Open Access

Open Journal of Internet of Things (OJIOT)
Volume 4, Issue 1, 2018
http://www.ronpub.com/ojiot
ISSN 2364-7108

Middleware Support for Generic Actuation in the Internet of Mobile Things

Sheriton Valim, Matheus Zeitune, Bruno Olivieri, Markus Endler

Departamento de Informática, Pontifical Catholic University, Rua Marques de São Vicente 225, Rio de Janeiro, Brazil, {svalim, bolivieri, endler}@inf.puc-rio.br, matheuszeitune@aluno.puc-rio.br

Abstract

As the Internet of Things is expanding towards applications in almost any sector of our economy and daily life, so is the demand of employing and integrating devices with actuation capabilities, such as smart bulbs, HVAC, smart locks, industrial machines, robots or drones. Many middleware platforms have been developed in order to support the development of distributed IoT applications and facilitate the sensors-to-cloud communication and edge processing capabilities, but surprisingly very little has been done to provide middleware-level, support and generic mechanisms for discovering the devices and their interfaces, and executing the actuation commands, i.e. transferring them to the device. In this paper, we present a generic support for actuation as an extension of ContextNet, our mobile-cloud middleware for IoMT. We describe the design of the distributed actuation support and present a proof of working implementation that enables remote control of a Sphero mobile BB-8 toy.

Type of Paper and Keywords

Regular research paper: *Internet of Things, Actuation, Middleware, Bluetooth, Actuator Representation*

1 Introduction

Actuator devices, or Actionable Smart Things (AST), are an essential element of several IoT applications, such as smart homes, smart transportation, healthcare, retail or smart industry, because they can change something in their physical environment. Examples of AST are HVAC, smart lights, smart locks - for doors and equipment -, robots, drones and other mobile or static equipment. In all these cases, the actionable smart thing is usually connected wirelessly to some gateway, which connects the AST to backstage services running in a cloud or cluster. And many of such AST use Bluetooth Low Energy (BLE) for wireless communication, because of BLE's low power consumption, fast discovery and pairing with devices, and its reliable communication link. As the downside is the restricted wireless coverage, which is approximately 50 meters. On the other hand, almost any modern smartphone also has a Bluetooth interface, making it eligible to play the role of an Internet gateway for the smart things, as long as it is within its BLE range.

This motivated us to tackle the Internet of Mobile Things (IoMT), where any smart thing – and the gateways – are mobile, i.e. can be potentially moved or move autonomously, in case of autonomous robots. And to implement this support we developed the Mobile Hub (M-Hub), an Android middleware service that runs in background and is independent of other mobile apps[15]. Although in IoMT the system has to cope with eventual wireless disconnection of the AST, which may happen

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2018)* in conjunction with the VLDB 2018 Conference in Rio de Janeiro, Brazil. The proceedings of VLIoT@VLDB 2018 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

when the smartphone or the smart thing moves away from de mutual BLE coverage, this is quite seldom when the smartphone user is the one that is driving/controlling the actuation on the AST. Otherwise, if the actuation commands are produced remotely, and the smartphone is a mere intermediary, then a reliable actuation has to rely on the presence of several Mobile-Hubs close to the AST, and a smart and seamless handover process of the AST among the Mobile-Hubs.

In most of the current IoT systems, actuation-specific logic is usually hard-coded and intertwined with the remaining application logic. This is so because, so far, IoT applications are tailored to very specific - and proprietary - smart things with particular/proprietary actuation protocols and low-level actuation instructions and feedback signals. Furthermore, most of IoT systems assume a stable wireless connection between the wireless gateway and the smart things, which is not the case in the IoMT.

However, when applications are required to interact with - and act upon - many and different sorts or products of ASTs over time, a hardcoded solution has to be replaced by a more flexible mechanism implemented in the middleware layer. For example, consider an IoT application that is able to dim (or change color of) all smart light bulbs inside several conference/meeting rooms of an institution. However, since these rooms belong to different departments, the kinds and makes of the smart bulbs are different and therefore use different GATT/BLE commands for the controls, that are defined by the vendor. Hence, if a user would need to control the lights in previously unvisited room, he/she would first have to inspect the bulbs, download the mobile app and then change the light intensity or color. Moreover, in the same room, smart light bulbs of different vendors may be used simultaneously, perhaps because some bulbs are more economic but the previous bulbs "still work well". On the other hand, it would be good if the user had access to high level commands such as *setdim(50)*, *setRGB(20,10,50)*, or *turn(off)*, that would be uniformly applied to any smart light bulb, in any room.

Another hypothetical example is that of futurist vacuuming robots, deployed for cleaning the floor of a house, e.g. of elderly people. Many of these vacuuming robots already now learn the floor plan of the house to be cleaned so that most of the time they manage to roam around and clean all the rooms well. But once and a while it may happen that the robot faces some obstacle hindering it to proceed with the planed itinerary. In this case, the robot may get information about the type and ID of the probable obstacle (maybe through cameras or other sensors). It may then issue simple high-level actuation commands like *door.open(X)*, *wheelchair.rotate(W,30)* or *serumSuppport.move(Z,100)* for opening a door X, rotating by 30 degrees a wheelchair W or moving forward 1 meter a motorized serum support Z. But such generic actuation commands are only possible if the IoT software infrastructure can discover and identify the smart things in the vicinity of the robot, automatically download driver software for actuation and convert the generic commands into the vendor-specific bytecode sequences to be sent to the AST.

On-demand drivers enable large-scale actuation scenario, because in addition to being able to control multiple devices simultaneously, in the broad scope (e.g. of a Smart City) drivers can be switched on-demand by the gateways, allowing them to be reused in the intermediary function of the actuation. Furthermore, scripts can be used to remotely control multiple mobile objects at the same time, since the actuation will be performed by the millions of gateways scattered.

Unlike other works that assume that the IoT smart things have sufficient memory and processing resources so as to allow the provision of an API with high-level and complex operations for actuation control, or even the execution of a virtual machine in the AST, we take a more challenging approach, where we want to primarily support very simple and resource-constrained smart things, where a few basic commands and parameters in raw bytecode format are used to control the device.

This motivated us to extend our IoT middleware ContextNet [8, 6] with a service and protocol that support such generic actuation. These services run partly in the cloud/cluster and partly on a Mobile-hub, that directly interacts with the AST. The currently supported WPAN is Bluetooth Low Energy (BLE). The goal was to allow that actuations composed of sequences of basic instructions be formulated in terms of high-level control commands (CC), that are then translated into corresponding low-level strings of bits-and-bytes that can be processed by the specific smart thing's actuation processor. In order to enable this, two new components, the Mobile-Actuator (M-Act) which is a microservice for the Mobile-Hub (for Android devices) and the Smart Objects Manager (SOM), a microservice of the SDDL core executing in cloud/cluster were developed. These two services exchange MACTQuery messages in JSON format in order to instrument and drive the Mobile-Hubs to interact and control some AST in its vicinity.

This paper is structured as follows: We proceed with an overview of the ContextNet middleware (Section 2) and of BLE technology that was used for interfacing with the Actionable Smart Things (Section 3). In Section 4 we first give an overview and then present in more details the M-ACT and SOM microservices, which are the building blocks of ContextNet's generic actuation support (in Subsections 4.1 and 4.2, respectively).

Section 5 introduces our proof of concept AST, the Sphero's toy BB-8, and the peculiarities of its actuation protocol and commands. We then discuss the rare related works (in Section 6) and close with concluding remarks (in Section 7) on actuation in IoT and over wireless connections.

2 The ContextNet Middleware

In order to support IoMT applications, we developed the ContextNet middleware, that has a three tier software architecture that consists of: (a) a SDDL core, a P2P of nodes executing in a cloud or cluster of stationary machines and interacting with (b) Mobile-Hubs executing on Android devices or boards (Android phones, Raspberry PIs, etc.), and that use their WPAN interface to discover, select, connect to and exchange sensor data or actuation commands with sensors and actuators, respectively, of smart objects/things (AST). Currently, our Mobile-Hub is capable of connecting only to Classic Bluetooth and Bluetooth Smart (BLE) enabled smart objects.

The SDDL core [6] handle the communication architecture, by enabling a mobile message-oriented middleware platform. These platform present lightweight mobile protocol, the MR-UDP [6], that handle intermittent connections, including soft handover. This protocol can also maintain data-streams to the Mobile-Hubs through the usage of low-rate keep-alive (heartbeat) messages.

The Mobile-Hub [16] is the smartphone middleware level that supports M-Act as a microservice. The Mobile-Hub provides multiple Android native access to M-Act as the lower-level connection to actuators through Bluetooth and Bluetooth LE and further communication channels.

2.1 The Mobile Hub

The Mobile Hub (M-Hub) "bridges the gap" between the Internet connection with the SDDL Core, and the short-range wireless connections with smart things. It is composed of the following independent micro-services that interact through Publish/Subscribe using the EventBus. The **LocationService** is responsible for sampling the M-Hub's current position and attaching it to whatever message is sent to the SDDL Gateway (GW), which can be either a static, manually entered geo-point, or the latest geo-coordinate obtained from the smart phone's embedded GPS sensor. The **Short-Range Sensor, Presence and Actuation (S2PA)** Service implements the TechnologyListener and interacts with all nearby smart things with either a Bluetooth Classic or BLE interface (more details in section 2.2). The **ConnectionService**, runs the ClientLib/MR-UDP for communication with the SDDL Core and, in order to optimize transmission over the wide-area Internet link, this service may group several messages (commands or acks) for optimized, bulk transmission.

In order to support local processing of SDDL Core-inbound or out-bound data M-Hub also features the *Mobile EPA (M-EPA)*, which is a full-fledged Complex Event Processing (CEP) engine that is able to analyze data streams (e.g. commands or sensor data), perform filtering, aggregation, summarization, and window-based detection of event patterns, that are described and implemented as independent ECA rules in a specific EPL language. The periodicity and duration of many of these services' actions, is influenced by the device's current energy level (that is classified as LOW, MEDIUM, HIGH). This level is set by the last service, the **Energy Manager**, which periodically will sample the device's battery level and check if the smartphone is connected to a power source.

For the purpose of managing the actuation with smart things we thus introduced a new M-Hub microservice: M-Act. Since actuation involves interactions through the WPAN it is necessary to take a closer look at the S2PA, which provides some basic BLE interactions functions for M-Act.

2.2 S2PA

The Short-Range Sensor, Presence and Actuation API (S2PA) is a microservice of the M-Hub to handle the discovery and connection with nearby M-OBJs through different short-range wireless technologies (WPAN). It was designed to handle any WPAN, implementing an interface that can be mapped to the capabilities of the supported WPAN. In order to support any WPAN, S2PA defines some basic methods and interfaces that should be implemented in all these technologies: 1) Discovery of, and connection with M-OBJs, 2) Discovery of services provided by each M-OBJs, 3) Read and Write of service attributes and 4) Notifications about disconnection of M-OBJs. Currently, S2PA supports Bluetooth Low Energy (BLE) and Classic Bluetooth.

3 Bluetooth Low Energy (BLE)

BLE is becoming a very popular WPAN technology for IoT because it is extremely power efficient, features fast discovery of peripheral devices and supports up to 2000 simultaneous connections of peripherals with devices in master. The main reason for its wide dissemination - and hence, our reason for choosing it as the IoT WPAN - is the fact that BLE is now available on any modern smartphone. Moreover, BLE technology is also being embedded into more and more peripheral devices, such as gadgets, wearables, badges with beacons, Smart (sensor) tags, smart light bulbs, toys, household appliances, etc.

The communication between two connected BLE devices is based on the Generic Attributes Profile (GATT). It defines a hierarchical data structures used to connected devices, which allows the discovery of services provided by a BLE device. A Profile is composed of a collection of **services** that describe the device's use case. And a **service** represents a specific, independent functionality (or data source) of the device, and is a collection of **characteristics** and/or references to other **services**. A **characteristic** contains a value/parameter used by a service and may also have information about the value. For example, in the SensorTag[1] the **Accelerometer Data** is associated with the characteristic UUID **aa11** and the **Accelerometer Configuration** is associated with the characteristic UUID **aa12**.

In the BLE environment, in order to send an actuation command, each function of an AST is associated with one tuple [**service**, **characteristic**]. By varying the **service** and **characteristic** the command is assembled. For the specific example of the Sphero's BB-8 Toy, a **service 2ba0** that holds a **characteristic 2ba1** and the tuple [**2ba0**, **2ba1**] is responsible for all commands.

4 Overview of our Approach

The extension of ContextNet required for supporting the control of actionable smart things (or objects) is based on two microservices, named the Smart Objects Manager (SOM) and Mobile-Actuator (M-Act). The SOM runs in the cloud/cluster, i.e. a processing node of the SDDL Core, and holds a repository of drivers for actuators, as well as a `connectedTable`, associating each discovered AST device with an actuator to a single M-Hub, that is the current intermediate/surrogate for controlling this device. Notice that this surrogate M-Hub for a device may change over time, as a M-Hub may encounter and connect to many AST devices over time, and AST devices may, accordingly, be discovered and served by different M-Hubs. The M-Act is a microservice running in the M-Hub, i.e. on any Android based device. The main objective of M-Act is to connect to actionable BLE devices, discover the actuator of the device, download the driver from SOM, and then translate a generic actuation protocol into the native protocol of the actuator manufacturer.

The actuation process is illustrated in Figures 1 and 2. Periodically, the M-Hub scans for devices in its vicinity and immediately after the discovery of a new AST, it checks if there is an appropriate driver for the actuator in its local "drivers cache" (Figure 1 - Case 1). If it does not find a suitable driver in its cache, it requests this driver to the SOM (Figure 1 - Case 2). When the SOM has the driver for the discovered AST, it replies by sending to M-Act the requested driver, otherwise it notifies M-Act that no suitable driver was found. When the driver has arrived and properly deployed in M-Act, it will be able to connect to the AST and notify the SOM about the connection status. After this point, actuation commands can be received, translated into native commands, and sent to the AST.

In order to send actuation commands to an AST, a client[2] must first query SOM to learn which M-Hub is the current surrogate of the target AST, and receive a list of commands available in the AST's driver specification. Moreover, a client can query the SOM to receive a list of available ASTs at a specific location, sending the GPS coordinates of an area. After the client receives the SOM reply with the UUID of the M-Hub associated to the AST and the list of commands, it will be ready to control de AST by sending actuation commands to it. These generic actuation commands are delivered to the M-Hub that is the current surrogate of the target device, and will be processed by the M-Hub's local M-Act service. M-Act and will then translate the generic, high-level actuation commands into the specific bytecode-level messages of the native protocol of the AST, in accordance to the specification of the AST-specific driver. Then, M-Act will forward the native protocol message to the S2PA service, which deliver it the AST device using the corresponding technology. So far, we have only used it for BLE.

All the messages between the M-Act and SOM services have the MACTQuery format (Figure 3). The field type distinguishes a command (**cmd**) message from a **driver** message. Command messages may hold any generic actuation command aimed at an AST, while driver messages are used to haul a AST driver specification from SOM to M-Act, or else, used by M-Acts to request a required driver. When a client constructs an actuation command, it should fill the **target** field with the UUID of the AST device to be controlled.

4.1 M-Act

The M-Act is a service of M-Hub aimed at managing the wireless communication aimed at controlling an actuator of Actionable Smart Things. In a nutshell, M-Act performs two tasks: (i) to request from SOM a driver

[1] Texas Instruments CC2451 Sensor Tag - http://www.ti.com/lit/ug/swru271g/swru271g.pdf

[2] This may be any node, mobile or not, running the ContextNet middleware.

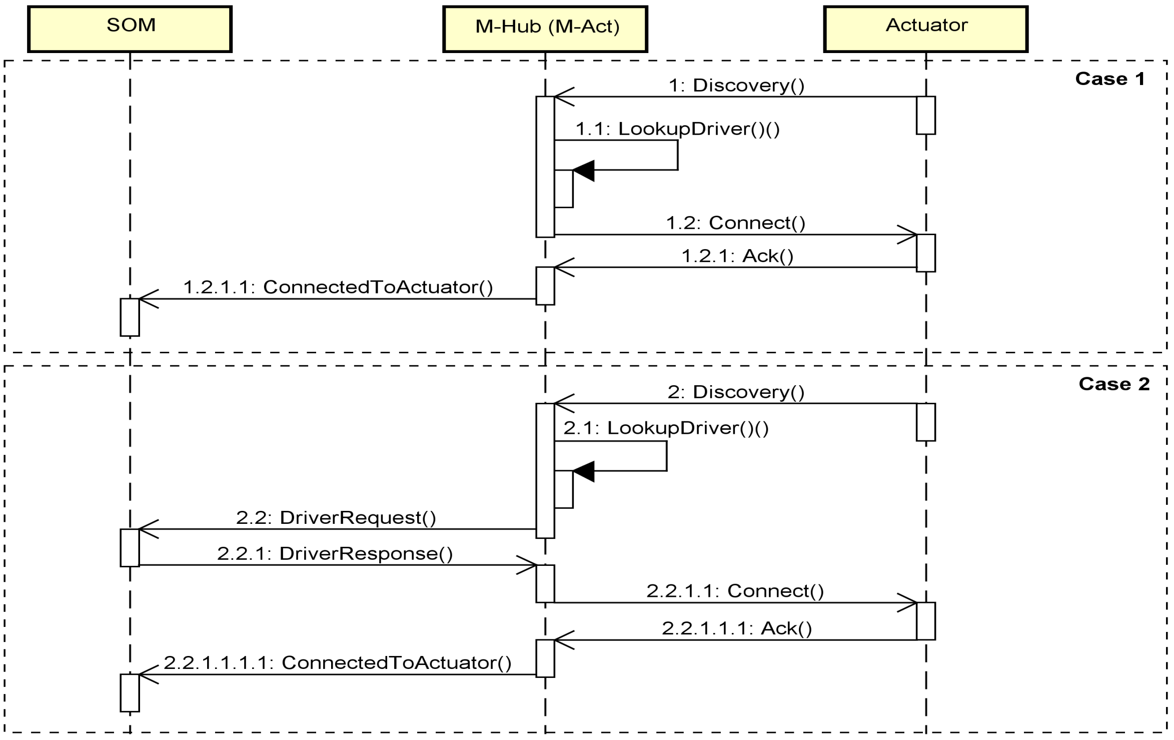

Figure 1: Connection steps

for this actuator; and (ii) to translate a generic actuation command (received from a remote ContextNet node) to the native protocol of the specific actuator.

As already mentioned, the translation from the generic command to the native protocol of the actuator is accomplished with the help of the driver. The driver is written in JSON format (Figure 4) and supports/implements all - or at least the most important - actuation commands recognized by the actuator, as well as the necessary steps to unlock the device, when needed. The use of drivers enables M-Act to control a wide range of different devices and gives the ability to extend, on demand, its actuation capability for new AST when they are first met. But M-Act will also not indefinitely keep the drivers of all AST devices that it had been connected to in the past. Instead, if some downloaded drivers are unused for some time, the M-Act will "garbage collect" and dispose them to open some space in memory for the download of other drivers needed to control newly encountered ASTs.

As a microservice of the M-Hub, M-Act interacts with other services running in the M-Hub. It does this through the eventBus (Figure 5), which is a Publish/Subscribe (asynchronous) communication interface (Figure 6). M-Act works receives MACTQuery messages from the ConnectionService, which handles inbound and outbound messages from the SDDL core (i.e. the cloud). A received MACTQuery message may hold and generic actuation command addressed to a specific AST device and M-Act translates it into the native protocol of the actuator manufacturer, then it sends the translated message to the actuator through the S2PA Service. As shown in Figure 7, when M-Act receives a MACTQuery from ConnectionService, internally the message is received by `RegControl`, which chooses the appropriated driver from the local register, then sends the command and the driver to the Processor unit which parses the driver and the command and translates the command to the native protocol of the AST. Finally, M-Act sends the resulting native bytecode to S2PA.

With this basic actuation support of the middleware, there are essentially two ways remote actuation can take place on a AST device:

1. Via a remote application operated by a human (e.g. remote control of mechanical arm controlled by someone using a smartphone or PC), or

2. Actuation on a AST device is autonomously/automatically started as soon as some Complex Event Processing rule run locally (in M-Hub's MEPA service) or remotely (in a SDDL core Processing Node) as soon as some

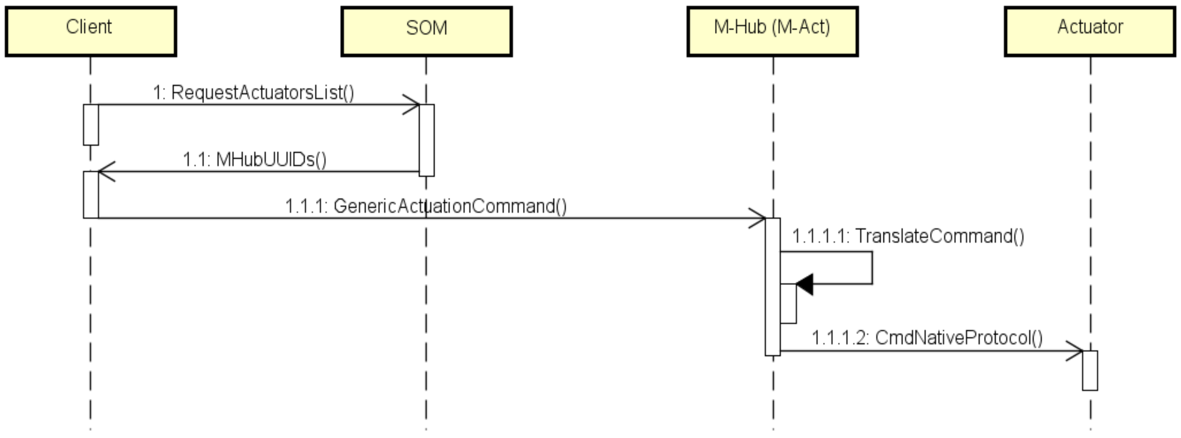
Client
SOM
M-Hub (M-Act)
Actuator
1: RequestActuatorsList()
1.1: MHubUUIDs()
1.1.1: GenericActuationCommand()
1.1.1.1: TranslateCommand()
1.1.1.2: CmdNativeProtocol()

Figure 2: Actuation sequence

sensor data pattern of interest is detected. E.g. after the local temperature reaches 30 degrees some HVAC in the surroundings are turned-on so as to drop the temperature)

4.2 Smart Objects Manager

The Smart Objects Manager (SOM) is a microservice of the ContextNet Core. It has two main tasks: (i) it holds a repository of drivers for sensors and actuators and (ii) it maintains the `connectedTable`. The former is responsible to provide drivers upon requests from M-Hub, control the drivers live updates and control distinct versions as well. Drivers are lightweight documents stored in and transmitted in JSON format.

The `connectedTable` is the directory that catalogs M-Hub connections to sensors and actuators. It prevents two M-Hubs from trying to connect with sensors and actuators at the same time, respecting the causal order of the requests. As illustrated in Figure 1, every time a M-Hub connects to an AST device, the M-Hub sends a found M-Obj message to SOM, to inform it about this connection and the SOM uses this message to update the `connectedTable` by inserting a tuple [UUID M-Hub, UUID M-Obj]. And when the M-Hub looses the connection with the AST, i.e. due to a wide relative movement among them or a WPAN connection problem, the M-Hub sends a disconnected M-Obj message to SOM to inform it about this disconnection, and the SOM updates the `connectedTable` by removing the tuple [UUID M-Hub, UUID M-Obj].

The AST driver contains several parameters that include the device ID, the driver version, the communication protocol used by the device (i.e. BLE, Classic Bluetooth), authentication credentials, when applicable, and the supported device commands. All these parameters are stored and exchanged in JSON format, being interpreted by M-Act. The device commands are the collection of cataloged possible commands for data/state query or actuations.

5 CONTROLLING THE TOY ROBOT BB-8

In order to demonstrate the feasibility of our approach and test it on a rather sophisticated AST, we acquired a small mobile toy robot from Sphero, with the Star Wars theme, the BB-8. The toy is controllable by a mobile app[3] and interacts with the robot through BLE.

After a detailed study of Sphero's official documentation, we learned BB-8's main characteristics and the main commands: for movement and controlling the light/color of its LED. We separated these characteristics in two parts: the payloads of the messages sent to the robot through BLE, and the available communication channels between the operator and the AST BB-8. The channels give access to the low-level, internal controls of the robot, such as registers for unlocking the BB-8.

5.1 Actuation Protocol

The communication with BB-8 employs some BLE services and characteristics that are represented through unique UUIDs. There are two essential BLE services: the one responsible for acceptance of actuation commands, and services responsible for communication with the system.

Before sending any actuation command, it is necessary to do a specific unlocking procedure. This unlocking has to be repeated for every new connection and consists of sending some commands to certain

[3] BB-8^{TM} App Enabled Droid, for iOS and Android.

```
{
   "MACTQuery" :
   {
      "type" : "cmd|driver",
      "label" : "command_label",
      "target" : "mobject id|mobject group id|...",
      "cmds" :
      [
         {
            "seq" : 0,
            "cmd" : "move|setColor|...",
            "args" : "cmd arguments accordingly to driver description“
         },
         {
            "seq" : 1,
            "cmd" : "move|setColor||...",
            "args" : "cmd arguments accordingly to driver description“
         }
      ]
   }
}
```

Figure 3: MACTQuery

```
{
    "device_type" : "bb8",
    "interface":"ble",
    "connection":
    {
        "service": <uuid_service>,
        "characteristic": <uuid_characteristic>,
        "cmd" : <cmd_array>,
    },

    "commands":
    {
        "roll":
        {
            "service": <uuid_service>,
            "characteristic" : <uuid_characteristic>,
            "cmd" : "0x30",
            "arg_size" : "5",
            "arg_type" : "byte",
            "offset" : <offset_expression>,
            "cmd_line" : "0xFF, 0xFE, <cmd>, <args>, <arg_size>, <offset>"
        },

        <list_other_commands_available>,
    },
}
```

Figure 4: Driver sample

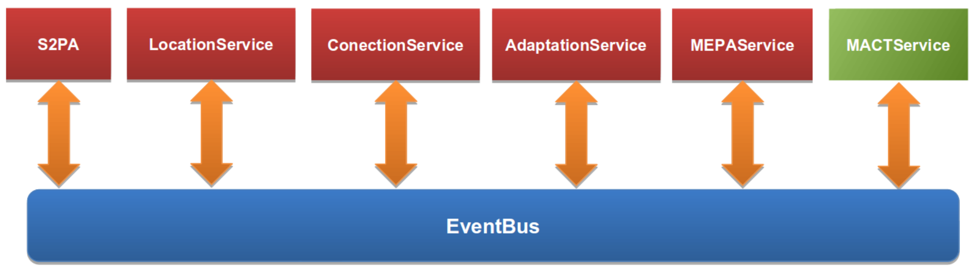

Figure 5: EventBus

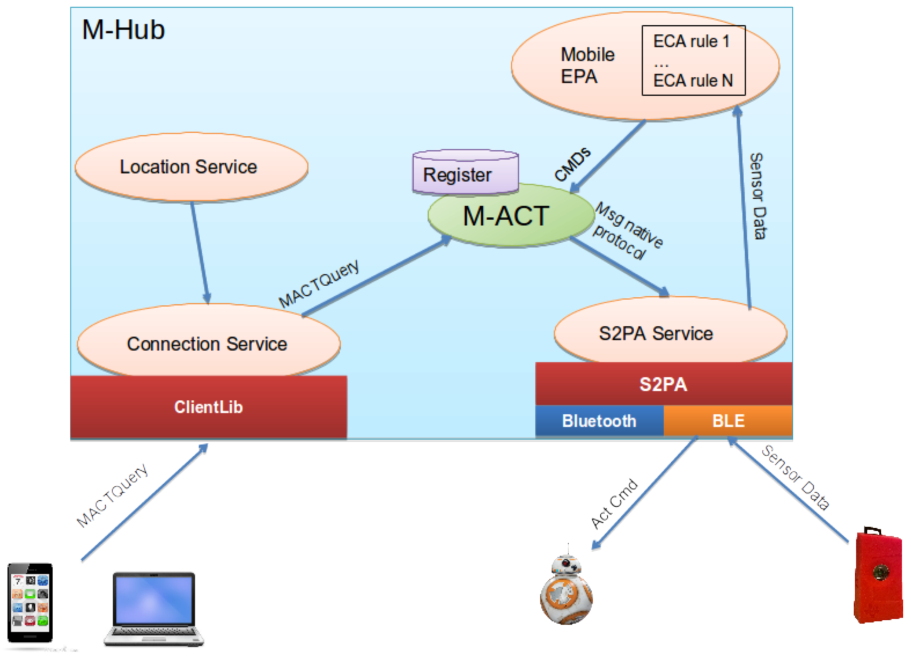

Figure 6: The M-Act microservice in the mobile hub

BLE characteristics. The commands free three different resources in BB-8, the *Anti-Denial-of-Service mechanism*, the *txpower control*, that is responsible for activating the electric motors and *wakecpu*, which "wakens up" BB-8 for our control commands.

The message format is a simple array of bytes, including a flag telling if communication will be synchronous or asynchronous, the command ID, the length of the bytearray, the parameters of the command and a checksum. In order to illustrate how the fields of the message are constructed let's take command `roll`, that commands BB-8 to move some distance in the "current" direction. To assemble the bytearray, we need the ID of Roll, the speed, the angle and a flag to tell if the robot is to start or to stop. Most of these parameters occupy one byte, except for the angle, that takes two bytes, so length specifies the total amount of bytes. If, on the other hand, we need to send command to change the LED color to red, the commandID would be SerRGB, and the length will be the total number of bytes.

5.2 Preliminary Tests

Having designed and implemented the driver, we proceeded to define the tests. Initially, we tested if the driver was actually playing correctly the commands, which it did. Then we aimed at evaluating the performance of the actuation mediated by the proxy. For this we measured the time from the instant that M-Hub receives the command from the Gateway of SDDL Core (which is then processed by M-ACT and sent via BLE to the robot) until M-Hub receives the response from the BB-8. We repeated this test 100 times, always establishing a new connection with the BB-8 and sending

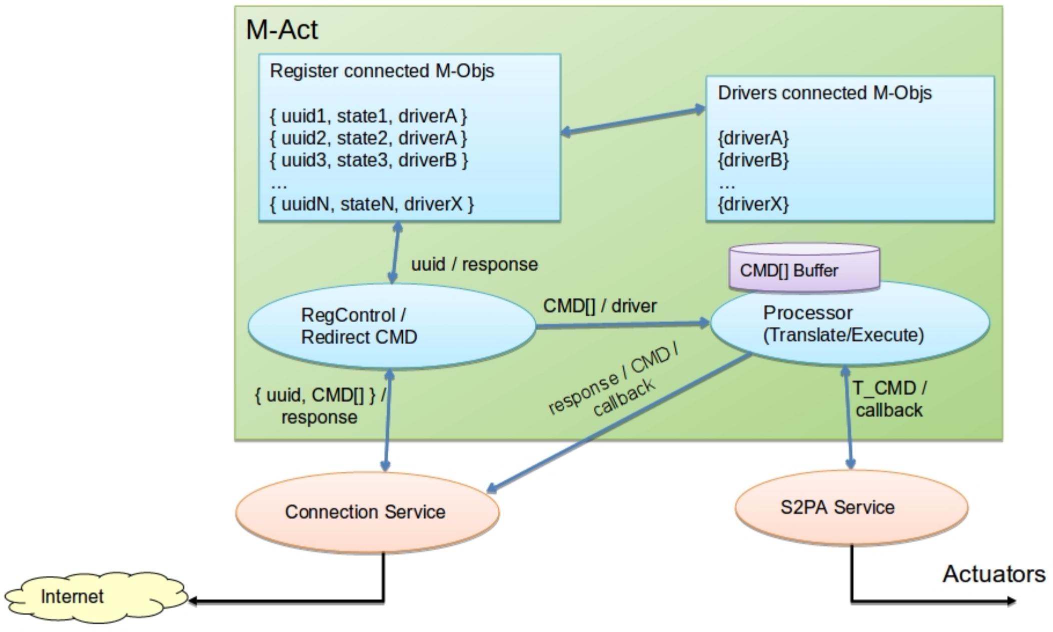
M-Act
Register connected M-Objs
{ uuid1, state1, driverA }
{ uuid2, state2, driverA }
{ uuid3, state3, driverB }
...
{ uuidN, stateN, driverX }
Drivers connected M-Objs
{driverA}
{driverB}
...
{driverX}
uuid / response
CMD[] Buffer
RegControl / Redirect CMD
CMD[] / driver
Processor (Translate/Execute)
{ uuid, CMD[] } / response
response / CMD / callback
T_CMD / callback
Connection Service
S2PA Service
Internet
Actuators

Figure 7: The M-Act architecture

a mix of Roll and SetRGB commands, and measured this round-trip delay of a command-and-acknowledgment transaction. The test environment had one M-Hub (on a Samsung Galaxy J5 2015 Android Marshmallow (6.0)) connected to a notebook (configuration i7-4510U, RAM 8GB, Dell Insiron 5447) both on WiFi in the same local area network. In these tests we measured following times: average 35.772ms, median 74.623ms and minimum time 29.763ms. Hence, if we subtract the 2ms that it takes to unlock the BB-8, we end up with even slightly smaller times.

In future, we plan to run several other performance tests, and make more thorough measurements. In addition, we want to start investigating the problems of actuation in the presence of sudden disconnections, automating actuation via scripts high-level commands, and coordinated actuation on two BB-8s that we have in the lab. By this, we will circumvent the limitation of the official Sphero mobile app, which allows only control of a single BB-8. But we want to enjoy seeing our robots move in sync.

6 Related Work

There are many approaches to middleware for IoT [14, 7, 12]. Some of them concentrate more effort in specific challenges, such as security and interoperability, others are focused on specific domains, such as Smart Cities applications [3], while others provide a more comprehensive support for IoT application development. There are also approaches to IoT as a Service [2]. However, many of them do not consider mobile nodes, do not consider movable smart objects or do not scale. We are unaware of a systematic approach to a scalable middleware architecture focused on the Internet of Mobile Things and that supports a uniform treatment of remote actuation over smart things with actuators.

The application of smartphones to enhance the monitoring and control of facilities has attracted academic attention [10]. Indeed, there are proposals that present integration solutions using smartphones to ambients where there moving by[10]. These proposals often take into consideration several sensors and some actuators[4]. The majority of those ambients are usually smart homes [9][13][17]. Some works threat open spaces with equipment like machines [11].

In the best of our knowledge, those proposals have a low level of abstraction between the smartphone software and the hardware to sense and control. Is necessary a previously know the device driver of the hardware. Moreover, the smartphone needs to coordinate the processing fully, without providing remote interaction. The following works present a more dynamically way to provide the use of the smartphone in such applications and scenarios.

Aloi et al. proposed architecture for communication between clients (as tablets and smartphones) and IoT devices (sensors and actuators)[1]. Their architecture suggests three layers: (1) Clients with mobile communication such as WiFI, 3G, and LTE; (2) Wireless Gateways and (3) IoT devices. The first and third layers regard the off-the-shelf equipment. The second layer is the abstraction layer that provides services from the IoT layers to the Clients layer. The Wireless Gateway has a local IoT devices configuration data bank to structure their interaction with devices. To describe the services provided by the Wireless Gateway from the IoT devices interaction the authors used the Sensor Markup Language (SenML). Nevertheless, the SenML is restricted to sensors but actuators. Our approach is similar to Aloi et al. [1] regarding the idea to use a plain Language to describe how to connect to the IoT devices by describing the devices specify a protocol.

In work [5] Datta et al. present the concept of IoT being accessed by the Internet by smartphone acting as gateways. The goal of their approach in [5] is to provide a mobile gateway and an IoT Device Management node that can maintain a device protocol list and act as a buffer of collected data from sensors. This related work[5] does not threat actuators.

Both related works [1] [5] present the necessity of a central broker to manage the connected devices such as our proposal of the SOM. As well the need a central repository of drivers to connect IoT devices. Furthermore, both works and ours share the idea of the use of the smartphone as a gateway that controls the IoT interaction work. Our work proposes to divide the IoT interaction between the Smartphone (through M-Act) and a broker (inside the SDDL core).

7 Conclusion

We have presented our middleware extension to provide generic actuation support for the ContextNet IoMT. We have detailed the whole architecture of the middleware and the drivers. We have also implemented an application sample in order to demonstrate the feasibility of our approach.

This research is still in its initial phase and new factors will be analyzed. New device drivers and applications are under implementation, and will be used to evaluate the generic actuation capability of the middleware in large-scale scenarios with thousands of smart things. As another direction of future work the impact of the WPAN communication delay and its consequences on remote actuation will be analyzed for different types of applications and smart things.

References

[1] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio, “Enabling IoT interoperability through opportunistic smartphone-based mobile gateways,” *Journal of Network and Computer Applications*, vol. 81, no. October, pp. 74–84, mar 2017.

[2] T. Aubonnet, A. Boubendir, F. Lemoine, and N. Simoni, “Controlled Components for Internet of Things As-A-Service,” *Open Journal of Internet Of Things (OJIOT)*, vol. 2, no. 1, pp. 16–33, 2016. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201704244995

[3] P. Bellavista, C. Giannelli, S. Lanzone, G. Riberto, C. Stefanelli, and M. Tortonesi, “A Middleware Solution for Wireless IoT Applications in Sparse Smart Cities,” *Sensors*, vol. 17, no. 11, 2017.

[4] B. P. Chandra, K. Geevarghese, and K. Gangadharan, “Design and Implementation of Remote Mechatronics Laboratory for e-Learning Using LabVIEW and Smartphone and Cross-platform Communication Toolkit (SCCT),” in *2nd International Conference on Innovations in Automation and Mechatronics Engineering*, vol. 14, jan 2014, pp. 108–115.

[5] S. K. Datta, C. Bonnet, and N. Nikaein, “An IoT gateway centric architecture to provide novel M2M services,” in *IEEE World Forum on Internet of Things*, 2014, pp. 514–519.

[6] L. David, R. Vasconcelos, L. Alves, R. Andre, and M. Endler, “A dds-based middleware for scalable tracking, communication and collaboration of mobile nodes,” *Journal of Internet Services and Applications*, vol. 4, no. 1, pp. 1–15, 2013.

[7] F. Delicato, P. Paulo, and T. Vasconcelos, *Middleware Solutions for the Internet of Things*, ser. Springer Briefs in Computer Science. Springer, 2013.

[8] M. Endler, G. Baptista, L. D. Silva, R. Vasconcelos, M. Malcher, V. Pantoja, V. Pinheiro, and J. Viterbo, “ContextNet: Context Reasoning and Sharing Middleware for Large-scale Pervasive Collaboration and Social Networking,” in *Proceedings of the Workshop on Posters and Demos Track*. ACM, 2011, pp. 1–2.

[9] O. Ghabar and J. Lu, “The Designing and Implementation of a Smart Home System with Wireless Sensor/Actuator and Smartphone,” *INFOCOMP 2014, The Fourth International Conference on Advanced Communications and Computation*, no. c, pp. 56–64, 2014.

[10] O. Ghabar and J. Lu, "Remote Control and Monitoring of Smart Home Facilities via Smartphone with Wi-Fly," in *Proceedings of The Fifth International Conference on Advanced Communications and Computation*, Brussels, Belgium, 2015, pp. 66–73.

[11] K. Kim, D.-H. Park, H. Bang, G. Hong, and S.-i. Jin, "Smart coffee vending machine using sensor and actuator networks," in *IEEE International Conference on Consumer Electronics (ICCE)*, jan 2014, pp. 71–72.

[12] 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*, vol. 4, no. 1, pp. 1–20, 2017.

[13] R. Piyare and S. R. Lee, "Smart Home-Control and Monitoring System Using Smart Phone," *ICCA, ASTL*, vol. 24, no. April, pp. 83–86, 2013.

[14] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

[15] L. Talavera, M. Endler, I. Vasconcelos, R. Vasconcelos, M. Cunha, and F. Silva, "The Mobile Hub concept: Enabling applications for the Internet of Mobile Things," in *Pervasive Computing and Communication Workshops (PerCom Workshops)*, March 2015, pp. 123–128.

[16] R. O. Vasconcelos, L. Talavera, M. Roriz, and M. Endler, "An Adaptive Middleware for Opportunistic Mobile Sensing," in *12th IEEE Workshop on Managing Ubiquitous Communications and Services (MUCS 2015)*, Saint Louis, 2015.

[17] H. Wang, J. Saboune, and A. El Saddik, "Control your smart home with an autonomously mobile smartphone," in *Proceedings of the IEEE International Conference on Multimedia and Expo Workshops*, 2013.

AUTHOR BIOGRAPHIES

Sheriton Valim is System Analyst at JGP Wealth Management and M.Sc. student at the Pontifical Catholic University of Rio de Janeiro. His research interests are mobile robots control, pervasive distributed systems and distributed algorithms.

Matheus Zeitune is undergraduate student of Informatics at the Pontifical Catholic University of Rio de Janeiro. His research interests are mobile devices.

Bruno Olivieri is Doctoral candidate in the Department of Informatics of the Pontifical Catholic University of Rio de Janeiro. His research interests include UAV swarm control, mobile robots, distributed algorithms.

Markus Endler is Associated Professor in Informatics at PUC-Rio and Principal Investigator of LAC. He is also member of the steering Committee of INCT InterSCity. His research interests include mobile and pervasive distributed systems, IoT, mobile robotics and stream processing and reasoning.