Dynamic and Interoperable Control of IoT Devices and Applications based on Calvin Framework

Fernanda Famá, Cleuves Cajé de Carvalho, Danilo F. S. Santos, Angelo Perkusich, Kyller Gorgônio

Embedded Lab, Federal University of Campina Grande, Campina Grande, Brazil
{cleuves.carvalho, fernanda.fama, danilo.santos, perkusic, kyller}@embedded.ufcg.edu.br

Abstract

The development of tools and smart devices has grown exponentially over the past few years, most due to the appearance of the Internet of Things (IoT). The large number of different connected devices highlighted challenges such as interoperability, scalability and reliability. In this scenario, for the development of new services and applications, it is necessary to use software platforms that ease the deployment and validation based on those challenges. With this, arises the need for middlewares, platforms that provide an environment for the development of such applications. In this way, in order to provide a simple, lightweight and dynamic approach, this article presents a study and development of an architecture that allows the communication, control and monitoring of IoT devices using an intuitive manner through an actor model. This is possible through the integration of the Calvin framework, the MQTT protocol and the OCF data model, providing an interoperable, reliable, dynamic and remote communication. A smart home environment was used for validation showing the relevance of the proposal.

Actor model; dynamic control; interoperability; IoT; MQTT; OCF; smart home

1 Introduction

Internet of Things (IoT) is described as a network infrastructure with interoperable communication standards and protocols where physical or virtual *"things"* share information in real time [22]. Aiming to make the Internet more comprehensive and immersive, IoT enables access to a wide range of devices. Those devices are used to build applications in many domains, including industrial and home automation, intelligent cities and healthcare [24].

Due to the increasing interest in IoT application, a wide variety of devices and objects that aims to provide utilities and services through the Internet are under development. This variety of devices and services creates new challenges, which demands the use of different architecture models in terms of Software Engineering.

The main challenges to achieve the maximum potential of IoT are interoperability, mobility, scalability, performance, security and privacy. Because of the heterogeneity of devices and platforms, interoperability is a major obstacle to be overcome. Standardization of protocols and pattern interpretations are important to make all devices accessible and interoperable. Therefore, data exchange between a large number of devices must be managed so that sensitive data is not compromised [1, 10].

Middlewares are used to allow the usage of heterogeneous components and to abstract implementation details of network protocols and communication resources. A middleware works as a communication link between devices and applications for IoT. It should provide interoperability, device discovery and management, security, privacy, and also high-volume data management [2, 15].

In this paper, we propose a new IoT System architecture model, based on an actor-based middleware, that is interoperable, dynamically controllable and manageable. This new system is built on top of Calvin [21] middleware, MQTT communication protocol, and oneIoTa models based on the OpenConnectivity Foundation (OCF)[1] specifications. While Calvin was used for the development, deployment and execution of an IoT smart home application, MQTT and oneIoTa provided interoperability and standardization between different devices. As a validation scenario, we built a smart-home application, which, based on the user behaviour, demands ways to dynamic control remote devices in a interoperable way.

The paper is structured as follows. Section 2 discusses the related work. Section 3 introduces the tools used for

DOI reference number:10.18293/SEKE2019-177

[1] http://www.openconnectivity.org

the IoT application. Specifically, the Calvin framework that supported the entire implementation, the MQTT protocol and the OCF "consortium" are described. Section 4 details the proposed architecture and the dynamically controlled smart home application. The implementation and validation of the proposed system are presented in Section 4. Finally, conclusions and future works are presented in Section 5.

2. Related Work

Several studies were executed to design, implement and control IoT systems. Konduru et al [11] presents a study that identifies challenges and solutions considering interoperability of devices. Tools such as Google Weaver, IoTivity, AllJoyn and Apple Home Kit are analysed. Google Weaver and the Apple Home Kit are focused on solving interoperability only for pre-defined devices. While AllJoyn is a framework that together with IoTvity are part of the Open Connectivy Foundation (OCF).

Belsa et al [3] also address the interoperability problem, but with a different approach. IoT platforms are integrated through a flow-based model. The proposed architecture is based on the Node-RED[2] middleware. However, unlike Calvin, that uses a distributed hash table (DHTs), it does not support the development of applications with distributed systems. [20], also uses Node-RED to design, deploy and control devices remotely in a smart laboratory.

Finally, [12] implements a Healthcare resource model using OCF and IoTivity platform. In this work, two OCF standards are used, the OIC Healthcare Resource and the OIC Healthcare Device. Some of these resource are blood glucose, body metrics, heart rate and oxygen saturation sensors. In which, each resource has a schema and RAML.

3 Basic Concepts

In this section we present the main technologies used in this work.

3.1 Calvin

Calvin is an open source framework aiming to ease the development of IoT applications in distributed systems. It is based on a data flow programming methodology through an actor model [14, 16]. The paradigm of data flow programming is present as an actor model. An actor is a software component that models functions, devices, services or some type of computing in the form of an object. In Calvin, tokens are used to establish communication between actors. These tokens are created when the input actors captures and processes data. Data is them transformed into resources, which in this environment are considered tokens, that will be consumed by other actors [18].

[2]Node-RED: `https://nodered.org/`

The framework has several standard actors and sensors responsible for input/output of data, media and network access (including HTTP, TCP and MQTT), among others [21]. New actors can be implemented in Python and the dataflow between them is expressed using a declarative language called CalvinScript [19]. Because of the ease of developing actors, Calvin simplifies the implementation of several new services based on this type of model.

To develop an application in Calvin it is necessary to follow a cycle consisting of four phases: describe, connect, deploy and manage. Firstly, the developer will describe how a task is executed by each actor. Secondly, it is necessary to make the connections between them. This can be done using the Calvin GUI. This tool lists the available actors and allows to connect them through their inputs and outputs. When performing this, the CalvinScript is automatically generated. Figure 1 shows a detail of Calvin's graphical interface. After creating connections, the application is already ready to deploy. Finally, during the management phase, modifications can be made in the application and Calvin executes the deployment [14, 16].

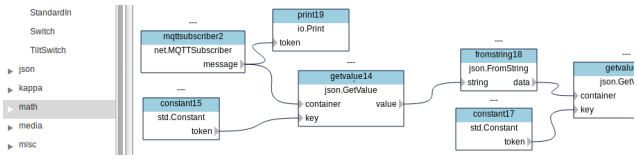


Figure 1. Calvin GUI.

In Calvin's architecture there is a layer called runtime where actors are executed. It has two sub-layers, platform dependent and platform independent. The first allows communication between several runtimes through the most varied communication protocols, such as WiFi and Bluetooth. The second, allows the execution of several actors at one runtime only. Calvin GUI works on that layer.

3.2 MQTT

Message Queue Telemetry Transport (MQTT) is an application layer protocol that centralizes the sending and receiving of data from applications using a wireless sensor network, enabling the communication between devices with limited power source. This architecture is based on a Publisher/Subscriber system. Publishers are responsible for sending data collected through sensors, while subscribers consumes the data that was collected [9].

A Broker is used to coordinate the sending and receiving of data in order to ensure that the communication will be reliably. The Broker receives the data collected and classifies the data as topics. The data is then sent to devices interested in a specific topic. The broker also ensures that the data will be received only by the subscribers who will consume them. To provide safety and quality of data com-

munication, MQTT uses encryption and login to guarantee a minimum level of quality of service (QoS) [13].

The MQTT also allows the creation of a development environment for IoT applications. This is due to its high integration capability with several IoT development platforms, as it enables the development of services using various programming languages, such as Python, Java, JavaScript, PHP, Ruby and C. Also, MQTT allows extending these services to mobile platform such iOS and Android [6].

3.3 OCF and oneIoTa

Open Connectivity Foundation[3] (OCF), is an IoT consortium that aims the specifications, open source and certifications needed for the development of an IoT environment with interoperability between devices that acts as OCF Clients and Servers [4, 17]. The exchange of information between devices uses the JSON format, following the specifications for identifying them and their connected resource, which makes it easier to validate the data structure [5]. To describe OCF services the specifications of the resources are made using a descriptive language of RESTful APIs, the RAML. To ease the implementation of the OCF data model, the consortium has developed the open tool oneIoTa[4] that has several examples of type RAML, Swagger and JSON Schema models. Users can also create their own templates that, if approved, are stored in a GitHub repository.

4 Proposed Solution

As stated before, we are motivated by the following aspects: interoperability; remote access and dynamic configurations. Interoperability is a critical problem in IoT. Application developers should consider providing services to all clients regardless of the hardware platform specifications they use. Integration with different communication devices is necessary so that new functions can be added to the application without compromising existing functions or even losing them [1].

Devices and appliances must also allow remote access in order to be monitored and controlled through a computer or a smartphone [23]. In security applications remote access is indispensable because it allows the user to monitor their home when they are traveling for example.

Finally, IoT applications must allow dynamic configuration. Most of IoT applications developed today were created to perform pre-defined tasks, for that reason they are static systems limited to certain actions. Dynamic configuration makes the system more elastic and comprehensive because the user can configure the applications to handle devices and sensors that will be added later. In addition, new monitoring and control applications can be created and modified at the users discretion.

To achieve the above goals and to facilitate the implementation of the system, we are looking for an IoT middleware that provides the best possible solution. The existing IoT architectures are divided into three types of classes: service-based; cloud-based; and actor-based. Service-based architectures and actors provide interoperability, they support a specific programming model or device abstraction. However, the service-based model provides limited functionality for the user when it comes to integrating with other applications or even interpreting data. The cloud-based model provides interoperability through specific standards, which is not desired, and the middleware can stop if the cloud provider terminates the service [15]. For these reasons, the actor-based model is the best solution for the smart home system we propose.

An actor-based architecture provides a better way of dealing with large-scale IoT devices, since middleware can be deployed across all layers of the architecture, so devices can perform actions where it is more adequate. Therefore, an actor-based middleware, such as Calvin and Node-RED, is a good choice in applications involving a large number of "things". In addition to having features such as interoperability, security, and privacy, where users can choose the form and location where the data will be stored [15].

The proposed architecture involves a variety of devices and technologies. The first part consists of sensors, for example temperature sensors, present in wireless nodes. These nodes are connected to a gateway unit which, in turn, is responsible for sending the sensor data, through messages, to the MQTT Broker. The gateway must then be equipped with the Calvin framework with a Publish application that sends the messages with a specific topic for the sensor identification. Calvin Constrained is a good choice in this case due to the limitation of some devices.

In Figure 2 a representation of the logical diagram of the proposed system can be seen. This diagram provides a view of the connection and communication between the architecture elements. Each component behaves as follows:

1. The Middleware receives the data sent by the sensors with its information, and modifies that data and transforms it into a JSON object. This object is a sequence of key/value pairs. A key must be a string, and the value must be a JSON base type;

2. The Gateway is a MQTT client that connects to the cloud server through a TCP/IP connection. Once connected the gateway can publish the data of sensors and devices for the broker to distribute to the controller and also subscribe to the data published by the broker to the devices. The broker acts as an intermediary between the gateway and the controller.

3. The Controller application subscribes to the sensors

[3]OCF: `https://openconnectivity.org/`
[4]oneIoTa: `https://oneiota.org/`

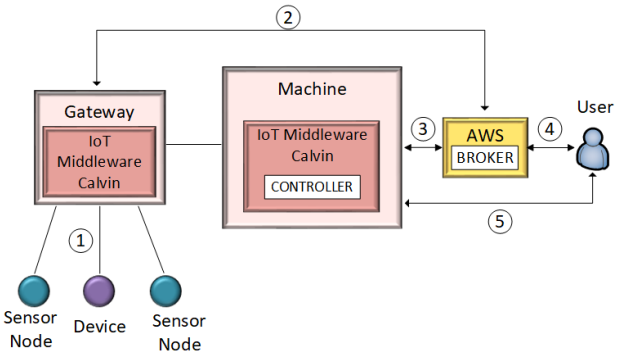


Figure 2. Architecture of the proposed system.

and devices data coming from the Gateway through the cloud, performs the control action and publishes the new data to the device;

4. The user has direct access to the cloud server through the platform on the Internet, and can manage the Broker data;

5. The user can access the application controller directly from the local machine or via remote access.

In order to exchange MQTT messages, a Broker, which is implemented on a cloud platform, is required. There are several free and paid cloud platforms available with support for gateways, services, and application protocols such as REST, COAP, XMPP, and MQTT [1]. Eclipse Mosquitto was chosen for the application because it is a Broker that provides a lightweight server implementation of the MQTT protocol and, moreover, it is open source (EPL/EDL licensed) [7, 8].

Finally, the control module provides the system-wide management through dynamic configuration. The application is implemented in Calvin GUI, which is a web-based GUI for Calvin application development. To access the GUI platform the user must have the Calvin runtime running on a local machine or on a machine on the same network. The dynamic configuration of this application is done in a simple way, given the ease in changing the actors and their data flows in the GUI platform. Therefore, you can at any time interrupt the application's data flow and modify it according to your needs.

4.1 Validation

A smart home application was implemented using the proposed architecture to validate the proposed solution. Figure 3 provides a visual illustration of the technologies and how they connect with each other. The application consists of a system that collects sensor data and transmits it to the controller. This data is used to monitor and to control the actuator device dynamically and remotely. The use case consists of performing the temperature control of environments by means of an air conditioning device and sensor nodes.

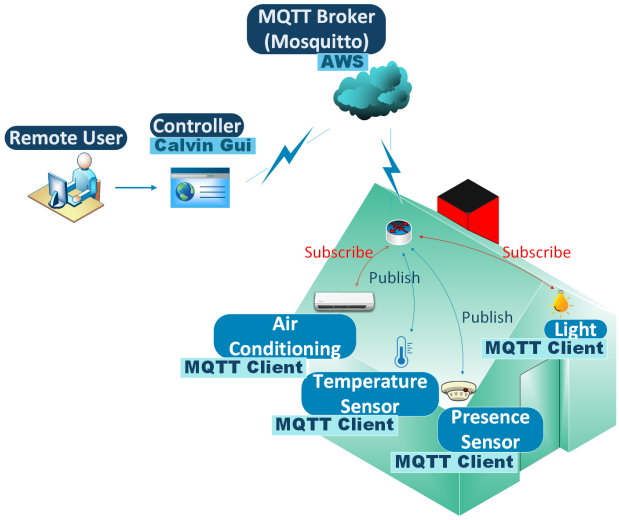


Figure 3. Smart home system overview.

A test environment consisting of four (4) Linux based Virtual Machines running Calvin the framework. Two of these machines were used to simulate an environment of a residence with their devices, i.e. the air conditioners. The other two machines were used to simulate the temperature sensors distributed in the environments. In Figure 4 the sequence of messages of two different simulated environments, living room and room, can be observed.


```
{'topic': u'ocf/rt/sala/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
"rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 22, "id": "tempe
ratura"}'}
{'topic': u'ocf/rt/sala/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
"rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 22, "id": "tempe
ratura"}'}
{'topic': u'ocf/rt/quarto/ar/oic.r.TemperaturaArCondicionado', 'payload':
'{"rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 20, "id": "tem
peratura"}'}
{'topic': u'ocf/rt/quarto/ar/oic.r.TemperaturaArCondicionado', 'payload':
'{"rt": ["oic.r.TemperaturaArCondicionado"], "temperatura": 20, "id": "tem
peratura"}'}
{'topic': u'ocf/rt/quarto/switch/oic.r.SwitchBinary', 'payload': '{"r": "o
ic.r.SwitchBinary", "id": "switch", "value": 1}'}
{'topic': u'ocf/rt/quarto/switch/oic.r.SwitchBinary', 'payload': '{"r": "o
ic.r.SwitchBinary", "id": "switch", "value": 1}'}
{'topic': u'ocf/rt/sala/ar/oic.r.TemperaturaArCondicionado', 'payload': '{
```


Figure 4. Sequence of messages between entities.

To validate the interoperable feature, it was used the oneIoTa specification for OCF devices[5]. Table 1 details the specifications of air conditioner devices. As can be observed in Figure 4, the payload of the devices are in accordance with the recommendations of the OCF, but as a

[5]OCF Device Specification: `https://openconnectivity.org/specs/OCF_Device_Specification_v2.0.0.pdf`

differentiation between the requirements of the sensor and the air conditioner, the air conditioner was modified to facilitate the understanding of reader. These specifications, as mentioned in 3.3, enables interoperability between devices. OCF defines the resource model that provides consistency among the devices in the home. This model uses the system of resources and devices, whose features exchanged are of various types, in the case of air conditioner has the type binary and the temperature. Each resource type defines a set of properties that are defined using the JSON format.

Table 1. OCF Device Resources.

Device Name	Required Resource name	Required Resource Type	RAML/JSON example
Air Conditioner	Binary Switch	oic.r.switch.binary	{ ”rt”: [”oic.r.switch.binary”], ”id”: ”unique_example_id”, ”value”: false }
	Temperature	oic.r.temperature	{”rt”: [”oic.r.temperature”], ”id”: ”unique_example_id”, ”temperature”: 20.0, ”units”: ”C”, ”range”: [0.0,100.0]}

To fulfill the remote access feature, Calvin GUI was used to perform updates remotely. The framework allows the control to be remotely carried out on the machine where Calvin is installed or on another machine on same network. In addition, the inclusion of a virtual machine in the cloud, used as a broker running the MQTT server, allows the user to subscribe to a topic of interest. The data published in this topic is sent to those who requested it. In Figure 4 it is possible to identify 3 different topics that the user can sign in. Figure 5 shows more clearly how the message exchange occurred in the application.

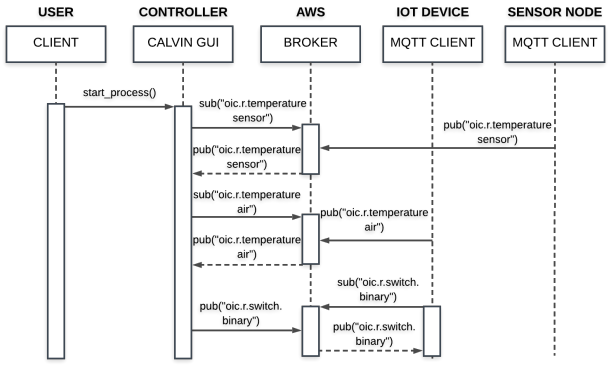


Figure 5. Subscriber topics in JSON format.

The user must initiate the process so that the controller requests the room temperature to the broker. The broker acquires this data from the sensor temperature and sends this value to the controller. The same occurs with the air conditioner. The control will determine whether the air conditioner stays on/off or modifies its state according to the data available and the user specification.

To achieve dynamic configuration, the user must be able to add new devices, modify the control model, and its parameters. Figure 6 shows a small part of the control implemented. Figure 6(a) shows a specific setpoint for the control of the switch of the air conditioner. In Figure 6(b) this setpoint was removed from the control and replaced in the Figure 6(c) (setpoint2). This sequence was executed in runtime, where the controller device was kept running during the changing of setpoints. Also, by the use of a standardized data model (oneIoTa), all remote clients were able to keep receiving updates without any interruption.

5. Conclusions and future work

This article presented an architecture that integrates several technologies, such as IoT protocols and frameworks, to provide a scalable and dynamic environment for the development of services to remotely control and monitoring devices such as, sensors and actuators. The whole architecture was developed following a model based on actors, which enabled a dynamic manipulation of components for IoT applications. As also, using a widely-used protocol, it was possible to establish the communication in a practical and intuitive way through a MQTT broker integrating with cloud services. A JSON based data model (oneIoTa) was used to ensure interoperability between devices. The approach has been successfully developed, as can be verified in its validation procedure. This infrastructure can be adapted to other environments, such as industrial, hospital or business, according to each user's need. In the future, we should include several other devices, providing a means of remote monitoring of these devices and their location in application domains.

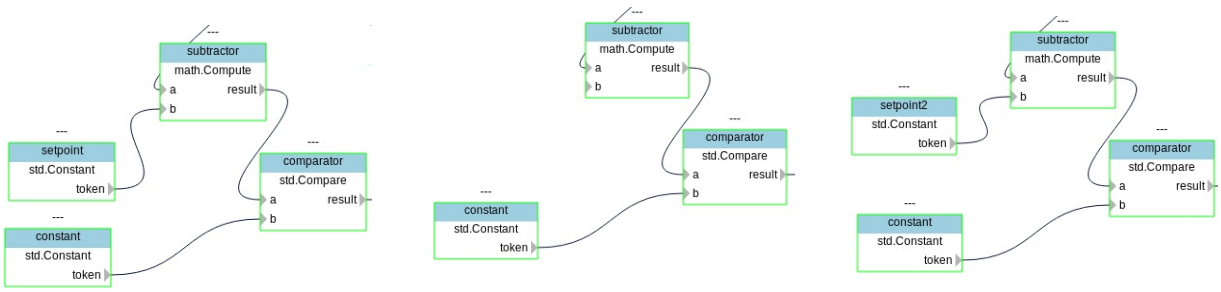
References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[2] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey*, 2(3):94–105, 2011.

[3] A. Belsa, D. Sarabia-Jacome, C. E. Palau, and M. Esteve. Flow-based programming interoperability solution for iot platform applications. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 304–309. IEEE, 2018.

[4] S. Cavalieri, M. G. Salafia, and M. S. Scroppo. Real-



(a) Initial temperature (b) Removed the block (c) Insert other block

Figure 6. Dynamic Control of Air Conditioner temperature.

ising interoperability between opc ua and ocf. *IEEE Access*, 6:69342–69357, 2018.

[5] S. Cavalieri and M. S. Scroppo. A proposal to make ocf and opc ua interoperable. In *Proceedings of ICIT*, volume 2018, 2018.

[6] M. Collina, G. E. Corazza, and A. Vanelli-Coralli. Introducing the qest broker: Scaling the iot by bridging mqtt and rest. In *Personal indoor and mobile radio communications (pimrc), 2012 ieee 23rd international symposium on*, pages 36–41. IEEE, 2012.

[7] I. Eclipse Foundation. *Eclipse Mosquitto: An open source MQTT broker*, 2018 (accessed August 23, 2018). `https://mosquitto.org/`".

[8] I. Eclipse Foundation. *Eclipse Mosquitto*, 2018 (accessed July 3, 2018). "`http://projects.eclipse.org/projects/technology.mosquitto`".

[9] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. Mqtt-sa publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE, 2008.

[10] R. Khan, S. U. Khan, R. Zaheer, and S. Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260. IEEE, 2012.

[11] V. R. Konduru and M. R. Bharamagoudra. Challenges and solutions of interoperability on iot: How far have we come in resolving the iot interoperability issues. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, pages 572–576. IEEE, 2017.

[12] J.-C. Lee, J.-H. Jeon, and S.-H. Kim. Design and implementation of healthcare resource model on iotivity platform. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 887–891. IEEE, 2016.

[13] S. Lee, H. Kim, D.-k. Hong, and H. Ju. Correlation analysis of mqtt loss and delay according to qos level. In *Information Networking (ICOIN), 2013 International Conference on*, pages 714–717. IEEE, 2013.

[14] A. Najafi Nassab. Mobile devices in the distributed iot platform calvin. 2017.

[15] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 4(1):1–20, 2017.

[16] T. Nilsson. Authorization aspects of the distributed dataflow-oriented iot framework calvin. 2016.

[17] S. Park. Ocf: A new open iot consortium. In *Advanced Information Networking and Applications Workshops (WAINA), 2017 31st International Conference on*, pages 356–359. IEEE, 2017.

[18] P. Persson and O. Angelsmark. Calvin–merging cloud and iot. *Procedia Computer Science*, 52:210–217, 2015.

[19] P. Persson and O. Angelsmark. Kappa: serverless iot deployment. In *Proceedings of the 2nd International Workshop on Serverless Computing*, pages 16–21. ACM, 2017.

[20] M. Poongothai, P. M. Subramanian, and A. Rajeswari. Design and implementation of iot based smart laboratory. In *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 169–173. IEEE, 2018.

[21] E. Research. calvin-base. `https://www.github.com/EricssonResearch/calvin-base`, 2018.

[22] S.-H. Yang. Internet of things. In *Wireless Sensor Networks*, pages 247–261. Springer, 2014.

[23] M. Yun and B. Yuxin. Research on the architecture and key technology of internet of things (iot) applied on smart grid. In *Advances in Energy Engineering (ICAEE), 2010 International Conference on*, pages 69–72. IEEE, 2010.

[24] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.