Virtual Objects for Robots and Sensor Nodes in Distributed Applications over the Cloud Continuum

Adriana Arteaga, Nikos Filinis, Lei Fu, Carol Habib, Leonardo Militano, Dimitris Spatharakis, Anastasios Zafeiropoulos, Thomas M Bohnert, Nathalie Mitton, Symeon Papavassiliou

Virtual Objects for Robots and Sensor Nodes in Distributed Applications over the Cloud Continuum

Adriana Arteaga*, Nikos Filinis†, Lei Fu‡, Carol Habib*, Leonardo Militano‡, Dimitris Spatharakis†,
Anastasios Zafeiropoulos†, Thomas M. Bohnert‡, Nathalie Mitton*, Symeon Papavassiliou†
*Centre Inria de l'Université de Lille, France
†National Technical University of Athens, Greece
‡Zurich University of Applied Sciences, Switzerland
Corresponding authors: milt@zhaw.ch, carol.habib@inria.fr, dspatharakis@netmode.ntua.gr

***Abstract*—The cloud-to-edge-to-IoT continuum represents a seamless flow of data processing and management, spanning from centralized cloud services to distributed edge computing and interconnected IoT devices. This paradigm can become very challenging in real implementations, especially in the presence of multiple stakeholders using proprietary and heterogeneous software and hardware. The Horizon Europe NEPHELE project proposes the virtualization of IoT devices through a specific software stack called Virtual Object Stack (VOStack) that promotes openness and interoperability. In this paper, we present an implementation of VOStack using W3C Web of Things (WoT) standard in a post-disaster domain for two different types of IoT devices: a ground robot (Turtlebot2) for navigation and mapping in an unknown environment, and a Raspberry Pi 3 acting as a wireless sensor network gateway. We propose an application graph for the resulting hyper-distributed application (HDA) and present our first implementation to validate the proposed solution.**

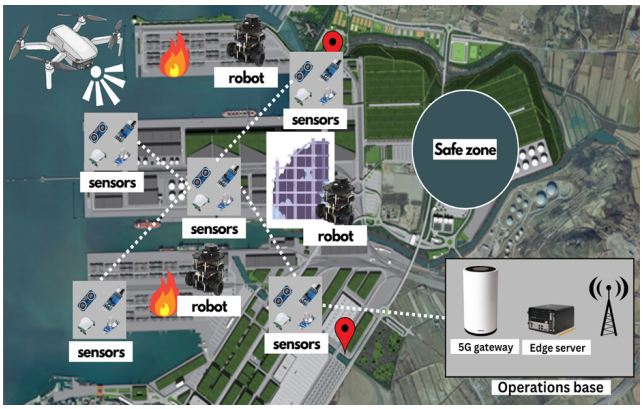


Fig. 1: Reference post-disaster scenario

I. Introduction

Modern software applications are distributed in nature, taking advantage of computing capabilities over the continuum ranging from the cloud to the edge of the network. The underlying network architecture follows a microservices approach whereby small, simple, single-functionality services are composed into coherent Internet-scale applications. The advantages of this approach are numerous, such as improved scalability, fault isolation, team productivity, deployment time and cost-efficiency. Nonetheless, in very heterogeneous and complex scenarios with multiple stakeholders and service domain actors, the implementation of an overall distributed application can become very challenging.

As a reference domain, we envisage a post-disaster scenario where multiple stakeholders are present and in urgent need of getting information about the environment, the presence of people, network statistics, maps of the area, sensing information [2]. Typically, multiple first responder teams such as firefighter brigades, police officers, first-aid teams, municipality officers, and business units approach the area with several high-tech devices. Each of these teams will have proprietary hardware (e.g., drones, ground robots, computing facilities) and software for a wide spectrum of heterogeneous devices, see Fig. 1 as a reference. These devices offer not only resources for computation and sensing but also produce information that can enhance the overall *situational awareness*. The main challenge however is, that a "silos-based" approach in developing hardware and software solutions makes it very difficult to allow hardware/software and data sharing.

To tackle this issue, among others, the Horizon Europe NEPHELE project [1] proposes the adoption of an IoT (Internet of Things) and edge computing software stack for leveraging virtualization of IoT devices at the edge part of the infrastructure and supporting openness and interoperability aspects in a device-independent way. To reach this goal, the Virtual Object (VO) concept is introduced as the virtual counterpart of an IoT device. The VO provides a set of abstractions for managing any type of IoT device through a virtualized instance while augmenting the supported functionalities through the hosting of a multi-layer software stack, called Virtual Object Stack (VOStack).

In this paper, we show how this innovation is used in a post-disaster domain where robotic hardware is used to map and navigate an unknown area, whereas a wireless sensor network (WSN) is used for collecting vital sensing information. The novelty of the implemented solution is based on the VO implementation for the involved IoT devices which are heterogeneous both in the hardware and the software and in the communication protocols they use. Based on the VOStack, these heterogeneous devices can now cooperatively *enhance*

[1]NEPHELE project website: https://nephele-project.eu/

the situational awareness of first responders. Data is collaboratively exposed and end-users can interact through user-friendly web-based solutions with technologically advanced hardware and software components like a robot. To reach this goal, the VO allows a ROS-based (Robotic Operating System) application to interact with the non-ROS world by exposing ROS topics and functionalities through the W3C Web of Things (WoT) standard. *To the best of our knowledge is this the first implementation of this kind*, demonstrating how technologies that are specifically developed for web users can also be used to interact and expose data from a robotic-specific application domain software. From the WSN perspective, developing a VO for its gateway, which is a sensor node acting as a sink and possessing energy, processing and memory resources, reduces the complexity that the WSN presents in terms of dynamics and constrained resources. In fact, in a post-disaster scenario, the WSN is subject to dynamic changes due to loss of connectivity, interference, and even real-time deployment [2]. Instead of creating a VO for every node in the WSN, we only create one for the consistent and unconstrained device, being its gateway. Therefore, energy-efficient and resource-aware self-organizing techniques and routing algorithms running in the WSN are isolated and handled locally within the network. The VO of the gateway then aims at aggregating the collected data in the WSN and calculating important networking metrics such as throughput, latency, and energy consumption.

The remainder of this paper is organized as follows. In Section II the main technological domains and standards for the paper are introduced. In Section III, the reference implementation scenario is introduced in details, whereas Section IV introduces the implementation aspects. In Section V the preliminary results are reported, before concluding with some future work analysis in Section VI.

II. Related Work

A. Cloud-robotics and ROS-based applications

The potential of the "Cloud Robotics" paradigm [1] is huge as it is deemed to solve typical issues with robots often requiring high investment costs (Capex) and having a reduced operational time. The adoption of cloud and edge resources may introduce improvements as cloud services can be shared across robots and the robotic hardware can be cheaper since part of the resources come from the Cloud. However, distributed applications have not yet reached its peak of acceptance and more research is required in this domain.

ROS is currently the most widely adopted middleware for the design and development of robotic systems. A ROS application is typically composed of distributed processes, ranging from sensor and camera drivers to algorithm implementations, or external interfacing and control. The processes (known as ROS *nodes*) communicate using either *topics*, an asynchronous pub/sub message system, *actions*, an asynchronous interface with a request-feedback-response architecture, or *services* which follow synchronous request-response patterns.

From the architectural point of view, ROS still treats the robot as a central point of the system and relies on local computation. This limitation makes the task of creating large-scale and advanced robotics applications much harder to achieve. By creating a cloud-to-edge-to-IoT continuum which provides the needs of such robotic environments, we can lower the hurdle for an application developer to use or extend robots' capabilities.

B. Wireless Sensor Networks and Virtualization

WSNs have been widely integrated to enhance context awareness throughout the IoT in different scenarios such as industry, healthcare, environment monitoring, and disaster response. A typical WSN is composed by a group of sensor nodes and a gateway. Usually, the sensor nodes are small, low-cost, and energy-constrained devices with sensing, processing and wireless communication capabilities. The gateway is a more robust device than the sensor node, so the gateway could act as the central point of the network for processing the data collected by the sensor nodes such as temperature, humidity, and pressure, or be a connection point to external components such as servers or the Internet. The WSN gateway virtualization allows functionalities running in the gateway to be accessed remotely, which brings flexibility and eases the management of sensor nodes considering their constraints. More importantly, it allows interoperability and ease of integration with other IoT devices, such as robots in our case.

Constraints such as the battery, memory, and storage of the sensors represent a challenge for the development of energy-efficient applications and, at the same time, implement a flexible and proactive solution for a dynamic context. A self-organized network refers to an adaptive communication infrastructure where devices autonomously organize and optimize their connectivity, adapting to changes in the environment or network conditions [5]. Self-organized networks in IoT leverage principles such as distributed decision-making, intelligent routing algorithms, and collaborative protocols to achieve efficient and adaptive communication among a diverse set of interconnected devices [4]. With cloud-to-edge-to-IoT continuum and virtualising specific unconstrained devices in the WSN (such as the gateway), it is possible to isolate the dynamic real-time management operations happening inside the WSN from the cloud, yet allow application developers to use or extend the capabilities of the WSN.

C. W3C WoT

The integration of IoT and the World Wide Web (WWW), called Web of Things (WoT) [6], proposes to integrate the existing Web ecosystem with Things (IoT devices) to provide an interoperable infrastructure that goes beyond basic network connectivity. The Thing is an abstraction that describes the interactions and characteristics of physical and virtual entities (e.g., IoT devices). Specifically, the Thing is defined by the Thing Description (TD) that includes *a)* a description of the available interaction affordances of the Thing i.e., properties, actions, and events, *b)* the protocol bindings to enable a protocol agnostic way of communication with the Thing, *c)* security mechanisms of the Thing, and *d)* semantic information

of the Thing. The main software building block of the WoT architecture is the Servient, which represents the software stack responsible for implementing the core WoT elements. The Servient is capable of hosting and exposing Things. As a result, the Servient has two different functionalities, namely; *i)* the Exposed Thing (ET) that represents a locally hosted Thing that can be accessed over the network by remote Consumers, and *ii)* a Consumed Thing (CT) that represents a remote Thing used by a local application.

D. VO for NEPHELE

As also mentioned in the introductory section, the NEPHELE project proposes the concept of VO following and implementing the WoT Things specification. As a vitalized entity, the VO is deployed on Edge/Cloud data centres and, therefore, can be orchestrated. The main challenges that the VO strives to address are *i)* interoperability aspects by implementing different communication protocols (i.e., HTTP, MQTT, CoAP); *ii)* secure communication aspects by integrating various security schemes (i.e., TLS, basic authentication, bearer authentication); *iii)* augmenting the capabilities of resource-constrained devices by exploiting the resource of Edge/Cloud layers to develop a set of resource-intensive IoT functionalities (e.g., data compression, data analytics functions, data aggregation), and *iv)* device management aspects by triggering actions or events that orchestrate the IoT devices by the application layer. The VO is defined and customized by *a)* the VO descriptor (i.e., the configuration for the VO), *b)* the TD (i.e., the descriptor that defines the Thing exposed through the VO), and *c)* the custom programming scripts for the user-defined functions.

III. Implemented scenario

This section describes the physical scenario to be vitalized using the VO and the VOStack to improve context awareness in post-disaster scenarios. The physical components are *a)* a ground robot Turtlebot2 running ROS version 2 that executes the mapping, and *b)* the WSN gateway executed on a Raspberry Pi 3. The robot is commanded through a dashboard to start the mapping task, control the movement of the robot, generate the map file, and display the battery level. On the other hand, the WSN gateway periodically sends temperature and humidity measurements to a web interface that displays the received values in real-time.

A. Application Graph

The resulting hyper-distributed application (HDA) can be represented using an application graph, where the nodes in the graph represent the application components, whereas the network interconnections link the components together to offer the full application functionalities.

Declarative statements can accompany the application graph in the form of tags or simplified expressions. These statements address nodes and/or links or the whole application and aim to specify the objective of the applications and its constraints regarding its deployment and operation to deliver the required results [3]. The VOs constitute part of the application graph and can be accompanied by a set of requirements, properties, and restrictions. In Fig. 2 the overall application graph for the implemented scenario is reported. We see the following elements:

Robotic hardware: for the robotic application component a physical robot is used for experimentation, namely a Turtlebot2 running ROS version 2 (Humble). Some application components are run as a Docker container directly on this device. Specific application components can be started on the physical robot on demand through so-called ROS launchfiles. These functionalities are exposed as actions through the ET. Similarly, a subset of the ROS topics and data available on the robot is exposed either as attributes or through a periodic function in the ET.

WSN: For the WSN application component, we suppose that a mesh WSN is sending monitoring data to its gateway using multi-communication technologies. The gateway is an unconstrained node. It exposes properties, actions and events to be consumed by its VO. Its NorthBound interface is configured to use the CoAP protocol.

Virtual Object 1 (VO1): Virtual representation of the physical ground robot. As its CT, it lists the physical robot which it exposes and possibly extends as its own ET to other application components and/or VOs. The communication from and to the VO exploits the Southbound and Northbound Interfaces offered by the VOStack. In this specific case the MQTT communication protocol is used between the physical robot and its corresponding VO (Southbound Interface of the VOStack), whereas the HTTP protocol has been selected for the Northbound Interface.

Virtual Object 2 (VO2): Virtual representation of the WSN gateway. It consumes the properties, actions and events exposed by the physical WSN gateway via CT and exposes them via ET to other consumers. It initiates and establishes communication with its VO using CoAP protocol. As a NorthBound interface, it uses HTTP protocol to expose its properties, actions and events to the web application (component 4).

Foxglove Dashboard: This container (component 1) is a ROS-based dashboard to visualize ROS-based topics and data and control the interaction with the physical robot. This container has a co-location constraint with another container (component 2). This container offers the bridge functionality towards the Foxglove dashboard and ROS-DDS bridge for communication over the Zenoh communication protocol.

Zenoh Router: this container (component 3) is the Zenoh router[2], for data sharing and communication in distributed systems supporting, among others, ROS2 middleware to enhance peer-to-peer connectivity through discovery-efficient pub-sub communications in ROS applications over the continuum.

GUI: a container (component 4) hosting a simple graphical user interface for interacting with the VOs in the application graph and displaying the retrieved information to the user. This is a Flask-based implementation linked to a web-based user

[2]https://zenoh.io/blog/2021-04-28-ros2-integration/

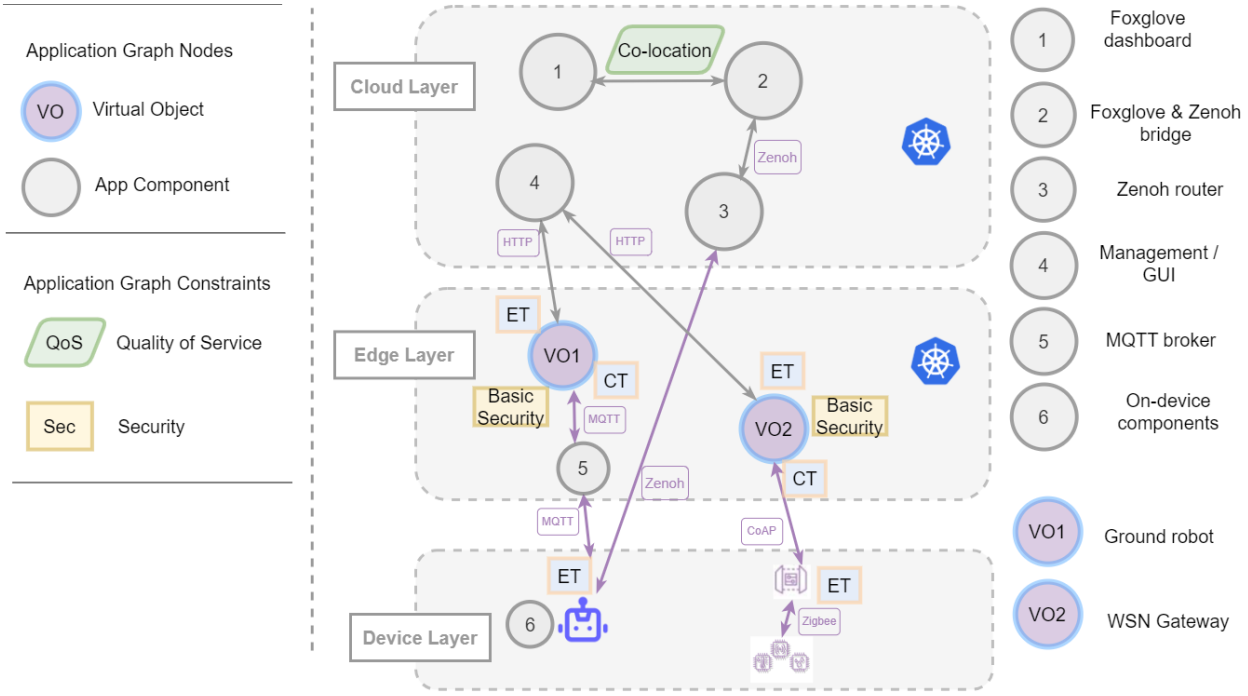


Fig. 2: Application graph for the implemented scenario

interface.

MQTT broker: a container (component 5) hosting an MQTT broker for communications between the physical robot and its VO.

IV. W3C WoT Implementation

A. WoT implemenation for Turtlebot2

The robotic hardware has the computing capabilities to run a WoT runtime environment to communicate with the VO. Specific ROS-based scripts have been developed so that the given functionalities exposed through the TD are supported and correctly implemented on the hardware.

Thing Description files: An extract of the designed TD for the Turtlebot2 robotic hardware is reported in Listing 1. The main building elements are the **properties** like battery charge level (%), battery charging status (boolean), available ROS launchfiles to be triggered (array of strings); **actions** being ROS launchfile to be triggered (string), built map export (string), current values (battery status and level) through a periodic function; **events** triggered in case of low battery level in our implementation.

VO descriptor files: define the specific Northbound (HTTP) and Southbound (i.e. MQTT) implementations to be used, the security aspects for the VO, the data storage aspects, information about the physical device and the functionalities being exposed or extended. In our specific case the VO acts as a proxy of the functionalities exposed by the physical devices. However, additional functionalities could be offered (e.g., data manipulation, statistics) for which the required supportive/generic function should be provided as an add-on to the VOStack.

Listing 1: Extract of robot WoT Thing Description

```
"properties":
  "allAvailableResources":
    "type": "object",
    "description": "Current resource level",
    "properties":
      "battery_percent":
        "type": "integer",
        "minimum": 0,
        "maximum": 100
      "battery_charging":
        "type": "boolean"
      "possibleLaunchfiles":
        "type": "array",
        "description": "List of launchfiles",
        "items":
          "type": "string"
"actions":
  "triggerBringup":
    "title": "Trigger ROS launchfiles",
    "input":
      "type": "object",
      "properties":
        "launchfileId":
          "type": "string",
    "output":
      "type": "object",
      "properties":
        "result":
          "type": "boolean"
      "message":
        "type": "string"
  "mapExport":
    "title": "Export map data",
    "output":
        "type": "object",
        "properties":
```

```
            "maptoString":
                "type": "string"
    "currentValues": ...
"events":
    "outOfResource":
        "data":
            "type": "string"
```

B. WoT implementation for WSN gateway

The WSN gateway has some computing capabilities, therefore it can run a WoT runtime to communicate with the VO. Given that, a valid TD, VO descriptor and python script should be provided on the VO side as well as on the WSN gateway side. In the following, we describe the main functionalities that we have configured or developed to create the custom version VO2:

Listing 2: Extract of WSN gateway WoT Thing Description

```
"properties":
    "temperature":
        "type": "double"
    "humidity":
        "type": "double"
"actions":
    "currentValues":
        "description": ...
```

Thing Description files: From the WSN gateway side, two *properties*: temperature and humidity and one *action* consisting on reading current properties are defined as shown in Listing 2. From the VO side, three *properties*: temperature, humidity and average values, as well as two *actions* consisting on reading current values and calculating the average values, are defined as shown in Listing 3. Their corresponding handler functions are included in the Python scripts at both sides.

VO descriptor files: Aside from specifying the communication protocols to be used by the NorthBound and SounthBound interfaces, some important parameters that are set up in the VO descriptor at the VO side, are generic available functions such as $mean\_value$, vo_status and $device\_status$ and periodic user-defined function allowing VO2 to periodically fetch data (properties values) from the WSN gateway. An asynchronous function calling the generic available functions and an asynchronous function for the user-defined function are included in the Python script of VO2.

Listing 3: Extract of VO2 WoT Thing Description

```
"properties":
    "temperature":
        "type": "double"
    "humidity":
        "type": "double"
    "average_values":
        "type": "object"
            "average_temperature":
                "type": "double"
            "average_humidity":
                "type": "double"
"actions":
    "currentValues":
        "description": ...
    "averageData":
        "description": ...
```

V. Preliminary Implementation section

We next report on our implementations of the proposed ideas discussed in this paper[3].

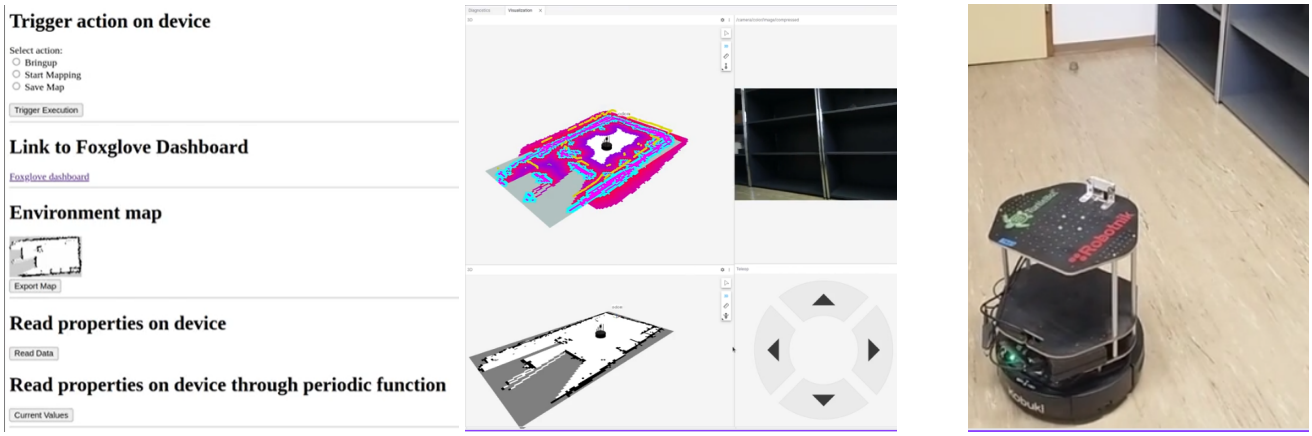
A. Testing the robot WoT implementation for a distributed application

We have deployed all the containers in our Kubernetes cluster on our Openstack installation. The first part of the demo showcases the advantages of using the VO stack in complex robotic software for navigation and mapping in an unknown environment. The GUI Web application, in Fig. 3a, is used to interact with the VO and a Foxglove dashboard, in Fig. 3b, to display the robotic messages and data. Using the Web application, we can interact with the robot device, in Fig. 3c, through the VO abstraction to read the exposed properties and trigger the actions on the robot. An action is triggered to launch a robotic ROS process on the device that brings up all robotic controllers and the basic scripts. To perform the mapping, we need to trigger an action on the robot through the VO. This will start a ROS mapping process on the device after verifying there is a sufficient level of energy in the battery. By controlling the robot through the dashboard, we can build a map of an unknown environment. This map can be saved using another action through the VO. The saved map will be on the physical device but will also be exposed for access through the VO. This can be retrieved using an action so that the map can be displayed in the Web application for further use in the disaster scenario. For instance, non-robotic experts and other first responders' teams can access the map for an enhanced situational awareness.

B. Testing the WSN gateway WoT implementation

We have tested the WoT implementation of the WSN using a Raspberry Pi 3 (Raspberry Pi OS) acting as the WSN Gateway and an Ubuntu computer hosting its VO. Both devices are in the same LAN and have static IP addresses. To visualize on a GUI the generated temperature and humidity data by the Raspberry Pi 3, a simple web application was designed using Flask for the Back-End development and HTML, CSS and Javascript for the Front-End development. The web application runs on another computer in the same LAN. It consumes the exposed thing by the VO. It displays plots that dynamically and periodically update with new readings. To execute and test the complete implementation, first, the program on the Raspberry pi is run using the WoT runtime. It then starts listening on the CoAP port and generating dummy temperature and humidity data as shown in Fig. 4a. Its thing description is ready to be consumed. Second, the program on the computer hosting VO2 is run using also the WoT runtime. As defined in its VO descriptor file, VO2 consumes the exposed thing on the WSN gateway. It also runs a user-defined periodic function to

[3]A demonstration video reporting our implementation and validation results can be found at the following link: https://acesse.one/bVrrh



(a) **(b)** **(c)**

Fig. 3: a) GUI for the web application, b) Foxglove dashboard to control robot, c) Physical Turtlebot2.

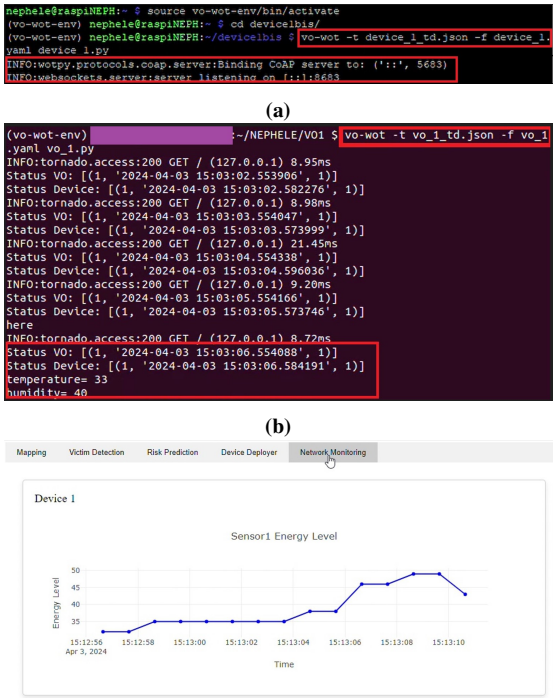


Fig. 4: a) Execution on Raspberry Pi, b) Execution of VO2, c) Web Application.

read data as well as generic functions to print its status and the status of the device it is consuming. Using CoAP protocol, it then starts periodically reading the dummy data generated by the WSN gateway as shown in Fig. 4b. Finally, the generated data can be visualized on the web application as shown in Fig. 4c. By that, the end-to-end communication from the physical device to the web application passing by the VO is tested and validated.

VI. Conclusions and Future Work

In this paper, we have shown an implementation of the VOStack in a post-disaster domain with heterogeneous devices. The implemented solution demonstrates the VO viability as a solution to integrate heterogeneous communication protocols, standards, software and hardware to an end-user using web-based solutions like W3C WoT. In future work, we intend to customize the WoT implementation of the WSN gateway to include more properties, actions and events specific to the nodes of the WSN that are deployed in the exploration region. Furthermore, we intend to further integrate the robotic side of the application with the WSN to ensure a better understanding of the exploration mission and enhance the overall situational awareness. This integration can occur not only at the level of the web application but also at the level of the VOs.

Acknowledgment

This work was funded by the European Union's Horizon Europe research and innovation program under grant agreement No. 101070487 (NEPHELE).

References

[1] Toffetti, G.; Bohnert, T. "Cloud robotics with ROS." Robot Operating System (ROS) The Complete Reference (Volume 4). Cham: Springer International Publishing, 2019. 119-146.

[2] Militano, L.; Arteaga, A.; Toffetti, G.; Mitton, N. The Cloud-to-Edge-to-IoT Continuum as an Enabler for Search and Rescue Operations. Future Internet 2023, 15, 55.

[3] Zafeiropoulos, A., et al. "Intent-Driven Distributed Applications Management Over Compute and Network Resources in the Computing Continuum." 2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT). IEEE, 2023.

[4] Nurlan, Z.; Zhukabayeva, T.; Othman, M. "IoT Hardware-Defined Routing Protocol for Dynamic Self-organizing Wireless Mesh Networks." 2020 IEEE 10th International Conference On Consumer Electronics (ICCE-Berlin). pp. 1-4, 2020.

[5] Foubert, B.; Mitton, N. "Lightweight network interface selection for reliable communications in multi-technologies wireless sensor networks." In 2021 17th International Conference on the Design of Reliable Communication Networks (DRCN), pp. 1-6, IEEE, 2021.

[6] W3C Web of Things, https://www.w3.org/WoT/, Last Accessed 23-04-2034.