cloud.iO, An Open-source W3C WoT Compliant Framework

Lucas Bonvin[1], Dominique Gabioud[1] and Michael Clausen[2]
[1]*Institute of Sustainable Energy, HES-SO Valais-Wallis, Route du Rawil 47, Sion, Switzerland*
[2]*Institute of Systems Engineering, HES-SO Valais-Wallis, Route du Rawil 47, Sion, Switzerland*

Keywords: IoT, Interoperability, W3C, Web of Things, Thing Description, Cloud, cloud.iO.

Abstract: The Internet of Things (IoT) is gaining more and more popularity. It is therefore not surprising that the number of IoT cloud-based solutions grows accordingly. Most of these solutions have their own semantics and syntax, hence creating a heterogeneous landscape with a lack of interoperability. W3C works on the Web of Things (WoT) standardization with the goal of bringing interoperability across IoT solutions. This paper presents how the interoperability of the open-source IoT solution cloud.iO has been enhanced by making it compliant with the W3C WoT recommendations.

1 INTRODUCTION

The growth of IoT is characterized by an increased number of connected devices but also by the multiplication of the cloud platforms in charge of their connection. According to Postcapes[1], more than 152 platforms are available. To this IoT platforms list, we can add cloud.iO[2], an IoT cloud solution developed by the Institute of Systems Engineering at HES-SO Valais-Wallis.

The IoT platforms feature a similar structure:

- The current state of connected Things is mirrored in digital replicas called digital twins.
- The platforms allow applications to monitor and control Things through their digital twins.

The IoT platforms, however, use proprietary syntax and semantics to describe the Things' data model and to communicate with their twin. Hence, a given application is in practice bound to a single platform, as cross-platform applications need data conversion and support of multiple APIs. The resulting fragmentation of the IoT scene limits the development of IoT based services.

Interoperability between applications and platforms requires compliance with a homogeneous interface defined in a recognized standard. W3C, through its Web of Things (WoT) Working Group, has the ambition to elaborate such a standard. To this end, it has already published the architecture for interoperable platforms in a candidate recommendation.

This paper describes how cloud.iO has been completed to become compliant with the W3C's WoT architecture.

The structure of this paper is the following: Section 2 presents the concept of interoperability in an IoT environment; Section 3 describes the W3C WoT vision; Section 4 introduces cloud.iO; Section 5 explains how cloud.iO was upgraded to become WoT compliant. Finally, Section 6 concludes this paper with perspectives on future work.

2 INTEROPERABILITY

Interoperability is the ability of systems and devices to provide services to and use services from other systems or devices, so as to enable them to operate effectively together. IoT interoperability appears in many contexts like device to device, device to platform, platform to application, and application to application. In this paper, the ability for a compliant application to be interoperable with compliant platforms is addressed. The vision is to allow an application to address IoT devices connected through different platforms without ad hoc adapters.

Interoperability is defined at two levels:

- Syntactic Level: interoperable systems agree on the format and sequence of exchanged messages.
- Semantic Level: interoperable systems exchange

[1]https://www.postscapes.com/internet-of-things-platforms/

[2]https://github.com/cloudio-project

Bonvin, L., Gabioud, D. and Clausen, M.
cloud.iO, An Open-source W3C WoT Compliant Framework.
DOI: 10.5220/0009470604050411
In *Proceedings of the 5th International Conference on Internet of Things, Big Data and Security (IoTBDS 2020)*, pages 405-411
ISBN: 978-989-758-426-8

data with unambiguous, shared meaning. Semantic interoperability may apply to IoT specific concepts (e.g."event" or "action") or domain-specific concept (energy, health ...).

Both the W3C architecture and cloud.iO have support for domain-specific semantics. However, this aspect is not the focus of this paper.

The targeted interoperability is illustrated in Figure 1. In this example, two applications can interact with Things connected through three platforms because all components implement the digital twin interface. On the other hand, the two applications can interact together to share access to their Things through their digital twins. These interactions are possible since both applications understand the normalized syntax of the digital twins.

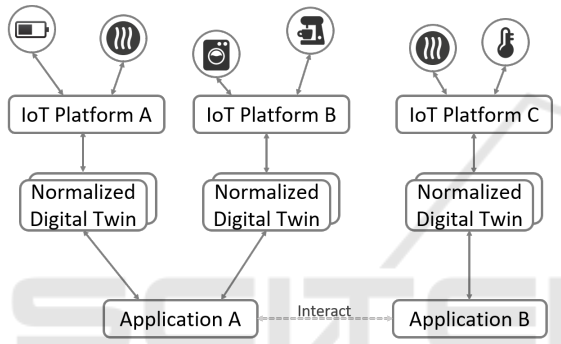

Figure 1: Interoperability between applications and IoT platforms.

Different projects have addressed the application - platform interoperability question. symbIoTe (Žarko et al., 2019) provides an abstraction layer to harmonize IoT platforms. Compliant platforms can register their resources to symbIoTe. Applications can then discover resources using the symbIoTe semantic search engine and later on access them in a homogeneous way. symbIoTe recognized the importance of standardization since its Core Information Model is based on the Semantic Sensor Network Ontology, a W3C recommendation, and on the OGC SensorThings API.

Another project proposing interoperability for IoT is BIG IoT (Jell et al., 2017). BIG IoT's goal is to build an IoT ecosystem with multiple IoT platforms, services, and applications through the definition of a uniform platform interface known as BIG IoT API. The BIG IoT API and its information models have been built in collaboration with the W3C's Web of Things Interest Group.

These projects and others paved the way for interoperability. However, large-scale deployment requires a stable interface definition backed by a recognized standardization body like W3C.

3 WEB OF THINGS

The W3C standardization work related to IoT is held by the Web of Things (WoT) Interest Group (WoT IG) and the Web of Things Working Group (WoT WG). The latter has released two candidate recommendations: the Web of Things (WoT) Architecture (Kovatsch et al., 2019) and the Web of Things (WoT) Thing Description (Kaebisch et al., 2019).

The WoT architecture is made up of four building blocks:

- The WoT Thing Description defines an information model for Things based on a semantic vocabulary and a serialized representation based on JSON.
- The WoT Binding Templates specify syntactic and protocol-related information for Thing access.
- The WoT Scripting API proposes a programming interface that allows applications to discover, fetch, consume, produce, and even expose WoT Thing Descriptions.
- The WoT Security and Privacy recommendation provides security-related guidelines for all components.

The main concepts of the WoT architecture are explained in this paragraph. A normalized digital twin is called a Thing and is defined as "the abstraction of a physical or virtual entity (...) described by standardized metadata". The latter are registered in a JSON formatted Linked Data document named Thing Description (TD). The entity interacting with a TD is called a Servient. It is defined as a software component that implements one or many WoT building blocks. Servients can be Consumers, which are entities "that can process TDs (...) and interact with Things". TDs allow Consumers "to identify what capabilities a Thing provides and how to use the provided capabilities" (Kovatsch et al., 2019). Interaction Affordances link the "what" and the "how" for each capability. "Whats" are named Interactions, whereas "hows" are Protocol Bindings following WoT Binding Templates.

3.1 Things Description (TD)

A TD is basically a collection of Interaction Affordances. Three types of Interactions suffice to model nearly all Interactions found in IoT devices and services:

- A Property describes a state of a Thing. A Consumer can read it and possibly also write it.

- An Action refers to a function on a Thing. Invoking an Action can change the state of a Thing.
- An Event describes an asynchronous data exchange initiated by a Thing.

The following use case illustrates how a TD provides access to a Thing representing a smart heating system with ambient temperature measurement and a temperature set point. In the TD, both the real and target temperatures are Properties that can be accessed by a hypothetical HTTP operation. The heating regulation can be switched on and off. This is represented by an Action in the TD. Finally, our heating appliance can detect and announce changes in the ambient temperature. This is transposed as an Event in the TD. Listing 1 represents a basic TD implementation for this smart heating system.

```
{
  "@context": "https://www.w3.org/2019/wot/td/v1",
  "id": "urn:heater",
  "title": "myHeater",
  "securityDefinitions": {
    "basic_sc": {
      "scheme": "basic",
      "in": "header"
    }
  },
  "security": ["basic_sc"],
  "properties": {
    "temperature": {
      "type": "integer",
      "forms": [{
        "href": "https://myHeater.com/get"
      }]
    },
    "targetTemperature": {
      "type": "integer",
      "forms": [{
        "href": "https://myHeater.com/set"
      }]
    }
  },
  "actions": {
    "toggleRegulation": {
      "forms": [{
        "href": "https://myHeater.com/toggle"
      }]
    }
  },
  "events": {
    "temperatureChange": {
      "data": {"type": "integer"},
      "forms": [{
        "href": "https://myHeater.com/change",
        "subprotocol": "longpoll"
      }]
    }
  }
}
```

Listing 1: TD example of a heating system.

Apart from the Interaction Affordances, a TD must contain four mandatory JSON objects all present in Listing 1: `@context`, `title`, `security`, and `securityDefinitions`. `@context` is a JSON-LD specific name whose corresponding value refers to the TD ontology definition. The `title` field provides a human-readable title to the Thing. Finally, the `security` object refers to `securityDefinitions` defined in the TD. To access a resource, the conditions of the mentioned security definition must be fulfilled.

4 cloud.iO

cloud.iO is a scalable open-source IoT cloud-based platform developed and maintained by the Institute of Systems Engineering at HES-SO Valais Wallis. cloud.iO has been used in several projects, including the European FP7 SEMIAH and at the European Horizon 2020 GoFlex projects (Roduit et al., 2019) for monitoring and control of energy and heating in hundreds of buildings.

cloud.iO provides the following services:

- Property Access: Things properties can be read and written using different models.
- Directory: The list of Things with their information model can be browsed and searched.
- Historian: The history of Things' properties is made available.
- Remote Job Execution: Things local tasks can be started remotely.

cloud.iO also lets Things owners define privacy rules for their Things down to the property level.

4.1 Roles

cloud.iO defines two roles: Endpoint and User.

An Endpoint is an IoT device hosting one or more Things. Each Endpoint has its own information model, published, stored, and made available in the cloud. An Endpoint sends messages to the cloud upon status changes of hosted Things. Conversely, Things' properties can be remotely modified by incoming messages. Every change on Things properties is stored in a time-series database.

Users are Endpoints owners. In the cloud.iO ecosystem, a User is indifferently a real person or an application. A User has full access on its Things' properties and may delegate read or read/write access for a given property to another User.

Users may:

- Browse and search a Thing information model directory,
- Monitor and control Endpoints in real-time,
- Access Endpoints' past states (historian) through filtering, and
- Request the execution of jobs on Endpoints.

4.2 Architecture

The core of cloud.iO is composed of two open-source frameworks: the application for micro-services Spring[3] and the message broker RabbitMQ[4]. cloud.iO is made up of a set of independent, mostly stateless, low-complexity micro-services interacting through the message broker (see Figure 2). These architectural choices - message-driven processing and micro-services - make cloud.iO a simple, scalable, and extensible IoT platform.

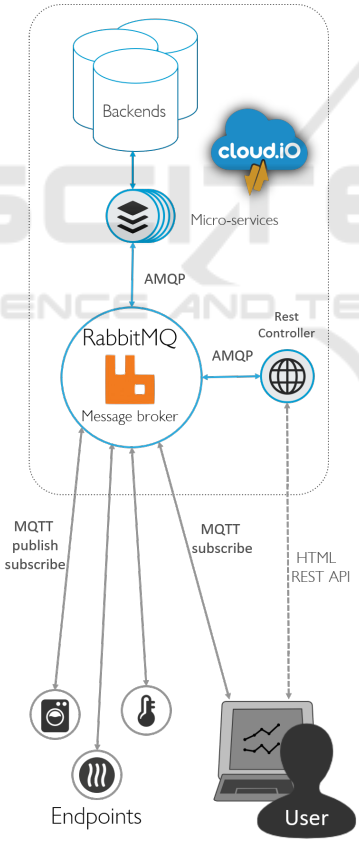

Figure 2: cloud.iO complete architecture.

Micro-services implement the following functions: database access, Endpoints and Users management, privacy rules definition, and REST services implementation.

Endpoints establish a TLS-secure MQTT link with the message broker. X509 certificate-based Endpoint authentication is mandatory. High-level Java and Python Endpoint libraries are provided for efficient Endpoint development.

Users can interact with Things either through messaging (AMQP, MQTT) or through a RESTful API.

4.3 Endpoint Data Modeling

Data modeling in cloud.iO is based on three concepts:

1. An **Attribute** represents an atomic property of a Thing (e.g. a numerical value).
2. An **Object** is a collection of Attributes forming a coherent unit (e.g. an analog measurement made up of Attributes for numerical value, engineering unit, quality...).
3. A **Node** is a collection of Objects modeling a subsystem of an Endpoint.

The cloud.iO data model is represented in the UML class diagram of Figure 3.

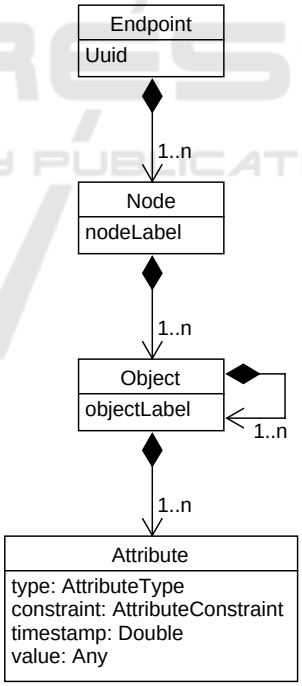

Figure 3: cloud.iO class diagram for an Endpoint.

cloud.iO Objects in an Endpoint have a tree-shaped structure, with Attributes as leaves.

The elements composing a Node or an Object can be either freely defined or comply with a semantic data model specific to an application domain.

[3]https://spring.io
[4]https://www.rabbitmq.com

A cloud.iO Attribute contains four fields. The Type field indicates the format of the Value field by reference to a JSON type (Boolean, integer, number, string). Values are associated with a Timestamp in epoch format. Finally, the Constraint field defines the category of the Attribute, and indirectly its read / write access mode. A Constraint may take one of the five following value:

- Static: Read-only, static value.
- Status: Read-only, computed value.
- Measure: Read-only, measurement result acquired by a sensor on a physical process.
- Parameter: Read/write, configuration parameter acting on the Endpoint's behavior. Persists after reboot.
- SetPoint: Read/write, target value for an actuator, non-persistent.

Endpoints feature globally unique identifiers (UUIDs). Nodes, Objects, and Attributes have each a label with a local scope. Consequently, each Attribute can be referred to by a full label obtained by the concatenation of the names of its containing elements.

Listing 2 presents the cloud.iO data model for a Node modeling the above-described heating system. The `myHeater` Node contains a single Object (`temperatures`) with two Attributes representing the instantaneous ambient temperature (Measure) and the target temperature (SetPoint).

```
{
  "myHeater": {
    "implements": [],
    "objects": {
      "temperatures": {
        "conforms": null,
        "objects": {},
        "attributes": {
          "temperature": {
            "constraint": "Measure",
            "type": "Integer",
            "timestamp": 1.57476099056E9,
            "value": 25
          },
          "targetTemperature": {
            "constraint": "SetPoint",
            "type": "Integer",
            "timestamp": 1.57476098001E9,
            "value": 24
          }
        }
      }
    }
  }
}
```

Listing 2: Digital twin of cloud.iO Node.

4.4 cloud.iO Operation

Attributes can be classified in two overlapping categories: readable Attributes and writable Attributes.

An Endpoint monitors its readable Attributes locally. Any significant change of a value triggers the emission of an MQTT message with a topic containing the Attribute's full label and a message body made up of the JSON representation of the Attribute's fields. Conversely, a write operation on an Attribute is triggered by the reception of an MQTT message with a similar structure.

Users can perform read operations on Attributes (HTTP) and write operations on Attributes (MQTT, HTTP). They can also subscribe to Attributes updates and be notified on Attribute changes (MQTT, HTTP long poll).

5 cloud.iO AND WEB OF THINGS

The cloud.iO data model is flexible enough to represent the digital twin of any Thing. However, as it is specific to cloud.iO, applications (Users) must also be cloud.iO specific. Hence application - platform interoperability as defined in Section 2, is not ensured. To open the cloud.iO platform to WoT compliant applications, cloud.iO data models must be translated into WoT TDs. Figure 4 illustrates a micro-service hosting and exposing a WoT Thing for an application acting as Consumer.

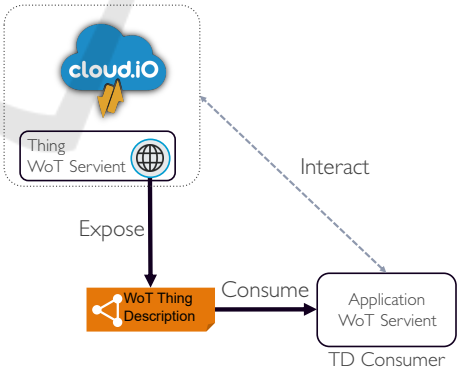

Figure 4: Interaction between cloud.iO and a Consumer.

The core of the translation from cloud.iO data models to WoT TD is the implementation of the different Interactions Affordances. Those are built according to the following guidelines:

- A cloud.iO Node is mapped to a TD Thing.
- A cloud.iO Attribute is mapped to two TD Interactions: one of type Property and a second of type Event.

- A TD Consumer performs a read or write operation on a cloud.iO Attribute through its Property representation.
- A TD Consumer is notified of a Attribute change through its Event representation.

Listing 3 presents the Property Affordance related to the `targetTemperature` Attribute.

```
{
...
  "properties": {
  ...
    "property-endpointUUID.myHeater.
    ↪ temperatures.targetTemperature": {
        "type": "object",
        "properties": {
          "constraint": {
              "type": "string",
              "enum": ["SetPoint"]
          },
          "type": {
              "type": "string",
              "enum": ["Integer"]
          },
          "timestamp": {"type": "number"},
          "value": {"type": "number"}
        },
        "required": ["constraint", "type", "
    ↪ timestamp", "value"],
       "forms": [{
          "href": "https://cloud.iO:8081/api/
    ↪ v1/getAttribute/endpointUUID.myHeater.
    ↪ temperatures.targetTemperature",
          "op": "readproperty",
          "contentType": "application/json"
        },
        {
          "href": "mqtts://cloud.iO:8883/@set/
    ↪ endpointUUID/myHeater/temperatures/
    ↪ targetTemperature",
          "op": "writeproperty",
          "contentType": "application/json"
        },
        {
          "href": "https://cloud.iO:8081/api/
    ↪ v1/setAttribute/endpointUUID.myHeater.
    ↪ temperatures.targetTemperature",
          "op": "writeproperty",
          "contentType": "application/json"
        }]
      }
    }
  },
  ...
}
```

Listing 3: Property Affordance of heating System target temperature.

Listing 4 shows the Event Affordance (linked to the `targetTemperature` Attribute).

```
{
...
  "events": {
    ...
    "event-endpointUUID.myHeater.temperatures.
    ↪ targetTemperature": {
      "data": {
        "type": "object",
        "properties": {
          "constraint": {
              "type": "string",
              "enum": ["SetPoint"]
          },
          "type": {
              "type": "string",
              "enum": ["Integer"]
          },
          "timestamp": {"type": "number"},
          "value": {"type": "number"}
        },
        "required": ["constraint", "type", "
    ↪ timestamp", "value"]
      },
      "forms": [{
        "href": "https://cloud.iO:8081/api/v1/
    ↪ notifyAttributeChange/endpointUUID.
    ↪ myHeater.temperatures.targetTemperature
    ↪ ",
        "op": "subscribeevent",
        "subprotocol": "longpoll",
        "contentType": "application/json"
      },
      {
        "href": "mqtts://cloud.iO:8883/@set/
    ↪ endpointUUID/myHeater/temperatures/
    ↪ targetTemperature",
        "op": "subscribeevent",
        "contentType": "application/json"
      }]
    }
  }
}
```

Listing 4: Event Affordance of heating System target temperature.

Generating a TD from a cloud.iO data model is a four-step process:

1. One Thing per cloud.iO Node is created, with the Node's full label as TD field `id`.
2. Two Interaction Affordances per cloud.iO Attributes are created. Both contains the Attribute's full label in their key of Interaction Affordance map.
3. The (unique) schema for an Attribute is associated to each Interaction.
4. cloud.iO Attributes access definition, which is implicit in cloud.iO data models, is made explicit as WoT Protocol Bindings.

Native and TD based cloud.iO services are equivalent. Note that cloud.iO was made WoT compliant just by translating its native data model into a TD. Access to Endpoint resources did not require changes as it could be expressed by legacy TD Interaction Affordance.

6 CONCLUSION

In this article, we presented how cloud.iO has been upgraded to become a platform compliant with the W3C Web of Thing Candidate Recommendations. We applied the principle of a WoT Servient to cloud.iO by implementing one of the WoT building blocks: the Thing Description.

The implementation consisted of an automated translation of its native data model to a WoT TD document: cloud.iO Attributes were converted into TD Interaction Affordances and, by doing so, exposing the cloud.iO API. This straightforward conversion highlights the fact that an existing IoT cloud-based platform can be retrofitted with a limited effort.

6.1 Future Work

First, new versions of the still evolving WoT candidate recommendation will be implemented in the cloud.iO platform, thus turning it into a test and validation platform for the WoT architecture. Secondly, we intend to deploy an interoperability test bed where applications interact with several WoT compliant IoT platforms. Finally, the TDs will be complemented with an application level ontology like SAREF[5].

REFERENCES

Žarko, I. P., Mueller, S., Płociennik, M., Rajtar, T., Jacoby, M., Pardi, M., Insolvibile, G., Glykantzis, V., Antonić, A., Kušek, M., and Soursos, S. (2019). The symbiote solution for semantic and syntactic interoperability of cloud-based iot platforms. In *2019 Global IoT Summit (GIoTS)*, pages 1–6.

Jell, T., Bröring, A., and Mitic, J. (2017). Big iot – interconnecting iot platforms from different domains. In *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 86–88.

Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., and Kovatsch, M. (2019). *Web of Things (WoT) Thing Description*. Retrieved November 25, 2019, from https://www.w3.org/TR/wot-thing-description/.

[5]https://ontology.tno.nl/saref/

Kovatsch, M., Matsukura, R., Lagally, M., Kawaguchi, T., Toumura, K., and Kajimoto, K. (2019). *Web of Things (WoT) Architecture*. Retrieved November 25, 2019, from https://www.w3.org/TR/wot-architecture/.

Roduit, P., Gabioud, D., Basso, G., Maitre, G., and Ferrez, P. (2019). cloud.io: A decentralised iot architecture to control electrical appliances in households. In Donnellan, B., Klein, C., Helfert, M., and Gusikhin, O., editors, *Smart Cities, Green Technologies and Intelligent Transport Systems*, pages 67–89, Cham. Springer International Publishing.