsk5* and *tsk7*) is not significant in comparison with the time spent by get data from the blockchain network (*tsk6*) and the data transfer from sensors to the blockchain network (*tsk2*, *tsk3*, and *tsk4*). Figure 9 also shows that *DST* can capture the data produced by both, *VD*s (*tsk6*), and *PD*s (*tsk2*, *tsk3*, and *tsk4*).

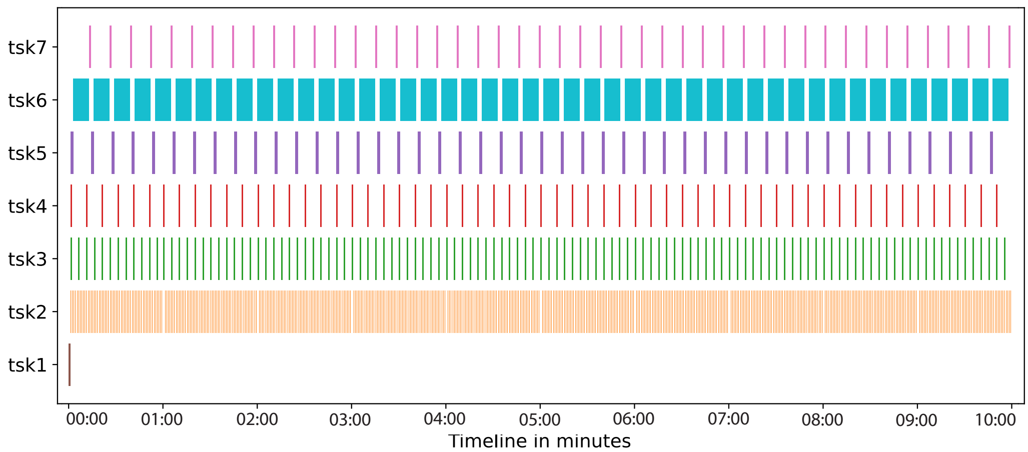

Figure 9. Time of tasks in the case study.

We observed that DSTs were able to inform to end-users, on demand and in real time, about contract violations. From the total number of requests (47) to the DSTs, just in 3 requests the DSTs informed contract violations.

The DSTs can also deliver, on demand and in real time, the data rate produced and received by PDs to the end-user. That means, the behavior of the PDs can be known by end-users in decision-making time by analyzing these data. In this case study the prototype showed a regular data production from sensors, with a reduction and increment of the data rate. This could imply to a potential bottleneck in the reception of data or a possible inconsistent data production from sensors at IoT devices. Figure 10 shows the received data amount of 47 user requests to the DSTs.

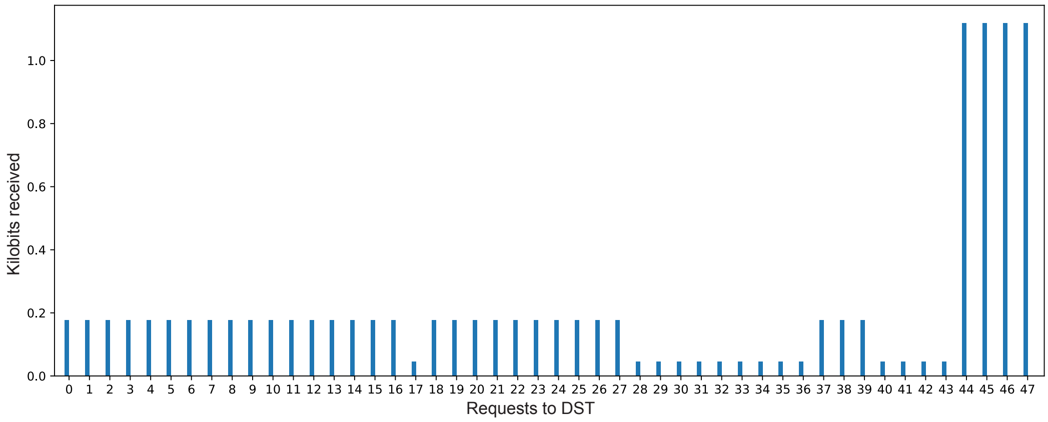

Figure 10. Kilobits received in the requests.

The averages of consumed resources r (processor -CPU-, memory -MEM-, file system -FS-, and network -NET-) by the prototype in the case study are shown in Table 2. To obtain them, first the consumption of such resources were measured before and during the case study, this was carried out 32 times ($w = 32$). Then the differences between initial ($r_{k_{ini}}$) and final ($r_{k_{fin}}$) values were computed and added. Finally the average of the differences were obtained, as shows Equation 27.

$$r_{k_{avg}} = \frac{\sum_{i=1}^{w} \left(r_{k_{fin_i}} - r_{k_{ini_i}} \right)}{w} \tag{27}$$

Table 2: Average values of resource consumption.

CPU (%)	MEM (megabytes)	FS (megabytes)	NET (megabits/sec)
2.306	33.884 (0.02%)	20.109	9.556

It is important to note that the blockchain network is not of exclusive use of this prototype, it can be consumed by external applications. This VCS (blockchain network) can be replaced by other VCS (e.g. a data analytics system), in such case the DST must deliver the data produced by this new VCS without performing deep changes, but rebuilding the DfE of the DST.

6. Discussion

In this paper, we demonstrated the viability of the proposed method by applying the implemented prototype in two scenarios. The first one is regarding a controlled evaluation for extracting data from traces produced by ECG medical devices. This scenario showed the Response and Service Time performance during the building and consumption of *DST*s. The second scenario demonstrated the flexibility of *DST*s to get information (verification of contract violations on a blockchain network) in real-time from a dataflow of transportation logistics.

The obtained prototype was tested on distinct scenarios for intermediate and partial experiments before obtaining the results reported in this paper. In all these experiments, the prototype showed good performance in several tasks, such as discovering *vc*s, monitoring *VCS*, supervising *CApps* through created *DST*s. Several interactions were performed on these *DST*s, accessed by other *CApp*s, human requests, and software applications.

According to the results of the controlled evaluation (subsection 5.2), we can see that augmenting the number of *vc*s per *DST* increases exponentially the Response Time for both the building and consumption of *DST*s. The building of 1 *DST* with 5 *vc* (*vc*5-*dst*1) takes an average Response Time of 0.90 seconds (see Figure 5). The consumption of the *DST* with the same configuration (*vc*5-*dst*1) takes an average Response Time of 0.52 seconds (see Figure 7).

The case study (subsection 5.3) supports the results achieved in the controlled evaluation. In this case, the average Response Time during the building of the *DST*s (sequence *vc*3-*s*2) was 1.2 seconds (see Figure 9). For the consumption of the *DSTs* the average Response Time was 13 seconds (see Figure 9). However, it is important to note that from these 13 seconds, 10 seconds correspond to the communication to/from the blockchain networks for obtaining data. Thus, we can conclude that 3 seconds is the real Response Time for the consumption.

In all the experiments of the prototype, the interaction with the created *DSTs* was easy because complex requests were not necessary. The benefits of using the created *DSTs* are as follows:

- Standardized interaction. Since a WoT card is based on W3C guidelines, a *DST* can be consumed by distinct users (humans, devices, or applications),
- Easy consumption. Through a *DST*, users can: *a)* access to data, properties, and events; and *b)* invoke tasks and functions, both directly on target devices (*VD*s or *PD*s),
- Flexible access. A *DST* can be exploited by external users by means of RESTful requests from distinct locations to the one of the *DST* environment,
- Decision-making aid. *DST*s can be used as a mean in decision-making tasks (discovering, classification, monitoring, supervising, migration, to mention a few),
- Generation of *DST*. The building of *DSTs* is quite simple and transparent if a well-structured file configuration (*YML* or *YAML*) is given,
- Minimal required resources. The execution of *DSTs* requires minimal infrastructure resources (CPU, MEM, FS, and NET).

7. Conclusions

This paper presented a cloud-based WoT method for creating digital twins of IoT devices, named (*Digital Sentinel Twins -DST-*). A *DST* is an object that abstracts physical or virtual devices to operate over them by consuming its properties, events, or invoking its functions. This object has the advantage that by investing minimal time and resources, an external user (human, software application, or virtual devices) can access to all the data and functions of those devices. That is useful for interacting with IoT devices in several scenarios.

The method comprises three phases: *a) Modeling*, where the data of the *VD* or *PD* are acquired, with these elements that device is modeled, generating a *Dataflow Entity* (DfE); *b) Standardization*, where the elements of the model are represented into a standardized representation named *WoT card*; this representation follows the guidelines of the Web of Thing to make its elements universally accessible by means of URIs; and *c) Consumption*, the advance of the *WoT card* generated is that it can be consumed in external scenarios by distinct users (human, software applications, or virtual devices) in different ways.

Based on the proposed method, a functional prototype was implemented. This prototype was tested by creating $DSTs$ in several experiments considering distinct scenarios of use (discovering and monitoring of VCs and applications, supervising $CApps$, etc.). By means of the created $DSTs$, it was possible to consume data and invoke functions of virtual and physical devices. In this paper, two experiments were reported to demonstrate the viability of the proposed method, creating flexible and useful $DSTs$. The first experiment was to show the spent time for creating and consuming $DSTs$. The second one was to demonstrate the use of $DSTs$ into a scenario of a blockchain network for verifying contract violation on sensors used in product transportation logistics.

A DST creates an abstract window for decision-making processes to get information/data from virtual and physical devices. It acts as a useful mechanism to interact with those devices in several scenarios. Its creation is not expensive in terms of time and computational resources, and it produces a access to data and functions of the target devices. These characteristics may be obtained without managing complex details associated to virtual and physical devices and cloud computing infrastructures.

Nevertheless the benefits obtained by the proposed method, it is important to mention some limitations of the proposed work:

- The creation of DSTs only can be achieved if a well-structured configuration file is given, in YML or $YAML$ format,
- A DST has no other way to consume it that RESTful requests,
- When target devices (VDs or PDs) and DSTs reside in the same infrastructure, the Response Time of performed tasks increases exponentially.

As further work, the inclusion of security aspects into the DSTs is considered; this will enable its manageability and control while maintaining its flexibility of use.

Funding: This research was partially funded by the project Num. 41756 "Plataforma tecnológica para la gestión, aseguramiento, intercambio y preservación de grandes volúmenes de datos en salud y construcción de un repositorio nacional de servicios de análisis de datos de salud" by FORDECYT-PRONACES, Conacyt (México).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Piccialli, F.; Casolla, G.; Cuomo, S.; Giampaolo, F.; Di Cola, V.S. Decision making in IoT environment through unsupervised learning. *IEEE Intelligent Systems* **2019**, *35*, 27–35. doi: 10.1109/MIS.2019.2944783.
2. Troussas, C.; Krouska, A.; Sgouropoulou, C. Improving Learner-Computer Interaction through Intelligent Learning Material Delivery Using Instructional Design Modeling. *Entropy* **2021**, *23*. doi:10.3390/e23060668.
3. Wang, Y.; Li, P.F.; Tian, Y.; Ren, J.J.; Li, J.S. A Shared Decision-Making System for Diabetes Medication Choice Utilizing Electronic Health Record Data. *IEEE Journal of Biomedical and Health Informatics* **2017**, *21*, 1280–1287. doi:10.1109/JBHI.2016.2614991.
4. Peralta, G.; Iglesias-Urkia, M.; Barcelo, M.; Gomez, R.; Moran, A.; Bilbao, J. Fog computing based efficient IoT scheme for the Industry 4.0. 2017 IEEE international workshop of electronics, control, measurement, signals and their application to mechatronics (ECMSM). IEEE, 2017, pp. 1–6. doi:10.1109/ECMSM.2017.7945879.
5. Al-Hamadi, H.; Chen, I.R. Trust-Based Decision Making for Health IoT Systems. *IEEE Internet of Things Journal* **2017**, *4*, 1408–1419. doi:10.1109/JIOT.2017.2736446.
6. Manavalan, E.; Jayakrishna, K. A review of Internet of Things (IoT) embedded sustainable supply chain for industry 4.0 requirements. *Computers & Industrial Engineering* **2019**, *127*, 925–953. doi:j.cie.2018.11.030.
7. Morabito, R.; Cozzolino, V.; Ding, A.Y.; Beijar, N.; Ott, J. Consolidate IoT Edge Computing with Lightweight Virtualization. *IEEE Network* **2018**, *32*, 102–111. doi:10.1109/M-NET.2018.1700175.

8. Truong, H.L.; Dustdar, S. Principles for Engineering IoT Cloud Systems. *IEEE Cloud Computing* **2015**, *2*, 68–76. doi:10.1109/MCC.2015.23.
9. Du, Y.; Huang, K. Fast Analog Transmission for High-Mobility Wireless Data Acquisition in Edge Learning. *IEEE Wireless Communications Letters* **2019**, *8*, 468–471. doi: 10.1109/LWC.2018.2876344.
10. Tadakamalla, U.; Menascé, D. FogQN: An Analytic Model for Fog/Cloud Computing. 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), 2018, pp. 307–313. doi:10.1109/UCC-Companion.2018.00073.
11. Sánchez-Gallegos, D.D.; Galaviz-Mosqueda, A.; Gonzalez-Compean, J.L.; Villarreal-Reyes, S.; Perez-Ramos, A.E.; Carrizales-Espinoza, D.; Carretero, J. On the Continuous Processing of Health Data in Edge-Fog-Cloud Computing by Using Micro/Nanoservice Composition. *IEEE Access* **2020**, *8*, 120255–120281. doi:10.1109/ACCESS.2020.3006037.
12. Carrizales-Espinoza, D.; Sánchez-Gallegos, D.D.; Gonzalez-Compean, J.L.; Carretero, J. A Federated Content Distribution System to Build Health Data Synchronization Services. 2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2021, pp. 1–8. doi:10.1109/PDP52278.2021.00011.
13. Yang, H.; Kim, Y. Design and Implementation of Fast Fault Detection in Cloud Infrastructure for Containerized IoT Services. *Sensors* **2020**, *20*. doi:10.3390/s20164592.
14. Sánchez-Gallegos, D.D.; Carrizales-Espinoza, D.; Reyes-Anastacio, H.G.; Gonzalez-Compean, J.; Carretero, J.; Morales-Sandoval, M.; Galaviz-Mosqueda, A. From the edge to the cloud: A continuous delivery and preparation model for processing big IoT data. *Simulation Modelling Practice and Theory* **2020**, *105*, 102136. doi:https://doi.org/10.1016/j.simpat.2020.102136.
15. Carrizales, D.; Sánchez-Gallegos, D.D.; Reyes, H.; González-Compeán, J.; Morales-Sandoval, M.; Carretero, J.; Galaviz-Mosqueda, A. A data preparation approach for cloud storage based on containerized parallel patterns. International Conference on Internet and Distributed Computing Systems. Springer, 2019, pp. 478–490.
16. Saiáns-Vázquez, J.V.; Ordóñez-Morales, E.F.; López-Nores, M.; Blanco-Fernández, Y.; Bravo-Torres, J.F.; Pazos-Arias, J.J.; Gil-Solla, A.; Ramos-Cabrer, M. Intersection Intelligence: Supporting Urban Platooning with Virtual Traffic Lights over Virtualized Intersection-Based Routing. *Sensors* **2018**, *18*.
17. Al-Tarawneh, M.A. Mobility-Aware Container Migration in Cloudlet-Enabled IoT Systems using Integrated Muticriteria Decision Making. *environments* **2020**, *11*.
18. Martínez-Gutiérrez, A.; Díez-González, J.; Ferrero-Guillén, R.; Verde, P.; Álvarez, R.; Perez, H. Digital Twin for Automatic Transportation in Industry 4.0. *Sensors* **2021**, *21*. doi: 10.3390/s21103344.
19. Ostermaier, B.; Schlup, F.; Römer, K. WebPlug: A framework for the Web of Things. 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010, pp. 690–695. doi:10.1109/PERCOMW.2010.5470522.
20. Assad, F.; Konstantinov, S.; Ahmad, M.; Rushforth, E.; Harrison, R. Utilising Web-based Digital Twin to Promote Assembly Line Sustainability **2021**.
21. Bevilacqua, M.; Bottani, E.; Ciarapica, F.E.; Costantino, F.; Di Donato, L.; Ferraro, A.; Mazzuto, G.; Monteriù, A.; Nardini, G.; Ortenzi, M.; Paroncini, M.; Pirozzi, M.; Prist, M.; Quatrini, E.; Tronci, M.; Vignali, G. Digital Twin Reference Model Development to Prevent Operators' Risk in Process Plants. *Sustainability* **2020**, *12*. doi:10.3390/su12031088.
22. Gao, Y.; Lv, H.; Hou, Y.; Liu, J.; Xu, W. Real-time Modeling and Simulation Method of Digital Twin Production Line. 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), 2019, pp. 1639–1642. doi:10.1109/ITAIC.2019.8785703.
23. Alaasam, A.B.; Radchenko, G.; Tchernykh, A.; Compeán, J.G. Analytic Study of Containerizing Stateful Stream Processing as Microservice to Support Digital Twins in Fog Computing. *Programming and Computer Software* **2020**, *46*, 511–525. doi:10.1134/S0361768820080083.
24. Lin, T.Y.; Shi, G.; Yang, C.; Zhang, Y.; Wang, J.; Jia, Z.; Guo, L.; Xiao, Y.; Wei, Z.; Lan, S. Efficient container virtualization-based digital twin simulation of smart industrial systems. *Journal of Cleaner Production* **2021**, *281*, 124443. doi:https://doi.org/10.1016/j.jclepro.2020.124443.
25. Borodulin, K.; Radchenko, G.; Shestakov, A.; Sokolinsky, L.; Tchernykh, A.; Prodan, R. Towards Digital Twins Cloud Platform: Microservices and Computational Workflows to Rule a Smart Factory. Proceedings of The10th International Conference on Utility and Cloud Computing. Association for Computing Machinery, 2017, UCC '17, p. 209–210. doi: 10.1145/3147213.3149234.

26. Paraiso, F.; Challita, S.; Al-Dhuraibi, Y.; Merle, P. Model-Driven Management of Docker Containers. 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 718–725. doi:10.1109/CLOUD.2016.0100.
27. Piraghaj, S.F.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience* **2017**, *47*, 505–521.
28. Medel, V.; Rana, O.; Bañares, J.a.; Arronategui, U. Modelling Performance Resource Management in Kubernetes. Proceedings of the 9th International Conference on Utility and Cloud Computing; Association for Computing Machinery: New York, NY, USA, 2016; UCC '16, p. 257–262. doi:10.1145/2996890.3007869.
29. Alam, M.; Rufino, J.; Ferreira, J.; Ahmed, S.H.; Shah, N.; Chen, Y. Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Communications Magazine* **2018**, *56*, 118–123. doi:10.1109/MCOM.2018.1701233.
30. Muralidharan, S.; Song, G.; Ko, H. Monitoring and managing iot applications in smart cities using kubernetes. *CLOUD COMPUTING* **2019**, *11*.
31. Muralidharan, S.; Yoo, B.; Ko, H. Designing a Semantic Digital Twin model for IoT. 2020 IEEE International Conference on Consumer Electronics (ICCE), 2020, pp. 1–2. doi: 10.1109/ICCE46568.2020.9043088.
32. Chittaro, L.; Guida, G.; Tasso, C.; Toppano, E. Functional and teleological knowledge in the multimodeling approach for reasoning about physical systems: a case study in diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics* **1993**, *23*, 1718–1751.
33. Lind, M.; Zhang, X. Functional modelling for fault diagnosis and its application for NPP. *Nuclear Engineering and Technology* **2014**, *46*, 753–772.
34. Chandrasekaran, B.; Josephson, J.R. Function in device representation. *Engineering with computers* **2000**, *16*, 162–177.
35. Yildirim, U.; Campean, F.; Williams, H. Function modeling using the system state flow diagram. *AI EDAM* **2017**, *31*, 413–435.
36. Umeda, Y.; Takeda, H.; Tomiyama, T.; Yoshikawa, H. Function, behaviour, and structure. *Applications of artificial intelligence in engineering V* **1990**, *1*, 177–193.
37. Luko, S. Risk Management Principles and Guidelines. *Quality Engineering* **2013**, *25*.
38. Guinard, D.D.; Trifa, V.M. *Building the web of things: with examples in node. js and raspberry pi*; Simon and Schuster, 2016.
39. Rosenblatt, M. A central limit theorem and a strong mixing condition. *Proceedings of the National Academy of Sciences of the United States of America* **1956**, *42*, 43.
40. Buttazzo, G.; Lipari, G.; Abeni, L.; Caccamo, M. *Soft Real-Time Systems*; Springer, 2005.
41. Martinez-Rendon, C.; Camarmas-Alonso, D.; Carretero, J.; Gonzalez-Compean, J.L. On the continuous contract verification using blockchain and real-time data. *Cluster Computing* **2021**, pp. 1–23.