Digital Dices: Towards the Integration of Cyber-Physical Systems Merging the Web of Things and Microservices

Manel Mena(✉), Javier Criado, Luis Iribarne, and Antonio Corral

Applied Computing Group (TIC-211), University of Almería, Almería, Spain
{manel.mena,javi.criado,luis.iribarne,acorral}@ual.es

Abstract. One of the main issues of devices and platforms related to Internet of Things (IoT) is that there exists a broad spectrum of different protocols addressing those devices. Management and connection to those things create integrability and usability issues. Hence, there is a need for a solution that facilitates the communication between different devices and platforms. The Web of Things (WoT) tries to handle interoperability issues by describing interfaces and interaction patterns among things. Thanks to the models proposed by the WoT, it is possible to decouple the description of things from the protocols handling the communication and implementation strategies. This article proposes Digital Dice as an abstraction of IoT devices inspired by the concept of Digital Twin, but capable of leveraging the advantages of microservices architectures. We focus on the creation of Digital Dices from WoT models. A Digital Dice consists in different facets that are able to handle a particular aspect of a thing, hence different WoT descriptions models will result in different microservices related to that particular thing. An architecture to handle multiple Digital Dices and their replicas is also proposed.

Keywords: Cyber-physical systems · IoT · Microservices · Web of Things · Digital Twins

1 Introduction

Different protocols related to the Internet of Things (IoT) must be taken into account to establish an ecosystem of devices capable of handling different aspects. Those protocols are usually divided into different layers [1], and their great number, heterogeneity and use of different technologies brings about the problem of interoperability between devices and/or platforms.

Related to the interoperability, there are solutions that try to support the integration of IoT devices in Smart Space environments [2], such as Node-RED, Eclipse Kura, Prodea, etc. These approaches are designed to control a limited

This work was funded by the EU ERDF and the Spanish MINECO under Project TIN2017-83964-R. Manel Mena has been funded by a Grant of the Spanish Government FPU17/02010.

number of devices, and are build as monolithic applications, so they lack a good scalability. Another common problem is to have bottlenecks in the communication due to the restrictions on most IoT devices (*e.g.*, low energy consumption, low computing power). In addition, there is a need to virtualize this type of devices for the purpose of carrying out tests without influencing the business processes [3]. To solve this last problem, the concept of Digital Twin (DT) was introduced as a virtual representation of a physical element, system or device, which can be used for different purposes [4]. However, this concept focuses on a monolithic approach for the management and representation of devices, and lacks of multiple levels that specifically addresses each facet of a device.

In this paper, we introduce the concept of Digital Dice (DD), an abstraction of IoT devices or cyber-physical systems capable of leveraging the advantages of microservices architectures. Furthermore, we explain how Digital Dices are closely related to the Web of Things (WoT) framework [5]. The WoT was created to define a common definition schema for Things. In particular, our proposal makes use of the Thing Description (TD) defined by the WoT. The Thing Description offers an overview of the features of a *thing*, as well as the endpoints to request those features. The TD gives a solution to the problem of feature discovery on *things*. The Digital Dices are a software abstraction of *things* that make use of TD to establish a connection to the thing as well as generate another Thing Description with new endpoints, offering a common communication protocol for the final user no matter the protocols used internally by the IoT device or the Cyber-physical System. Besides, we study the conversion between Things Description models and the different facets that are used by our DD.

The main contributions of Digital Dices are (*i*) the definition of a common communication protocol to control all kind of devices, a behaviour capable of lessening the number of requests on a IoT device; (*ii*) the capability of handling numerous request just replicating the facets needed; and (*iii*) the possibility using our Digital Dice as a Thing in other systems compatible with WoT.

This paper is structured as follows. In Sect. 2 we describe our Digital Dices as a solution to solve the problems described above. In Sect. 3 we study the Thing Description model [5] and how to convert Thing Descriptions to Digital Dices. Section 4 describes an example scenario. Section 5 offers a overview of related work, solutions and systems found in the literature. Finally, Sect. 6 describes the advantages of our Digital Dices and propose the future work.

2 Background

Digital Dices, like Digital Twins, are virtual representations of physical elements, but they offer several improvements. A digital dice proposes a virtual abstraction of IoT devices, cyber-physical systems or other virtual devices that is based on microservices, and aims to be agnostic to the protocols used by said devices. This DD should be compatible with the Web of Things framework, specifically the Thing Description model. Thanks to this fact, our digital dices can be totally compatible with different systems that make use of the standard, like Mozilla Web Thing framework [6].

The concept of *dice* (multiple faces) is given by how the microservices that represent our devices are characterized. These microservices oversee different aspects of the devices as follows:

(a) *Controller.* This microservice handles the communication with the user, and it is the one that manages the orchestration with the rest of the facets or directly to the IoT device as need.
(b) *Data Handler.* This facet is the one that handles the communication with the underlying database. This database has two main functions; it can log different requests done by the user so we can trace possible problems originated in our Digital Dices, and it can act as a buffer for the IoT device. This is done as follows. First, it tries to recover the requested data from the database before obtaining it from the physical device; if the data requested is newer than the time threshold configured (by default 10 s) by our DD, then this data will sent as response.
(c) *Event Handler.* This aspect processes the events generated in the IoT device. At the same time, it also will manage the connection with a future possible Complex Event Processing (CEP) subsystem [7].
(d) *User Inteface (UI).* The user interface established a micro-frontend for each feature controlled by our DD. The features can offer a UI that will be declared in the TD model that supersedes our DD. Besides those individual interfaces for each feature, we have a method in our UI microservice that can make a composition of all the individual features that have a UI. We call it the global UI of our Digital Dice. This approach provides the user with a reusable UI to interact with the device.

It is important to note that not all the facets will be always part of a DD. Furthermore, these facets can be extended, for instance, to include a Voice Activated Interface or an Open Data Handler capable of proactively send data dumps of a time frame worth of information into an Open Data system.

The connection of facets with IoT devices is one of the main problems that must be addressed. With this aim, a library capable of managing multiple protocols is required. Moreover, we need to establish what microservices have a direct connection with the IoT devices. To that end, we classified them in: (a) Hard Related Facets, with a direct connection with the IoT device (*e.g.*, Data Handler, Controller, Event Handler), and (b) Soft Related Facets, representing those that do not establish a direct connection with the IoT device (*e.g.*, UI).

The facets of the Digital Dices establish communication with the users following the standards, mechanisms and technologies by the W3C. One of the challenges that we face is to establish a system architecture for Digital Dices. This architecture has to be capable of sustaining multiple copies of the same microservices, having load balancing between them and detecting when a microservice is being over used so it can start a replica of it.

The microservice architecture [8] that we propose for the management of our Digital Dices is shown in the Fig. 1. It establishes a possible configuration proposed for the architecture. First, the `Edge` layer is composed basically by two

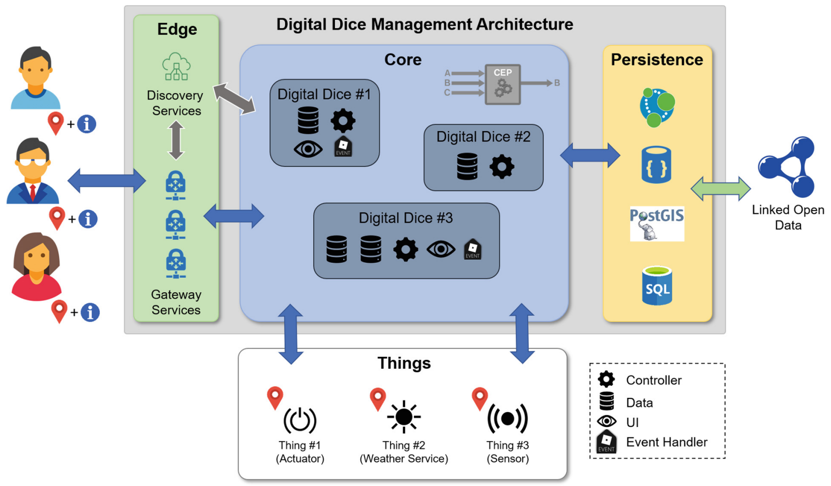


Fig. 1. Digital Dice architecture

types of microservices: (a) Gateways that will be in charge of redirecting the relevant requests to our Digital Dices, and (b) Discovery services, that keeps a register of each microservice contained in our Digital Dice Architecture. The Discovery Service allows our Gateways to redirect the request to the appropriate instance when necessary, allowing load balancing between the different replicas of said instances. Besides the *Edge*, the architecture has the *Core* of our system, which is where our Digital Dices and a series of auxiliary microservices are framed, such as authentication services or microservices for CEP. In addition, we will have the *Persistence* layer, which is composed of databases and possible services associated with them, for example different Open Data services. The *Things* layer of the architecture represents the physical devices associated with our Digital Dices, as well as external services, virtual components, etc.

Figure 1 illustrates the management of three Digital Dices. The DD #1 has the four facets described and it is responsible of handling an actuator to turn a switch on and off. The DD #2 only has two facets active because the interaction with the climatological information service is done through the Data Handler and Controller facets. The DD #3 has a duplicate Data Handler facet because the threshold number of accesses has been exceeded.

3 Thing Description to Digital Dice

In this section we describe how the conversion between a Thing Description (TD) and a Digital Dice (DD) is handled. First, we need to describe the formal model and a common representation for a TD, that can be considered as the entry point of a *thing*. This model is used as a centerpiece of our proposal as it helps us to define the actions, properties and events managed by our DD. At the same time, this model can be used as a starting point to generate a DD semi-automatically, by applying Model-to-Text transformation [9].

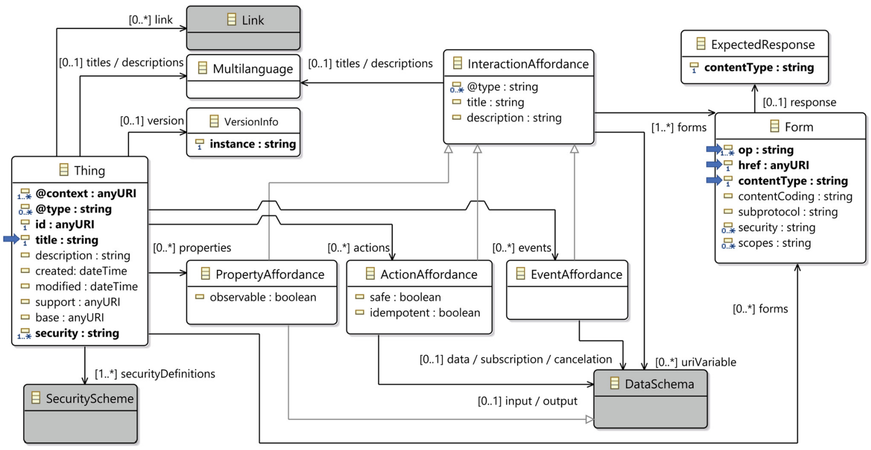


Fig. 2. Thing Description Model

Figure 2 shows an overview of the TD model. A *thing* is made up by the *properties*, *actions* and *events*. All the features defined by the TD are subclasses of *interaction affordance*, and it is made up by one or more *forms*. These forms will help us to define the methods that compose the DD generated by the TD. Figure 2 shows three fields in the form model that are mandatory, those three fields will be used to create our DD. The different possible values of each field can be consulted in the WoT TD definition [5]. Sometimes, those fields are omitted, that just means that they get their default value:

(a) `op` (operation) field has an array of string with the elements *readproperty* and *writeproperty*, if the feature is a `PropertyAffordance`. Furthermore, it will be an *invokeaction*, if it is an instance of `ActionAffordance`. *Subscribeevent* will be used as a default if it is an instance of `EventAffordance`;
(b) `contentType` field has as default value *application/json*.

The URIs generated in our DD will follow the pattern `https://{ip-address}:{port}/{thing.name}/{property|action|event}/{IntAffordance.title}/`. Besides the features defined in our DD, we have to offer an URI with the Thing Description Model that defines our DD, this can be accessed through `https://{ip-address}:{port}/{thing.name}/`.

Figure 3 describes the microservices or facets generated by the conversion of a TD into a DD. Properties always define a controller microservice, but the data handler will only be created if the property contains the operation *writeproperty*. In the same figure, we can depict how actions generate two microservices, a controller and a data handler. Nevertheless, if the *thing* has already a property created, it will only add the necessary methods to the corresponding microservices. In the case of the events, a controller and an event handler will be created.

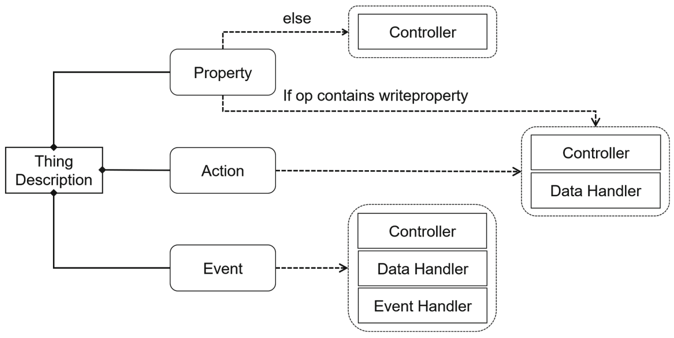


Fig. 3. Thing Description to Digital Dice.

As we explained in Sect. 2, our Digital Dice has another facet, the UI, that will be generated as a microservice. This microservice will be only generated if a property, action or event contains in `contentType` the parameter `ui = true`. This flag establishes that the DD has to create a visualization based on the data type of such feature. The visualization will be a micro-frontend, generated as an <`iframe`>, <`portal`> or <`component`> form. If our DD has one or more UI components (Fig. 4), a new method will be created in the UI microservice. This method will show a composition of all the generated components, and it will appear as a link in the TD that represents our DD.

Figure 4 shows how a Thing Description defines different actions, properties and events, and how those features can have associated different micro-frontend components. In this case, properties pi and p2, and action a3 have associated

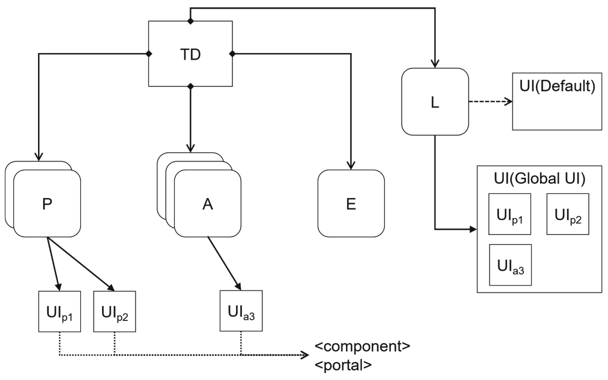


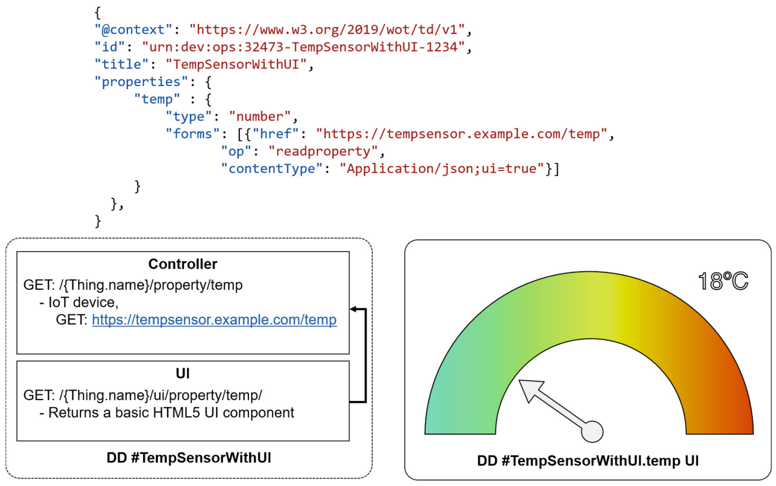
Fig. 4. Global UI composition.

UIs. Besides those three UIs, a global one will be generated as a link on the Thing Description with a composition of every UI component.

4 Example Scenario

In order to evaluate or approach, we propose an example scenario to transform a thing description into a digital dice. Figure 5 shows a temperature sensor represented by a TD and the respective microservices and methods created after the translation. The thing description represents a device with one property `temp` (number type) and contains a form with the address for obtaining the value from the physical device. The field `op` is established as *readproperty*, that means that only a controller will be created (because it is not *writeproperty*). Furthermore, the `contentType` has the parameter `ui = true`, which represents that our digital dice needs a UI microservice for the property.

As we can see in Fig. 5, two microservices are created. In the controller, a GET method responding with a simple number for the `temp` is deployed. The fact of being a GET method is because *readproperty* is bound to a GET method. The `temp` method builds a response with the data recovered from the IoT device. The UI microservice will make use of the same method of the controller to create a visual component that shows the value of `temp`. Since our DD has a feature with an UI, a composition with it will be created as a *link* in the TD of our DD. It should be stressed that the thing description depicting our digital dice does not has to be the same as the one from the original TD. It will probably have the same number of features, but with different `href` (the ones from our DD) and data related to the global UI composed by the declared UI features.



```
{
"@context": "https://www.w3.org/2019/wot/td/v1",
"id": "urn:dev:ops:32473-TempSensorWithUI-1234",
"title": "TempSensorWithUI",
"properties": {
     "temp" : {
         "type": "number",
         "forms": [{"href": "https://tempsensor.example.com/temp",
                "op": "readproperty",
                "contentType": "Application/json;ui=true"}]
     }
   },
}
```

Controller
GET: /{Thing.name}/property/temp
- IoT device,
  GET: https://tempsensor.example.com/temp

UI
GET: /{Thing.name}/ui/property/temp/
- Returns a basic HTML5 UI component

Fig. 5. TD to DD - TempSensorWithUI

Listing 1.1 shows the code generated for the Controller microservice. We use `Java` even though the process can be extended to any other programming language. The code utilizes of a set of properties (lines 3 and 5), first the `env` variable has the configuration of the microservice, with parameters such as the Thing Description of the represented device, ports, Discovery Service address, and parameters related to the database connection, among others. Secondly the `propertiesMongoRepository` (line 5) includes all the properties found in the related device, in this case the temperature. Furthermore, we can see the two methods generated (lines 7–14 and lines 15–22) following the specification defined in Sect. 3. The first method returns the TD managed by the DD. The second method recovers the temperature and generate a response with the default content type (`application/json`) for the user. The other microservice will make use of this last method to generate the UI.

```java
1   public class Controller {
2   @Autowired
3   private Environment env;
4   @Autowired
5   private PropertyRepository propertiesMongoRepository;
6   @GetMapping("/TempSensorWithUI",produces="application/json")
7   public ResponseEntity returnTD() {
8       try {
9           String td=env.getProperty("td");
10          return new ResponseEntity(td, HttpStatus.OK);
11      } catch (Exception e) {
12          return new ResponseEntity(e, HttpStatus.INTERNAL_SERVER_ERROR);
13      }
14  }
15  @GetMapping("/TempSensorWithUI/property/temp",produces="application/json")
16  public ResponseEntity getPropertyTemp() {
17      try {
18          PropertyData temp=propertiesMongoRepository.findPropertyTemp();
19          return new ResponseEntity(temp, HttpStatus.OK);
20      } catch (Exception e) {
21          return new ResponseEntity(e, HttpStatus.INTERNAL_SERVER_ERROR);
22  }...
```

Listing 1.1. Controller generated code (TempSensorWithUI)

5 Related Works

Linking IoT devices with the Web is not a new idea. Authors like Guinard et al. [10] are working actively in make this a fact. They propose a Web of Things architecture and best-practices based on the RESTful principles. Our approach of DD tries to go a step further leveraging the latest trends in the Web Services architecture, *i.e.*, the use of microservices as a building block of our solution. As we explained, Digital Dices are closely related to the WoT [5], being this a reference framework that seeks to counter the gap found in the IoT world. The idea of using the WoT comes from the need of making our DD concept compatible with other systems and software, such as Mozilla WebThings [6], which offers a unifying application layer and links together multiple underlying IoT protocols using existing web technologies. The Digital Dices can be used as virtual devices in said system.

More closely related to our approach, the authors in [11] propose a solution based on Jolie and Java programming languages to manage a prototype platform supporting multiple concurrent applications for smart buildings. This proposal uses an advanced sensor network as well as a distributed microservices architecture. The solution has the caveat of being focused on a specific domain thus not really giving a broad solution to the management different devices.

Other proposals such as [12] and [13] offer a general solution to the use of microservices in a non-domain specific approach. The solution proposed by [12] makes use of 8 different microservices to separate aspects of an IoT centric systems, such as security, events, devices, etc. But, from our point of view, this solution does not really take advantage of the power of microservices. In contrast, the Digital Dice Architecture is a more fine grained solution, because our proposal handles each aspect of a device independently as a microservice.

At a higher level, Niflheim approach [14] proposes an end-to-end middleware for a cross-tier IoT infrastructure that provides modular microservice-based orchestration of applications to enable efficient use of IoT infrastructure resources. We see Niflheim as a complementary system, since it could provide a reliable architecture for the deployment of our Digital Dice Architecture. The IoT-A project [15] is an Architectural Reference Model (ARM) that establishes a common understanding of the IoT domain and provides to developers a common technical foundation and a set of guidelines for deriving a concrete IoT architecture. This ARM is taken into account in both the WoT and the Digital Dice architectures to establish the communication between different sets of devices.

6 Conclusions and Future Work

The aim of this proposal is to improve interoperability, integration and management between both real and virtual IoT systems and devices. To do this, the functionality of IoT devices will be abstracted to a set of microservices making use of the standards set by the WoT.

The use of microservices architectures allows us to establish choreography mechanisms that take into account the requirements of the system, permits a better use of resources and facilitates the maintenance. This article offers a solution that let us establish a way to convert external IoT devices described as a WoT Thing Description into Digital Dices. At the same time, Digital Dices offer a Thing Description themselves so they can be used by external systems seamlessly. The example scenario was designed to understand conversion process.

There are multiple advantages when using Digital Dices. Being a software abstraction let us define a common communication language no matter which device our Digital Dice is representing, thus defining a common pattern to connect to features defined by said device. The internal behaviour of our Digital Dice lessens the number of request received by the device, in some cases there is actually no need for us to connect with the device when we want to recover a property value. Our system, by definition, is based on microservices, this allows us to replicate only the facets of the system that receive more requests, this give our system flexibility, thus always trying to minimize the use of resources. Another advantages is the compatibility of our solution with the WoT definition

Schema, this offers other systems like the Mozilla IoT Gateway the possibility of making use of the schema defined by our Digital Dice to interact with.

The main disadvantages of using Digital Dices are mainly inherent to the use of microservices. First, the architectural complexity that microservices usually requires. It is easier to develop a monolithic application than a software based in microservices. Furthermore this kind of software requires outside gateways, discovery and other auxiliary software to choreograph the communication inside our system. Besides this fact, in some circumstances DD can be slower to respond than direct connection to IoT devices but usually more reliable.

For future work, we want to develop Digital Dices with different programming languages so we can analyze which ones offer a better performance. Comparing the Digital Dice performance and reliability with IoT devices and other software solutions is also one of the next steps of our further work. The possible extension of the WoT Thing description model to better suit our concept of Digital Dice is our next research objective, especially trying to define complex events in the model and keeping the compatibility of the Thing Description of W3C.

References

1. Postscapes: IoT Standards and Protocols. https://bit.ly/2iAWbky. Accessed 24 May 2019
2. Ngu, A., Gutierrez, M., Metsis, V., Nepal, S., Sheng, Q.: IoT middleware: a survey on issues and enabling technologies. IEEE Internet Things J. **4**(1), 1–20 (2016)
3. Shetty S.: How to Use Digital Twins in Your IoT Strategy. https://gtnr.it/2FFU4al. Accessed 24 May 2019
4. Tuegel, E., Ingraffea, A., Eason, T., Spottswood, M.: Reengineering aircraft structural life prediction using a digital twin. Int. J. Aerosp. Eng. **2011**, 14 p, (2011). Article ID 154798
5. W3C: Web of Things. https://www.w3.org/WoT/. Accessed 28 May 2019
6. Mozilla Foundation: Mozilla IoT Web of Things. https://iot.mozilla.org/. Accessed 28 May 2019
7. Angsuchotmetee, C., Chbeir, R.: A survey on complex event definition languages in multimedia sensor networks. In: Proceedings of the 8th International Conference on Management of Digital EcoSystems, pp. 99–108. ACM (2016)
8. Nadareishvili, I., et al.: Microservice Architecture: Aligning Principles, Practices, and Culture. O'Reilly Media Inc., Sebastopol (2016)
9. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3), 621–645 (2006)
10. Guinard, D., Trifa, V.: Building the Web of Things: With Examples in Node.js and Raspberry Pi. Manning Publications Co., New York (2016)
11. Khanda, K., Salikhov, D., Gusmanov, K., Mazzara, M., Mavridis, N.: Microservice-based IoT for smart buildings. In: 31st International Conference on Advanced Information Networking and Applications Workshops, pp. 302–308. IEEE (2017)
12. Long, S., Yan, L., Memon, R.H.: An open IoT framework based on microservices architecture. China Commun. **14**(2), 154–162 (2017)
13. Vresk, T., Čavrak, I.: Architecture of an interoperable IoT platform based on microservices. In: 39th International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 1196–1201 (2016)

14. Small, N., Akkermans, S., Joosen, W., Hughes, D.: Niflheim: an end-to-end middleware for applications on a multi-tier IoT infrastructure. In: IEEE 16th International Symposium on Network Computing and Applications (NCA), pp. 1–8. IEEE (2017)
15. Bauer, M., et al.: Internet of Things – Architecture IoT-A Deliverable D1.5 – Final architectural reference model for the IoT v3.0 (2013)