Open Access

Open Journal of Internet of Things (OJIOT)
Volume 5, Issue 1, 2019

http://www.ronpub.com/ojiot
ISSN 2364-7108

Leveraging Application Development for the Internet of Mobile Things

Felipe Carvalho [A], Markus Endler [A], Francisco Silva e Silva[B]

[A] Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil, {fcarvalho, endler}@inf.puc-rio.br

[B] Laboratório de Sistemas Distribuíidos Inteligentes, PPGCC-PPGEE/UFMA Sao Luiz, Brazil, fssilva@lsdi.ufma.br

Abstract

So far, most of research and development for the Internet of Things has been focused at systems where the smart objects, WPAN beacons, sensors, and actuators are mainly stationary and associated with a fixed location (such as appliances in a home or office, an energy meter for a building), and are not capable of handling unrestricted/arbitrary forms of mobility. However, our current lifestyle and economy are increasingly mobile, as people, vehicles, and goods move independently in public and private areas (e.g., automated logistics, retail). Therefore, we are witnessing an increasing need to support Machine to Machine (M2M) communication, data collection, and processing and actuation control for mobile smart things, establishing what is called the Internet of Mobile Things (IoMT). Examples of mobile smart things that fit in the definition of IoMT include Unmanned Aerial Vehicles (UAVs), all sorts of human-crewed vehicles (e.g., cars, buses), and even people with wearable devices such as smart watches or fitness and health monitoring devices. Among these mobile IoT applications, there are several that only require occasional data probes from a mobile sensor, or need to control a smart device only in some specific conditions, or context, such as only when any user is in the ambient. While IoT systems still lack some general programming concepts and abstractions, this is even more so for IoMT. This paper discusses the definition and implementation of suitable programming concepts for mobile smart things - given several examples and scenarios of mobility-specific sensoring and actuation control, both regarding smart things individually, or in terms of collective smart things behaviors. We then show a proposal of programming constructs and language, and show how we will implement an IoMT application programming model, namely ObsAct*, on the top of our current middleware ContextNet.*

Type of Paper and Keywords

Regular research paper: *IoMT, mobile smart things, Programming Model, Middleware, ContextNet*

1 Introduction

Recent development and advances in the Internet of Things (IoT) have introduced several challenges for application developers [4]. In this sense, IoT scenarios have evolved from simple sensor readings (e.g., measure the temperature of a room) or sending single actuating commands (e.g., unlock a door), to the establishment of complex applications that combine multiple services of sensors and actuators. We will refer to these sensors and actuators as smart things (or smart objects) throughout this paper. The development of IoT applications requires a series of non-trivial steps that include: the deployment

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2019)* in conjunction with the VLDB 2019 conference in Los Angeles, USA. The proceedings of VLIoT@VLDB 2019 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

of smart objects, the discovery of services, configuration, subscription to services, and the effective use of these services to build meaningful applications.

Cloud and Fog computing are critical enablers for large-scale IoT applications, as they offer the necessary infrastructure to support scalability in terms of device virtualization, provisioning of virtual sensors and actuators, and providing proper execution infrastructure for complex IoT applications. We envision some IoT applications where smart objects are distributed throughout the environment (such as along streets, inside buildings, houses, yards, rural areas), and can be stationary or mobile. To connect these smart objects to the Internet in an opportunistic way, we need intermediary devices called hubs, which can also be mobile. This approach is termed opportunistic because of the spontaneous and dynamic nature of encounters between smart objects and hubs, which are possibly short-lived, given mobility aspects. These mobile hubs (M-Hub) [10] can scan for smart things using a Wireless Personal Area Network (WPAN) technology, such as Bluetooth Low Energy (BLE), and expose them to the Internet.

Moreover, these M-Hubs are devices that may offer processing capacities (e.g., Smartphones, Raspberry Pi) and can be programmed to not only act as dispatchers of raw data to the cloud but also perform some operations within the data (e.g., aggregation, filtration). As an example, let us suppose a user carrying her smartphone with an instance of this mobile hub executing. As she passes by some places, her smartphone opportunistically detects and exposes the services of discovered smart objects so that developers can use them to create applications.

A subset of IoT applications where devices can be mobile (the M-Hubs, as well as the smart objects themselves) is called the Internet of Mobile Things (IoMT). This concept poses several challenges for application developers as well as for middleware systems because of mobility's dynamic nature. For example, devices can connect and disconnect unpredictably from the infrastructure. Given this fact, middleware solutions need to deal with this dynamism, and since this is not directly related to the domain of the application, it is necessary to hide some aspects from developers like discovery and configuration.

In this sense, high-level programming models that abstract the mobility of devices and network dynamism are needed to tackle the challenge of developing IoMT applications. IoT developers might be experts in a specific application domain but less experienced as software developers. Therefore, they need tools that allow them to express concepts of the real world into coding languages easily. In this work, we are interested in the subset of IoT applications that can be implemented using rules that have observation inputs (e.g., sensor readings) and actuation outputs (e.g., actuation commands and notifications). IoT sensors readings may comprise a stream of events that can be processed to create meaningful applications. Rule-based and Stream-based engines are potent tools for IoT application development as they offer the support to automated, timely reactions of smart objects to situations of interest occurring in the physical world [7]. The reaction is done by continuously analyzing and integrating low-level sensor readings, to subsequently fire an actuation command or a notification event.

A developer would like to be able to write application rules into machine code to further implement an application. Therefore, we need to observe some details: (i) How to discover and connect to these sensors and actuators? (ii) How to configure these objects? (iii) How to collect data from sensors or send commands to the actuators? (iv) How to collect and dispatch data to consumers? (v) How to specify data access control? (vi) How to implement QoS parameters? (vii) Where is data pre-processed and processed (at the edge or the cloud) and how to deploy the code to fulfill these tasks? A high-level programming model would help the developer to build an application without concerning about the details mentioned above.

In this paper, we present our approach for high-level IoMT application programming, namely OBSERVATION-ACTUATION LOOP (OBSACT). Our solution is based on the extraction of concepts that rule IoMT. The remainder of this paper is structured as follows. In Section 2, we present some common scenarios in IoMT. Section 3 presents some challenges of IoMT programming. We proceed to Section 4, where we discuss our approach, namely OBSACT. In Section 5, we present ContextNet middleware and IoTrade service, used to implement this work. In Section 6, we present a proof of concept evaluation of the programming model. In Section 7, we show some related work and finally, in Section 8, we present the conclusions and next steps of our work.

2 SCENARIOS

In this section, we present some scenarios of IoMT applications. By investigating these scenarios, we will be able to present and discuss the main concepts that underlie most IoMT interactions. We can summarize an IoMT application as a composition of places and elements - either stationary or mobile - that interact within the desired domain. An IoMT developer will have to specify the relevant relations and interactions that make up the desired functionality. These IoMT elements are listed below:

- **Environment**: It is the physical location (i.e., a place in the world, such as a room, a corridor, a street) in which sensors, actuators, and other entities are located at a given time T. Also, an environment is a place where dynamic IoMT interactions may occur. The interactions are considered to be dynamic because the entities that are inside an environment are possibly mobile and are temporarily associated with that specific place. Nevertheless, the smart objects and entities can also be fixed (i.e., not mobile) at that place.

- **Entity**: An entity can be a human being that carries a smartphone, which may also function as a mobile hub (see below). It may also be an object, such as electronic devices (e.g., air conditioner, television), mobile robots, drones. Entities can move around environments and can interact with sensors and actuators.

- **Smart Object**: Also called Smart Thing. It is an object that contains one or more sensors and/or actuators (see below). Smart objects are the key enablers of IoT applications, as they are autonomous physical/digital objects augmented with sensing, processing, and network capabilities [5]. Also, they make it possible to interpret what is occurring within themselves, the environment and the entities, intercommunicate with each other, and exchange information with people.

- **Sensor**: Part of a smart object capable of measuring physical variables, like temperature, humidity, luminosity. IoT applications use data gathered by them.

- **Actuator**: Part of a smart object that receives commands. These commands trigger actions in entities, like locking a door, adjusting a thermostat, making a robot move forward. Actuators are the targets of IoT applications, thus receiving the outputs of processing to do some action within an environment or an entity.

- **Beacon**: Element that emits short-range (few meters) radio frequency announcements. It can be used to detect proximity or co-localization (e.g., a person with a smartphone is in the range of a specific room, that is, inside of the room). Beacons use technologies such as Bluetooth Low Energy (BLE), Near Field Communication (NFC), Radio Frequency Identification (RFID). In our proposed modeling, beacons are used to identify environments or entities (e.g., the beacon of a room, or the beacon of a person).

- **Mobile Hub**: An element that can scan for smart objects and beacons in short-range and expose their capabilities and data by being connected to the Internet. To do so, Mobile Hub must be equipped with short-range wireless technologies, such as Bluetooth, and Internet connectivity, such as Wi-Fi and 4G LTE. In some cases, it is also capable of processing data locally, transmitting more complex information, and not only raw data. A mobile hub can be a smartphone, or microcomputers such as Raspberry Pi, BeagleBone, Dragonboard.

By analyzing these elements, we can implement IoMT applications using mechanisms that execute actuation commands, according to a specific observation within some context. Context is a research field that can be defined as a set of relevant information about an environment or an entity [2]. The most common form of context information is the location (either symbolic or geographical). Symbolic location is relative information about a place (e.g., Room 101 at the Informatics Department of PUC-Rio), while geographical location is defined by Latitude and Longitude coordinates acquired from a GPS sensor. Also, the context can be time-related, which can be a date and time interval (from 13:00 to 14:00 at March 13^{th} of 2019), a fixed timestamp (13:00 at March 13^{th} of 2019), or even a relative date and time (3 minutes after an event occurs). Context does not only involve time or location awareness. Other examples: The quality of sensor measurements and relevant metadata about the service; The logical relation between objects, like the proximity, or even the absence of one kind of object in some place.

For a reactive system, it is natural to implement some rule that reacts to events, thus creating an observe-and-actuate paradigm. Events are atomic occurrences of physical observations made by sensors, plus the context information associated with them. We can define reactive rules that trigger output actuation commands, with observation events coming from sensors contained in entities or environments acting as inputs. In the following, we will describe some IoMT remote monitoring and control scenarios.

Adjusting a room's air conditioner thermostat: Let us suppose one is interested in monitoring the temperature (environmental variable) in a certain room (location context) so that an application can adjust the thermostat (actuation command) of the air conditioner of that room. In this case, the temperature sensor readings comprise a stream of input events, and the actuation command would be the output event.

The process, shown in Figure 1, proceeds as follows: (1) Someone enters the environment and her Mobile Hub detects the beacon associated with that room; (2) The

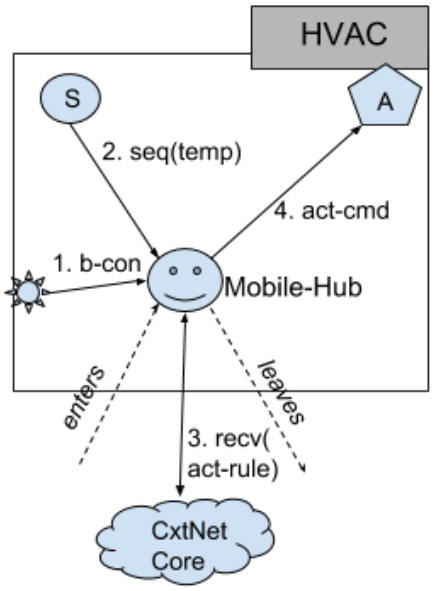

Figure 1: Scenario of a temperature sensor and a thermostat that regulate an air conditioner

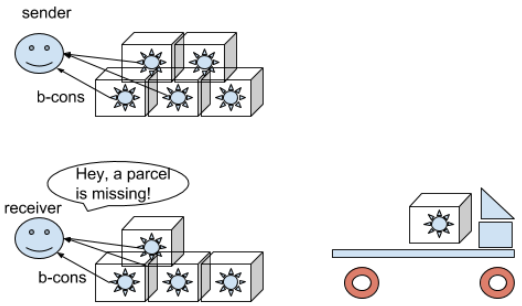

Figure 2: Scenario of a truck being loaded and unloaded with parcels (which should be transported together

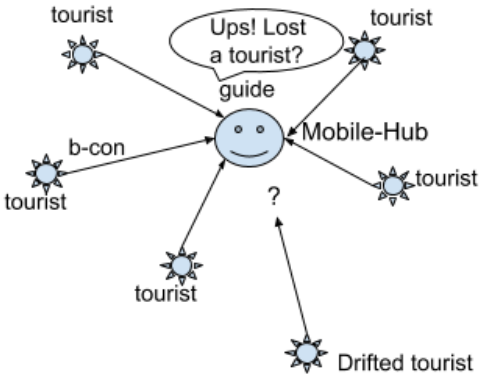

Figure 3: Scenario of a tourist guide and some tourists that should stay near the guide

mobile hub receives the temperature readings from a sensor in that environment; (3) When the temperature reading is above a threshold (e.g., 28^oC), a detection rule is triggered; (4) A Mobile Hub, that can be the same that received the sensor's readings or a completely different one, sends an actuation command to the thermostat, to set the temperature to 22^oC, for example. It is a simple reactive rule that measures an environmental variable and actuates in a device, so that a new environment state is achieved, in this case, to lower the temperature.

The scenario presented above consists of a single observation and a single actuation command. However, these observe-and-actuate rules can be more complex. Let us consider other scenarios.

Logistics Monitoring: IoMT applications also lend themselves to monitor and check the co-localization and/or co-movement of mobile entities (e.g., people, cars). Let us visualize the following scenario. A truck of a logistics company is carrying some parcels for delivery. A beacon identifies each parcel, and all of them must be transported together, as shown in Figure 2. If it would happen that the truck is opened and some of the parcels are removed, the IoMT application could trigger an alert for the postmen or the insurance company. In this case, the trigger fires when one or more parcels are not in the predefined group.

Tourists Guided Tour: As another example, consider a group of tourists and one tourist guide, all visiting a city. Now assume that due to some reason it is essential that all tourists always stay within a few meters from the guide (e.g., the security of the tourists that do not speak the native language, or simply to keep the schedule of the planned visits). In order to continuously check such collective "movement restriction", tourists may carry beacons that announce their proximity to the guide's smartphone. In a case that some of the tourists happen to drift away, this could be immediately notified at the guide's smartphone with, tourist name, the time and the exact place (context) in which this separation happened. This scenario is illustrated in Figure 3

Formation Flight of UAV Swarms: Unmanned Aerial Vehicles (UAVs) may be deployed in swarms, flying in specific formation patterns (e.g., line, square, finger-four). This is useful for applications that, for example, have to video scan a forest area or collect data from wireless sensors on the ground. To keep a formation pattern, there must be a leader, and each other UAV should stay within a defined range in relation to the leader. We assume that the UAVs have means to measure the distance in relation to each other, by using

sonar sensors or video processing techniques, for example. The actuation commands in this example are adjustments for maintaining the pattern, or even changing it.

All of the above scenarios introduce some challenges of IoT with mobility that will be discussed in Section 3

3 Challenges of IoMT Application Programming

By using a rule-based approach for IoMT programming, some monitoring control rules have to be generic for many types of environments, i.e., without the need to indicate specific identifiers (ID numbers) for smart objects and beacons. Application developers would like to focus on domain-specific problems for their IoMT applications. For so, they could delegate the discovery, selection, and configuration of smart objects to a middleware solution. With all of these steps being settled by the middleware, they could use a high-level programming model to express the functionalities they need.

Any internet connectivity of sensors, beacons, and actuators depends on the presence of a nearby smartphone running the Mobile Hub. Also, any location information is provided by a Mobile Hub. In the thermostat scenario presented in Section 2, a Mobile Hub is needed to enable the environment control at an actuator according to the data received from the corresponding sensor. This means that the Mobile Hub must be able to detect its location and inject this information into sensors and actuators events that are sent to the cloud.

Due to the large number of environment variables and entities to be controlled, and the uncertainty about which Mobile Hub will be the enabler of the control, corresponding drivers for communication with sensors and actuators have to be fetched on demand from the cloud service. After a Mobile Hub is selected by the middleware to deal with an observation or an actuation rule, drivers are fetched dynamically so that it can communicate appropriately with smart objects. For scalability reasons, they cannot store all drivers locally for all types of devices. Therefore, a cloud service that stores drivers is needed to solve this problem.

The unrestricted mobility paradigm, where every smart object and Mobile Hub are possibly mobile, poses some challenges for the development of applications. As there is no guarantee that an object is at a specific place, middleware solutions have to continuously and opportunistically check the environment for new sensors, beacons, and actuators. The conditions programmed by a developer must specify how much time a command should wait for the arrival of smart objects and beacons.

In the example of the group of tourists that should stay together, presented in Section 2, the programming of the application should allow the specification of a dynamic set of beacons representing the tourists. Also, it should guarantee that the proximity of all the registered beacons is continuously verified. The actuation command or notification should be only about the beacon that drifted away, but it should happen as soon as possible to the guide. Also, in this application, there must be some way to enable and disable the monitoring of the group vicinity, due to the common habit of introducing some group dispersal intervals.

4 Observation and Actuation Approach (ObsAct)

To facilitate IoMT programming, underlying details should be transparent to developers. For some applications, developers are interested in solving a specific domain problem, and they are prone to worry less about which specific sensor they are using, i.e., which identifiers (IDs) belong to that sensor. Also, for these applications, developers are not interested in how the middleware will deal with selection and communication with smart objects. In the thermostat regulator scenario, presented in Section 2, application developers would like to be able to express their requirements as close as possible to their natural language. An English formulation could be: "I want to adjust the temperature of the air conditioner to 22^oC when the temperature rises above 28^oC." We emphasize here that Natural Language Processing (NLP) is not the scope of this work. Instead, we would like to have programming constructs that will allow developers to express their application requirements.

To deal with application requirements, in this Section, we present our programming model, namely Observation-Actuation Loop (ObsAct). As the name infers, this model allows to declaratively express the focus of application demands, in terms of observation and actuation rules. The proper selection of smart objects with the given context is made by the Observation mechanism of ObsAct, as well as the reaction chained by these observations, thus creating an Actuation command. We present the abstractions used for the proper selection of sensors and actuators, given some context parameters, so as the mechanisms for reaction to these events.

The nature of IoMT applications is essentially a stream of events. An application consists of a set of rules that focus on an event stream. Actuation commands are output events generated from the comparison of attribute values within a designated time window. This model is essentially Complex Event Processing (CEP), as a stream of single events can be composed to generate complex events. CEP is a programming paradigm that allows reactions to a stream of event data in real time [6] by using continuous queries and inserting data through them.

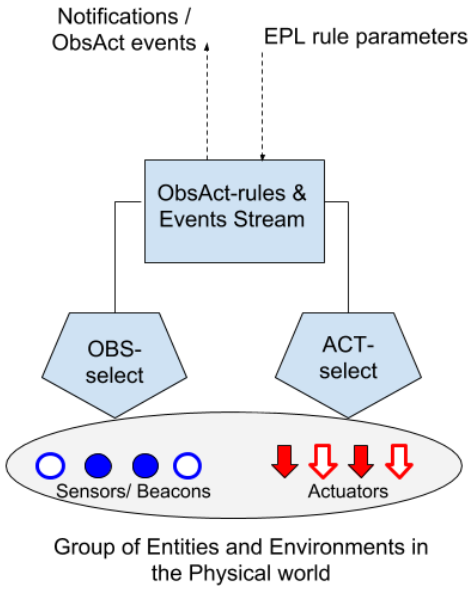

Figure 4: ObsAct architecture

We use EPL[1] (Event Processing Language) notation in this work, but this same model could be written in another CEP language.

OBSACT architecture is shown in Figure 4. The developer implements her application using EPL rules that are the inputs of OBSACT. Then, the OBS-select component maps the selections of beacons and sensors that the developer coded for use in the application. The ACT-select component does the equivalent of the OBS-select, but for actuators that matched the selection.

Listing 1 shows the basic object types that are present in OBSACT. Essentially, we have sensors, beacons, environments, and actuators events that are generated and inserted into the engine of OBSACT. Beacons are associated to Environments by their *placed_at* attribute, that refers to a valid *env_id*. Sensors and Actuators are associated to Environments by the *at_environment* attribute, that refers to a valid *env_id*.

The Observation and Actuation events are generated by the developer, where they express their application requirements. Listing 2 presents an example of observation rule, written in EPL, that inserts an Observation event into the CEP engine. In this example, the rule is looking for temperature sensors that are measuring above 30^oC in an environment named "LAC". This rule uses a time window of 60 seconds and outputs the first occurrence inside this window. Listing 3 shows

[1] EPL is the language used in Esper CEP Engine - http://www.espertech.com/esper/

```
1 Sensor(sensor_id string, sensor_name string
      , sensor_value float, at_environment
      string, timestamp java.util.Date);
2 Beacon(beacon_id string, rssi int,
      placed_at string, timestamp java.util.
      Date);
3 Environment(env_id string, type string,
      env_name string);
4 Actuator(act_id string, act_name string,
      at_environment string, act_command
      string);
5 Observation(env_id string, sensor_id string
      , beacon_id string, value float,
      sensor_name string);
6 Actuation(target string, command string,
      arguments string);
```

Listing 1: ObsAct schema

```
1  @Name('Detect') insert into
2  Observation(env_id, sensor_id, beacon_id,
       value, sensor_name)
3      select e.env_id, s.sensor_id, b.
           beacon_id, s.sensor_value, s.
           sensor_name
4      from Sensor#time(60 seconds) as s,
5      Beacon#time(60 seconds) as b,
6      Environment#time(60 seconds) as e
7      where b.placed_at=e.env_id
8      and s.at_environment=e.env_id
9      and s.sensor_value>30
10     and s.sensor_name='temperature'
11     and e.env_name='LAC'
12     output first every 60 seconds;
```

Listing 2: Rule that detects temperatures above 30^oC inside a specific environment

the Actuation event, that is inserted into the CEP engine to set the value of the thermostat to 22. This Actuation is triggered when an Observation event is detected inside the time window of 60 seconds.

We can notice at examples in Listings 2 and 3 that no specific identifier of a smart object had to be hard-coded into the rules. The developer of these rules had to only concern about which kind of smart objects they wanted, which environments, and what were the values for the observation and actuation commands.

Listing 4 presents a rule that continuously checks for beacons signals. This is the case of the Tourists Guided Tour presented in Section 2. Mostly this rule checks, for each beacon event occurrence, if the same beacon is not seen at least one time in the next 10 seconds. Also, there is a check if that beacon belongs to that particular application, i.e., if it is into the BeaconsList window. The

```
1 @Name('Trigger') insert into
2 Actuation(target, command, arguments)
3 select act_id, act_command, '22'
4 from pattern [every obs=Observation(
      sensor_name='temperature') where timer:
      within(60 seconds)]#time(60) as pat,
5 Actuator#time(60) as act
6 where pat.obs.env_id=act.at_environment
7 and act.act_name='thermostat'
8 and act.act_command='set_value'
```

Listing 3: Rule that triggers an actuation command to set thermostat value to 22

```
1 @Name('Detect') select b1.*
2 from pattern
3 [every b1=Beacon ->
4 (timer:interval(10 sec)
5 and not b2=Beacon(b1.id=b2.id))]#time(10),
6 BeaconsList#keepall() as bl
7 where b1.id=bl.id;
```

Listing 4: Rule that triggers when a beacon of a defined group is not seen in the past 10 seconds

output actuation event for this rule could be to send a notification to the guide's smartphone.

As any objects can be mobile in an IoMT paradigm, these rules are ready to be deployed in scenarios of unrestricted mobility. The event stream processing model works with the concept of windows, that can be related to time or the number of events. When a developer deploys a CEP rule, it is instantiated in the engine. The engine then continuously checks for events that satisfy the conditions. As this checking is continuous, it fits mobility scenarios naturally, because new sensors and actuators may be discovered opportunistically [1] by mobile hubs and thus satisfying the rules at some point.

5 Implementation Architecture

Our solution was developed on top of a middleware called ContextNet. ObsAct is an abstraction layer that maps the ObsAct rules into more basic criteria for entity selections, wireless communications, and stream processing at the middleware layer. In this Section, we will focus on describing the ContextNet middleware, so as the IoTrade service. Both were used as the basis for our implementation.

5.1 ContextNet Middleware

ContextNet [3] is a scalable mobile-cloud middleware for the Internet of Things. Its unique characteristic is that it employs Mobile hubs (M-Hub) executing on smartphones, as the means of discovering smart objects with Bluetooth Smart radio and let them communicate with backstage IoT services hosted in a cloud/cluster, and among themselves, all through the cloud. Of course, the Mobile Hub can also execute on stationary SoC boards such as Raspberry Pi or ESP32, in which case they assume the role as conventional, stationary hubs for Bluetooth devices.

However, the M-Hub's potential mobility opens room for remote monitoring, remote control and data analytics of any such smart thing with sensors, beacons or actuators, be them stationary or in motion, through popular smartphones, thus enabling the sort of IoMT applications described in Section 2.

Bluetooth Low Energy (BLE) is emerging as a promising short-range, low-power connectivity technology (WPAN) for IoT. The reason is that it is embedded in almost all models of smartphones, and also being integrated into a growing assortment of smart devices, like bulbs, locks, smartwatches, fitness and activity monitors, and tiny beacon tags and sensor devices.

By using ContextNet, it is possible to associate any reachable mobile objects (M-OBJ) with a geographic position that will be provided by the connected M-Hub, either by its GPS sensor - if outdoors, or by a previous encounter with a BLE beacon, if indoor. On the other hand, the various communication modes among M-Hubs, mediated through Scalable Data Distribution Layer (SDDL) core services, will enable rich forms of M-OBJ search, sensor data subscription and remote monitoring and control of actuators embedded into smart things.

Through the dynamic discovery and connections with Bluetooth Low Energy (BLE)-enabled devices and support for stream processing using Complex Event Processing (CEP) technology at the mobile hubs (i.e. smartphones), as well as in the cloud, ContextNet supports IoMT applications that embrace mobility and intermittent connectivity in the edge networks, i.e. of the hubs, of sensors, actuators, and beacons. By providing CEP edge computing, ContextNet also allows much reduction of the Internet traffic of raw data to/from the cloud-hosted services.

There are three types of IoMT applications that best served by the ContextNet middleware:

1. *Localized IoT applications*, where the user's smartphone connects to and interacts directly with BLE Sensors, beacons or actuators in its vicinity, for example, at home or in his/her office (cf. Figure 1);

2. *Participatory IoT applications*, where regular smartphone users may get incentives – or even monetary benefits - for donating their 3G/4G connectivity and a portion of their device's energy budget for hauling sensor data, beacon-IDs or actuation commands to/from the cloud. Since ContextNet's mobile hub software runs in background, it is independent of the active mobile apps, and it has shown to drain only a little the smartphone's energy budget [10].

3. *Embedded mobile IoT applications* where the Mobile Hub runs on small SoC boards (RaspberryPI, ESP32 or a Qualcomm Dragonboard) which may be within a vehicle, a mobile robot or a drone, and is able to monitor/control smart things both inside the vehicle (e.g. temperature sensor) or close to where the robot/drone is (e.g., humidity sensors in precision agriculture).

There exists a large number of other middlewares for IoT, but none of them utilizes regular Android-based devices as communication hubs, offering also stream processing capabilities for IoMT through the powerful Complex Event Processing technology. Moreover, this hub-based stream processing capabilities are fully integrated with the data stream processing and IoMT control services running in the cloud. Examples of such cloud-hosted services of ContextNet include a NoSQL distributed persistency service and a location-aware group communication service.

The middleware ContextNet has a micro-service architecture, both for the backstage services in the cloud, as well as in the mobile hub component, making all its functions modular, exchangeable, and highly scalable in respect to the communication among the smart things and the data processing of their sensor or state data.

5.2 IoTrade Service

One of the central ContextNet components used by OBSACT is a distributed bidding and matchmaking service named IoTrade [9]. This service allows IoMT application components to select Mobile hubs and smart (mobile) devices dynamically, e.g., their sensors and actuators, or else, data stream processing elements along with the entire network of all ContextNet-connected devices, according to *IoTrade customer* demands. These customers demands mostly are: "target zones" (i.e., geographic regions), data transmission quality parameters, sensor data, and actuation quality (QoS) and cost parameters.

IoTrade was originally conceived as a multi-sided platform for trading IoT sensor data and control of smart devices between their owners and clients interested in paying for an IoT monitoring and control service spanning a specific geographic region.

For example, some IoT customer - a person or company - needs to check the current weather conditions (e.g., if there is black ice on a section of a road), because it has to dispatch an urgent delivery of goods. So, s/he marks this place as a "target zone" in IoTrade, indicating the need to get temperature, humidity and precipitation data with high precision and accuracy. This request is then advertised to all data and connectivity providers (e.g., smartphone owners with the M-Hub) and providers of corresponding roadside sensors in the target zone so that these can give their bid. All the delivered bids are then sent to the IoTrade server, which will do the matchmaking and composition of providers that best fit the parameters that the client has specified. These attributes and the target zone determine that such a request is delivered only to providers that have appropriate sensors or actuators in the specified geographic zone. Moreover, only the devices are selected whose wireless connectivity quality (i.e., in terms of latency, reliability, throughput), and sensor data accuracy, actuator quality, or data analysis matches the quality requirements specified in the customer request.

Following the same process, OBSACT will act as the IoMT customer, will determine the "target zone" (e.g., room LAC) and set the required quality parameters for sensors and actuators (e.g., temp-sensor has accuracy level 8), and IoTrade will deliver the UUIDs of the selected devices back to OBSACT. With these identifications, the system can then establish the connections to the sensors and actuators and start receiving the stream of sensor data and/or sending corresponding actuation commands to the devices to control.

5.3 OBSACT Implementation

Observation and Actuation Rules are deployed at the OBSACT CEP engine, as shown in Figure 5. For each deployed rule, a corresponding listener is instantiated, which allows a reactive execution. Each rule contains a desired logical condition between observation inputs along with location, time, and other context information. The location context is used by the Reference Selection component to build a list of candidate Mobile Hubs that could be used for the application. The Mobile Hub Registry is a component that catalogs all registered Mobile Hubs. As an example, if an application is defined to run at a specific university, only the Mobile Hubs that are geographically inside that region can be selected.

At the bottom of the architecture, sensors (e.g., S1 and S2), actuators (e.g., A1 and A2), and beacons (e.g., B1 and B2) are discovered by selected Mobile Hubs opportunistically. After the discovery of smart objects,

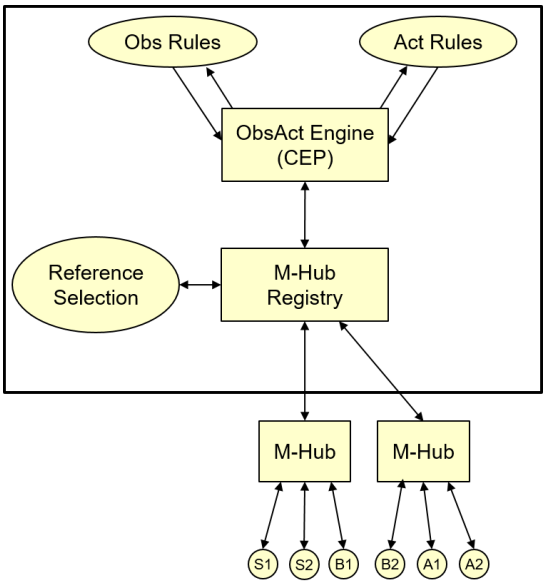

Figure 5: ObsAct implementation architecture

their data are made available as a stream of events to the OBSACT Engine. Each sensor event is an atomic occurrence of a sensor reading. Similarly, each actuator event is one of the possible functionalities that the actuator can perform. Also, each beacon event is the detection of a beacon that can represent a location or an entity. All of the mentioned events have a timestamp of occurrence.

As the stream of events passes by the CEP engine, it checks if the deployed rules are satisfied. When an observation rule is triggered, this subsequently triggers a referenced actuation rule. The callback of the actuation rule is to effectively send the command to an actuator or a notification to some defined target.

Each smart object that has actuators uses an open or proprietary protocol for actuation and control. For this, drivers implement these controls for each different device. To deal with this communication, we used M-ACT [11], a middleware that supports generic actuation commands for IoT devices. M-ACT allows to fetch drivers on demand from the cloud and install them on the Mobile Hubs that will send the actuation commands. These commands are generic (e.g., turn on, turn off, set temperature, increase the level.) and are converted to specific instructions for each device.

6 EVALUATION

The evaluation of OBSACT for this work consisted of a proof of concept implementation. For this, we generated a set of events and simulated their arrival in a timeline. We used Esper Online Notebook[2], that is a tool for prototyping CEP rules and run a stream of events with functions to simulate time advancement.

We used the scenario presented in Listings 2 and 3 from Section 4, where the application needs to observe temperature sensors that measure above 30^oC in an environment named "LAC", and when this triggers, an actuation command is fired to set the thermostat to 22^oC. Listing 5 presents the list of events used for this simulation. For the sake of readability, some attributes were omitted. This simulation starts with two beacons, two identifiers of environments, two sensors, and one actuator. This set of events generate one Observation event, shown in Listing 6, which triggers at the Sensor event in line 6, where the temperature value is 40^oC. This Observation event triggers one Actuation Event that sets the thermostat value to 22^oC. After 600 seconds, a new Observation is triggered, this time with a temperature of 35^oC, which generates another Actuation event.

EPL rules represent a high-level abstraction for developers, as they allow to express situations of interest in terms of sensor and actuator selection, along with context information. After the step of Observation and Actuation, shown in the simulation of this Section, OBSACT will use these events to dispatch the commands for the ContextNet middleware to execute the actuation effectively. This step of execution of actuation commands is out of scope for this paper.

7 RELATED WORK

In this Section, we show some related works that also propose programming models for IoT, and have similarities to ObsAct Approach. Each of these works addresses some aspect towards an IoT programming model.

Nastic et al. (2013) [8] implemented a framework named PatRICIA (PRogramming Intent-based Cloud-scale IoT Applications). According to the authors, PatRICIA aims to provide an ecosystem for development and provisioning of cloud-scale IoT applications. The framework is based on two primitives: (1) Intent, that is used to represent tasks that will be executed in an environment (e.g., to collect sensor data or execute an actuation command); (2) IntentScope, that represents a group of entities, which share some common properties (context). The framework then associates Intents and IntentScopes to create applications.

Khaled et al. (2018) [4] introduce an IoT device description language (IoT-DDL), for their project named Atlas Thing Architecture, that presents three primitives: (1) Thing Service, as an abstraction on the function offered by a "thing"; (2) Thing Relationship, as an

[2] https://notebook.esperonline.net/

```
t = '2019-03-31 08:00:00.000'
Beacon={beacon_id='bid01', placed_at='
    room01'}
Beacon={beacon_id='bid02', placed_at='
    room02'}
Environment={env_id='room01', env_name='LAC
    '}
Environment={env_id='room02', env_name='
    Telemidia'}
Sensor={sensor_id='sid01', sensor_name='
    temperature', sensor_value=40.0,
    at_environment='room01'}
Sensor={sensor_id='sid02', sensor_name='
    humidity', sensor_value=90.0,
    at_environment='room01'}
Actuator={act_id='aid01', act_name='
    thermostat', at_environment='room01',
    act_command='set_value'}
t=t.plus(600 seconds)
Beacon={beacon_id='bid01', placed_at='
    room01'}
Environment={env_id='room01', env_name='LAC
    '}
Sensor={sensor_id='sid01', sensor_name='
    temperature', sensor_value=35.0,
    at_environment='room01'}
Actuator={act_id='aid01', act_name='
    thermostat', at_environment='room01',
    act_command='set_value'}
```

Listing 5: Timeline of simulated events

```
Observation={env_id='room01', sensor_id='
    sid01', beacon_id='bid01', value=40.0,
     sensor_name='temperature'}
Actuation={target='aid01', command='
    set_value', arguments='22'}
```

Listing 6: Triggered Observation and Actuation events

abstraction on how different thing services are linked together; and (3) Recipe, as an abstraction on how different services and relationships build up a segment of an application. According to the authors, the IoT-DDL is a schema used to describe, through a set of attributes and parameters, the thing in a smart space in terms of the set of resources, inner entities, cloud-based attachments, and interactions that engage the thing with other things and cloud platforms. Their work focus on the capability of vendors to describe services and relationships of a thing, using IoT-DDL; the capability of the framework to connect different primitives and operators to build an IoT application; and the capability of a thing to generate services and formulate appropriate APIs.

Although these works present a programming model for IoT, they do not handle the mobility aspects of smart objects and edge devices (hubs) nor offer explicit support for this paradigm. One aspect of ObsAct approach when compared to the works presented above is that everything can be mobile. So, smart objects and mobile hubs can be associated with environments temporarily, and they can unpredictably change their context (e.g., move from one place to another). Also, our programming model is based on the well known CEP paradigm, which offers a structured language for querying streams of events. Developers write CEP rules to express the desired behavior for their applications. They do this by coding attributes and describing the relationship between observation and actuation events, thus associating a selection of smart objects with desired context. In this sense, mobility results in a dynamic and non-deterministic scenario, as the desired matching depends on the context at a given time.

8 Conclusion and Next Steps

In this paper, we propose ObsAct, an event-based approach for programming the continuous monitoring of - and actuation commands on - smart objects, with sensors, beacons, and actuators, that can be mobile or not. The framework is intended for, and heavily relies, upon the ContextNet middleware as it uses the concept of Mobile Hub and the IoTrade service.

The ObsAct approach allows the description of observation and automated reaction both over changes in fixed environments (e.g., in a smart building), as well as in collective location and mobility patterns, paving the way to radically new IoMT applications.

The ObsAct essentially borrows some concepts and the declarative rule-based paradigm of Context Event Processing [6]. Moreover, in particular, EsperTech's Event Processing Language for expressing the element's properties and the observation (OBS) and the actuation (ACT) rules. The closeness with the CEP language EPL was on purpose: it primarily facilitates to map (adapt and split) the ObsAct schema declarations and rules to the mobile and cloud CEP processing elements of a ContextNet deployment.

This mapping is not yet finished. In the current stage, we are still in the process of developing the software modules for this translation, splitting, and decentralized deployment.

As future lines of research, we envisage the possibility to define some language primitives that would describe relationships between elements that are commonly used, and that would thus simplify even more the ObsAct IoMT reaction logic.

Moreover, we think that IoTrade's QoS parameters - that allow defining the smart objects better to be selected

- has yet to be better explored, to give the ObsAct programmer a higher degree of control when many sensors or actuators are available.

Acknowledgements

This research was partially supported by CNPq scholarship 140914/2017-0 and grant Edital universal 2018.

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, pp. 74–84, 2017.

[2] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A survey of context data distribution for mobile ubiquitous systems," *ACM Computing Surveys (CSUR)*, vol. 44, no. 4, p. 24, 2012.

[3] M. Endler and F. S. e Silva, "Past, present and future of the contextnet iomt middleware," *Open Journal of Internet Of Things (OJIOT)*, vol. 4, no. 1, pp. 7–23, 2018, special Issue: Proceedings of the International Workshop on Very Large Internet of Things (VLIoT 2018) in conjunction with the VLDB 2018 Conference in Rio de Janeiro, Brazil. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201808051932326762 2857

[4] A. E. Khaled, W. Lindquist, and A. S. Helal, "Service-relationship programming framework for the social IoT," *Open Journal of Internet Of Things (OJIOT)*, vol. 4, no. 1, pp. 35–53, 2018, special Issue: Proceedings of the International Workshop on Very Large Internet of Things (VLIoT 2018) in conjunction with the VLDB 2018 Conference in Rio de Janeiro, Brazil. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201808051930228699 0058

[5] G. Kortuem, F. Kawsar, V. Sundramoorthy, D. Fitton *et al.*, "Smart objects as building blocks for the internet of things," *IEEE Internet Computing*, vol. 14, no. 1, pp. 44–51, 2009.

[6] D. Luckham and R. Schulte, "Event Processing Glossary - Version 2.0," p. 20, 2011.

[7] R. Mayer, B. Koldehofe, and K. Rothermel, "Predictable low-latency event detection with parallel complex event processing," *IEEE Internet of Things Journal*, vol. 2, no. 4, pp. 274–286, 2015.

[8] S. Nastic, S. Sehic, M. Vögler, H.-L. Truong, and S. Dustdar, "Patricia–a novel programming model for iot applications on cloud platforms," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*. IEEE, 2013, pp. 53–60.

[9] L. G. Pitta and M. Endler, "Market design for iot data and services the emergent 21th century commodities," in *2018 IEEE Symposium on Computers and Communications, ISCC 2018, Natal, Brazil, June 25-28, 2018*, 2018, pp. 410–415.

[10] L. E. Talavera, M. Endler, I. Vasconcelos, R. Vasconcelos, M. Cunha, and F. J. d. S. e Silva, "The mobile hub concept: Enabling applications for the internet of mobile things," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 2015, pp. 123–128.

[11] S. Valim, M. Zeitune, B. Olivieri, and M. Endler, "Middleware support for generic actuation in the internet of mobile things," *Open Journal of Internet Of Things (OJIOT)*, vol. 4, no. 1, pp. 24–34, 2018, special Issue: Proceedings of the International Workshop on Very Large Internet of Things (VLIoT 2018) in conjunction with the VLDB 2018 Conference in Rio de Janeiro, Brazil. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201808051932233723 2186

AUTHOR BIOGRAPHIES

Felipe Carvalho received his M.Sc. in informatics in 2017 from Pontifical Catholic University of Rio de Janeiro, Brazil. He is currently a researcher at Laboratory for Advanced Collaboration (LAC), and Ph.D. student in the Department of Informatics at Pontifical Catholic University of Rio de Janeiro. His active research interests include Internet of things, middleware for mobile device communication and programming models for IoT.

Dr. Markus Endler is associate professor at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio) and founder and head of the Laboratory for Advanced Collaboration (LAC). He obtained his doctoral degree from GMD Institute and TU Berlin (1992), and the Livre docente title (Habilitation) from the University of São Paulo (2001). So far, he has supervised 13 doctoral thesis, 29 M.Sc. dissertations, and co-authored more than 190 articles published in international journals and refereed conferences, as well as seven book chapters. His main research interests include distributed, mobile and pervasive computing, context-awareness, mobile coordination, Internet of Things and Smart Cities.

Dr. Francisco Silva e Silva is Associated Professor at the Federal University of Maranhão, and leads the LDSi lab. His research interests include distributed and ubiquitous systems with current emphasis in IoT and Smart Cities middleware.