ISSN 0103-9741

Monografias em Ciência da Computação

nº 04/2023

Design and Implementation of a Flexible Architecture for Mobile Edge Devices

Luis E. Talavera

Tatiana Reimer

Millena Cavalcanti

Gabriel Cantergiani

Marco A. Teixeira

Markus Endler

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 04/2023 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena May, 2023

Design and Implementation of a Flexible Architecture for Mobile Edge Devices

Luis E. Talavera, Tatiana Reimer, Millena Cavalcanti, Gabriel Cantergiani, Marco A. Teixeira and Markus Endler

luis.talavera@ucsp.edu.pe, tatireimer99@gmail.com, mcavalcanti@inf.puc-rio.br, cantergiani5@hotmail.com, marco.bteixeira@gmail.com, endler@inf.puc-rio.br

Abstract. The ever-growing use of smartphones, especially in urban centers, which are held/carried almost anytime and anywhere by users, combined with the devices' ubiquitous Internet connectivity and the presence of embedded sensors and short-range wireless interfaces (NFC, Bluetooth, Bluetooth Smart) now enables new kinds of participatory mobile IoT applications. In these, smartphones may act as universal hubs for interaction with wireless IoT devices, sensors, actuators - or mesh networks thereof - that have only short-range wireless connectivity. This paper describes the main features and architecture of our redesigned (mobile) edge middleware Mobile-Hub (M-Hub2). It is a Kotlin-based Edge component for Android devices (smartphones, etc.), that can discover nearby Bluetooth Low Energy (BLE) devices (sensors or actuators, beacons, etc.) and opportunistically connect them to back-end services and IoT applications. The main novelty of M-Hub2, compared to its original version is its modular software design and its ability to use multiple WWAN and WPAN protocols.

Keywords: Internet of Things. Middleware. Mobile objects; Mobile Sensing. Dynamic Adaptation; Remote Sensing and Actuation.

Resumo. O crescente uso de smartphones, especialmente em centros urbanos, onde os usuários os levam consigo praticamente o tempo todo, em qualquer lugar, combinado com a conexão onipresente à Internet dos dispositivos e a presença de sensores embarcados e interface sem fio de curto alcance (NFC, Bluetooth, Bluetooth Smart) permite agora novos tipos de participações de aplicações IoT móveis. Nelas, smartphones podem agir como hubs universais para interação com dispositivos IoT sem fio, sensores, atuadores - ou rede mesh dos mesmos - que possuem apenas conexão sem fio de curto alcance. Este artigo descreve as principais funcionalidades e arquitetura de nosso middleware edge (móvel) Mobile-Hub (M-Hub2). É um componente Edge baseado em Kotlin para dispositivos Android (smartphones, etc.), que descobre dispositivos Bluetooth Low Energy próximos (sensores ou atuadores, beacons, etc.) e oportunamente conecta eles a serviços back-end e aplicações IoT. A principal novidade do M-Hub2, comparado à versão original, é o design modular do software e sua habilidade de usar múltiplos protocolos WWAN e WPAN.

Palavras-chave: Internet das Coisas; Middleware; Objetos Móveis; Sensoriamento Móvel; Adaptação Dinâmica; Sensoriamento Remoto and Acionamento.

In charge of publications:
PUC-Rio Departamento de Informática - Publicações
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: publicar@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction **1**

2 Mobile Edge-based IoT Middleware and Related Work **3**

3 General Architecture **5**

4 M-Hub2: a remodel M-Hub architecture **6**

4.1 Core Library 6

4.1.1 **Connection Gateway** 8

4.1.2 **S2PA Gateway** 9

4.1.3 **MEPA Gateway** 10

4.2 MQTT, BLE, and Asper Libraries 10

4.3 EdgeSec service 10

4.4 Access to smartphone's local sensors 11

5 M-Hub2 Applications **11**

6 Tests and Performance Evaluation **12**

6.1 M-Hub2 Connection Time Evaluation 13

6.2 M-Hub2 Handover Evaluation 13

7 Conclusion and Future Work **14**

References **15**

1 Introduction

The volume - and diversity - of IoT End devices — and the growing trend to use many of them while “on the move”, whether they are smartphones or embedded sensors and actuators — has presented an annual growth rate of more than 25% [3]. Due to this growth, the connectivity infrastructure has evolved towards heterogeneous networks, where wireless communication technologies, coverage, configuration, communication and security parameters, data rate, and transmission latency, among other elements, are different.

To support this growth, and the heterogeneity of the devices and their context environment, the edge computing paradigm has emerged. As edge computing provides a pool of virtually operated computational and storage capabilities at the edge of the network, using the proximity to IoT devices to decrease the latency [1], once it can pre-store and pre-process the data, and then be in charge of forwarding it to the Internet through network technologies such as Wi-Fi or 3G/4G/5G.

Considering those personal mobile devices (smartphones and tablets) and mobile Internet are becoming increasingly ubiquitous, more affordable, and powerful, and that opportunistic and intermittent connectivity will become commonplace in a world filled with mobile, wearable, and embedded technology (but the data streams, rather than individual data samples or messages, will be of importance), such mobile personal devices become the natural candidates to be the propagator nodes (i.e. gateways to the Internet) for simpler IoT objects.

To support the development and operations of mobility-savvy IoT applications, such as in logistics, transportation, home automation, etc. we developed the ContextNet middleware [5], which consists of (i) a scalable communication infrastructure for backend services (a.k.a. ContextNet Core), and (ii) the Mobile-Hub, a mobile edge component that runs on Android OS devices and smartphones.

The idea of a such mobile edge is not new, as has already been introduced in [21] and [17]. As proposed by Talavera et al [17], the Mobile Hub (M-Hub) is a general middleware service responsible for discovering and opportunistically connecting a myriad of simple M-OBJs (e.g. IoT devices with sensors/actuators) accessible only through short-range WPAN technologies to the Internet.

The M-Hub plays the role of a communication hub between wireless PANs and Wi-Fi/4G connectivity, discovering nearby IoT peripherals through the corresponding WPAN-specific scan/advertisement protocols. But, contrary to other similar software for edge computing, the M-Hub is prepared to handle relative mobility and intermittent WPAN connectivity among itself and the M-OBJs, either because of itself or because the devices are moving. For this reason, in the remainder of this document, we will refer to them as *Mobile Objects (M-OBJs).*

Figure 1 shows how M-Hubs can be used as the intermediate edge elements for sensing (e.g. remote monitoring of driver behavior using a smartwatch) or affecting a remote mobile device (e.g. the hospital robot). In both cases, the smartphone running the M-Hub must be kept within the range of the WPAN (in the car, within the OBD2 device, or else in the hospital, in the smartphone of a technician responsible for the room).

The first version of the M-Hub - released in 2015 for the Android platform - evolved towards a sophisticated micro-service-based architecture with several optional extra services interacting through an EventBus.

The most prominent micro-services supported were:

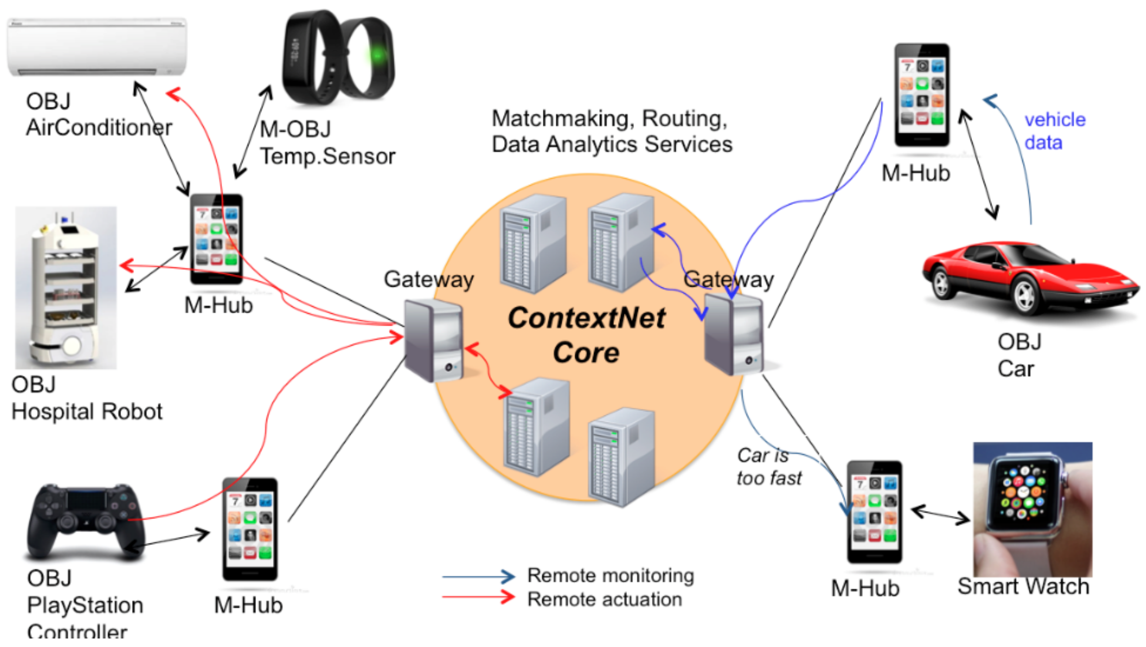

Figure 1: Possible use cases of The M-Hub for remote sensing and actuation.

- the Mobile Event Processing Agent (*M-EPA*) [15], a full-fledged Complex Event Processing engine;
- the Mobile Generic Actuation Service (*M-ACT*) [19, 18], a protocol agent that allows to trans-code high-level actuation commands (e.g. start, stop, dim, turnLeft, etc.) into equivalent byte-level sequences of commands that are specific for the type/product/model of the actionable M-OBJ;
- a Persistence Service using NoSQL for storing context data, intermediate sensor data, or use as a cache for data going in either direction.
- a Security Service built as an abstraction layer on top of WPAN protocols to create security mechanisms when communicating with M-OBJs and the ContextNet Core.

One big challenge of general-purpose edge middleware for opportunistic mobile sensing/actuation and stream processing is the large set of wireless network protocols and the heterogeneity of sensor/actuator types, models, and manufacturers. In addition, mobile edges typically roam through different places, therefore may encounter different WWAN and WPAN networks and IoT technologies along their trip.

Aiming to deal with these challenges, the M-Hub middleware has been re-designed from scratch. This paper presents the new architecture of this mobile edge software, named M-Hub2, in which the main principles of the original design have been preserved but flexibility, heterogeneity and ease of customization has been added. More specifically, the main contributions of this paper are:

- We propose and present a highly modular and configurable software architecture for the mobile edge.

- This architecture supports the use - and seamless adaptation - of alternative wireless WWAN and WPAN technologies, as well as different data flow processing engines. This is discussed - and tested - on the basis of a fully fledged implementation of the M-Hub2.
- We compare the Connection Time with peripheral M-OBJs, the Time to receive first (TTR) sensor data, as well as influence of the WPAN connection handover on TTR in versions of the M-Hub2 that use the MR-UDP and the MQTT, respectively.

In the following sections, we start with a general discussion of IoT middleware that employs mobile edge devices and related work in this field. In Sections 3 through 5 we describe the general architecture - and some implementation details - of the *M-Hub2*, its mandatory and its optional components. In Section 6 we present some applications using the M-Hub2. The results of some performance tests, their adaptability, and extensibility regarding multi-protocol communications are shown in Section 7. In Section 8 the conclusions and future works are discussed.

2 Mobile Edge-based IoT Middleware and Related Work

Massive smartphone usage enables potentially new pervasive and participatory IoT applications where smartphone users can act as universal hubs for sensing of, or actuation on, simpler M-OBJs (or mesh networks of them) that have only short-range wireless connectivity, such as ZigBee or Bluetooth LE. These M-OBJs, which may even be mobile (attached to vehicles, pets, or other goods) can take advantage of the eventual proximity of the mobile edge (M-Hub) to opportunistically use it as an intermediary to transmit data and/or receive command parameters to/from backend IoT services.

For instance, a mobile edge can retrieve various information (temperature, humidity, CO concentration, etc.) from sensor nodes around the user, both for making this information available to them, or for relaying it through the mobile Internet (3G/4G, or Wi-Fi) to a public environment.

Since such edge hubs are not stationary but move, the accessibility of the dispersed M-OBJs (sensors and actuators) is intermittent, temporary, opportunistic, and automatic (without user intervention). Therefore, the software running in them must support different types of connectivity protocols, and perhaps even switch seamlessly among them. Moreover, the middleware must be extensible to cope with varying drivers to access and interact with the highly heterogeneous plethora of M-OBJs.

In addition to environment context variability, mobile edge, such as smartphones, also tackle hardware limitations while providing effective computation. To handle those challenges, computation offloading techniques are popularly used [13]. The algorithms that implement them improve offloading tasks management [23], allocation of transmit power [7], resource allocation [9] and energy saving [6] [11].

Despite all the improvements in mobile edge computing, mobility awareness is still a significant problem, as most existing work assumes that users are stationary during task assignment and that communication between mobile devices and edge nodes are always available [10].

Pereira *et al* [12] advocate that mobile phones can act as IoT gateways given their current high computation and communication capabilities. The devices of their network architecture are BLE sensor nodes and standard mobile phones - acting as access points to the Internet - which can perform functions at the edge that complement processing in the cloud, such as data filtering, alarm management, and others, to enable distributed processing. The authors also mention the importance of the large set of sensors embedded in mobile phones for IoT because they enable monitoring and control automation applications in a wide range of domains such as healthcare and industry 4.0. They remark that the use of smartphones as gateways/hubs is necessary since several of the current IoT solutions cannot handle scenarios where many (or most) of the objects are mobile, such as in VANETs, swarms of drones, rovers, etc. Their approach aims at a holistic network architecture for the exchange and processing of sensor and actuator data with the Internet, making use of embedded RESTful web services while supporting true mobility. They use CoAP(Constrained Application Protocol) and data format EXI (Efficient XML Interchange) to achieve their goal.

Perera *et al* [2] propose a mobile application - Mobile Sensor Hub (MoSHub) - that allows a mobile phone to connect to a variety of different sensors and therefore collects, combines, processes, and sends sensor data to a server. A software architecture was developed to dynamically interconnect sensors to a mobile application (on smartphones) by generating a wrapper class called Sensor Device Definition (SDD), which describes the sensor's capabilities, and a virtual sensor, that hides implementation details and sensor data access.

The work of Zachariah *et al* [25], envisioned that any BLE device could leverage any smartphone as a temporary IP router and act as a normal IP end host. Hence, they propose a smartphone-centric approach where any phone could proxy a Bluetooth profile (services, characteristics, and attributes) to the cloud on behalf of a device. They also suggest a possible new role for the smartphone, as an opportunistic context provider for the nearby devices which can request certain services like location or current time, and explain that each IoT device must provide some meta information (content, type, destination, etc.) in its advertisement package that dictates how the phone should proxy the BLE profile data. In a recent review, Zachariah *et al* [24] despite the endorsement that mobile infrastructure helps alleviate the gateway problem remains, identifies smartphone operating systems as a limit to long-term and delay-intolerant connectivity due to performance optimizations. To handle those restrictions, the authors proposed a static gateway approach, implemented as a standalone device using ESP32 BLE/Wi-Fi SoC.

To facilitate the use of smartphones as mobile edges, several works use the Android platform, primarily due to its open and extensive nature. Zheng et al. [22] propose BraceForce, a middleware platform that incorporates event and model-driven concepts to provide efficient and simpler access abstractions to sensing devices in mobile applications. They require sensor devices to implement a standard Java interface (sensor driver) to be manipulated by the middleware. This API contains essential commands (e.g., open, close), configurations (e.g., start, restart), and communication primitives for sending and receiving key-value data. However, BraceForce does not address or provide operations to retrieve or restore a sensor state. Although BraceForce can discover sensor devices, it can only do this for known devices, i.e., using pre-developed sensor drivers. Thus, contrary to our approach, it is not capable to adapt (load/unloading sensor modules) for access to previously unknown sensor devices.

3 General Architecture

One of the huge challenges of IoT Edge middleware is to support the co-existence of multiple communication interfaces and protocols for local communications with nearby IoT devices on one side, and for WWAN communications with cloud-based services on the other side. The former requires compatibility with a big myriad of short-range low-power devices and the latter for communication with remote application services. Furthermore, it is also important to note that IoT edge devices are not just working as data sources but also as processing nodes to enable fog computing. In this aspect, several technologies should be easily interchangeable within the M-Hub.

Nowadays, it is possible to find several kinds of mobile/portable IoT devices behaving as M-OBJs, i.e., car keys, blood glucose meters, thermostats, multipurpose truck sensors, wearable devices, portable Ultrasonic Noise and Leak Detectors, a PLC Basic Controller for mobile robots, etc. that are deployable in a wide range of application areas, such as in smart spaces/buildings, industry and logistics, healthcare, agriculture, and maintenance. These devices usually communicate using several WPAN technologies such as ANT+, Zigbee, BLE, WiFi-Direct, etc.

Hence, this heterogeneity of connectivity types makes it very important M-Hub to be able to expand its WPAN support easily and future low-range, low-power connectivity technologies. While this heterogeneity of connectivity types is more notable for local communication of the M-Hub, it applies also to cloud-bound communication. Here we find several protocol options for use of as WWAN technology, such as CoAP, MQTT, MR-UDP, HTTP.

Our previous version of the M-Hub [15] was already designed and implemented as an application composed of several Android services and a manager, all executing in background and interacting with each other asynchronously through PubSub. The **S2PA Service** was responsible for discovering, managing WPAN connections, and interacting with all close-by M-OBJs in the M-Hub's WPAN range. This service kept a record of the current provided sensors/actuators (e.g., temperature, accelerometer, humidity) in each M-OBJ and internally published the sensed information to all other M-Hub components that subscribed for the data. One of them was the **Connection Service**, which was responsible for sending this data to the cloud in a JSON message or receiving (actuation or configuration) commands from the cloud through an Internet connection. Important messages - e.g., when a new connection/disconnection to an M-OBJ is detected - were sent immediately to the cloud. On the other hand, regular sensor data or low-relevance messages (e.g., temperature readings) were accumulated in a buffer and transmitted as a bulk message after some time intervals.

In any case, all messages addressed to the cloud-based "back-end" service were *enriched with context information*, which includes the timestamp of sensor data arrival, the current position of the M-Hub, and any other data related to the smartphone's embedded sensors (e.g., the level of background noise). The M-Hub's location was obtained through the **Location Service**, responsible for sampling the M-Hub's current position obtained from different providers like GPS, network, or manually entered (in case of a fixed location).

The periodicity and duration of these three services' activities were determined by the device's current three energy levels (LOW, MEDIUM, HIGH), which were set by the **Energy Manager**. These services would occasionally sample the device's battery level and check if their power supply came from the wall power outlet. All the services interacted through an **EventBus**[1], a Publish-Subscribe (PubSub) event bus optimized for Android. This asynchronous mode of interaction helped to decouple the M-Hub services and made it easy to add new components or remove existing services without the need to change any code.

Although the M-Hub was a pretty good initial achievement, it was yet a monolith application. This hindered other apps to interact and smoothly exchange information with it. In fact, in most cases, mobile apps would only be able to access the data collected (sent to the cloud by their co-located M-Hub) by having the app connect and retrieve data from the back-end services with the cloud.

Although the original plan was to support several WPAN technologies, this support for heterogeneity was never really implemented. The problem was that since everything was packed in the same module and all WPAN technologies ended up being part of the S2A service. So, it became very hard to include new technologies such as Classic Bluetooth and ANT+. In particular, incorporating a new protocol would require a deep knowledge of the specifics of the M-Hub, which makes any such extension quite a challenge.

New technologies were also not added as plug-and-play components, which made co-operation and separation of responsibilities difficult. And with time the codebase grew so much that compilation times eventually will get unsustainable.

4 M-Hub2: a remodel M-Hub architecture

To fix the aforementioned limitations, the M-Hub was redesigned as a multi-module library that can be included in any mobile application. Due to its modular design, one can select the components that will be part of a specific M-Hub, according to the requirements of the IoT application and the resource capability of the device used as a mobile edge (e.g. typically a smartphone).

The M-Hub's *core library* is the mandatory and basic part of the library which handles the overall M-Hub2 configurations and its main business rules (i.e. use cases). Other libraries – designed as *plugin libraries* - are used to extend the M-Hub's core with the suitable WPAN technology (e.g. BLE, Classic Bluetooth, etc), a WWAN technology (e.g. MQTT, MR-UDP), and even a complex event processing technology (e.g. Asper). The core thus interacts with multiple interfaces, either by receiving events as input (e.g. new sensor data) or else, by requesting some library interface to perform some action. Each plugin library, in turn, includes support for multiple communication technologies (e.g. BLE, ANT+, MQTT) and specific functionalities (e.g. communication with nearby M-Hubs).

Table 1 presents the architecture characteristics and features supported by the M-Hub and M-Hub2. The most important difference was made in the architecture, which is explained in detail in the subsections below. All the features supported by the previous version [20] were maintained on M-Hub2, which also includes three new features.

4.1 Core Library

The core library contains basic interfaces to facilitate its extension, so other developers don't need to know a lot of details on the M-Hub2 core library, but just to implement an interface in a separate module (extension library). This will also reduce compilation

[1]https://github.com/greenrobot/EventBus

Table 1: Architecture and supported features of M-Hub (MH1) and M-Hub2 MH2)

Characteristics	**MH1**	**MH2**
Architecture		
Monolith architecture	X	-
Service based	X	-
Multi-module library based	-	X
Plug-and-play components	-	X
Supported General Features		
Discovery, identification, connection and| monitoring of nearby sensors	| X	| X
Sensor Data Transcoding	*	X
Caching of probed sensor data	X	X
Configuring and Probing sensors	-	X
Pre-processing of sensor data	X	X
Sensor Data Transcoding	X	X
Dynamic adaptation to different WWAN and | WPAN networks	| -	| X
Dynamic adaptation to several Internet-based| session protocols (MR-UDP, MQTT, RestFul, etc.)	| -	| X
Access Authorization and Accounting	-	X

- : not supported   * : partially supported   X : supported

times since we keep everything as separate modules. Basic communication of the M-Hub2 components is managed by domain logic on specific use cases and all the information can be listened to on the UI by using a subscription on specific events. Thus, there is not an event bus for decoupled communication, instead, we rely on design patterns such as repositories to interact with the data.

The core library works as an entry point for building M-Hub2s (IoT Gateway) that fit multiple requirements. It provides a limited functionality as it only includes, by default, the MR-UDP protocol implementation for communication with the backend services. Any other necessary feature can be included or extended, if not available, with the specific plugin libraries, e.g. to allow the M-Hub2 to detect and connect to nearby BLE M-OBJs, or use a different WWAN protocol. Thus, it allows the M-Hub2 to work within mobile or static environments and simplifies collaborative development, since it is easy for other developers to include new plugins for specific use cases. With this in mind, a domain-based approach was used to define the main use cases of an IoT-Gateway and include them in the library.

Use Cases define the main logic and algorithms behind the M-Hub2, and they interact with multiple interfaces that are not necessarily implemented in the core library. Among those interfaces, WLAN is supposed to be implemented by any long-range communication technology (e.g. MQTT, AMQP, MRUDP). WPAN that handles short-range communication (e.g. BLE, Bluetooth, WiFi-Direct). And finally, CEP (Complex Event Processing) manages processing logic (e.g. Asper, siddhi). Implementations of such interfaces must be done as separate modules of the core library, and hence be included during

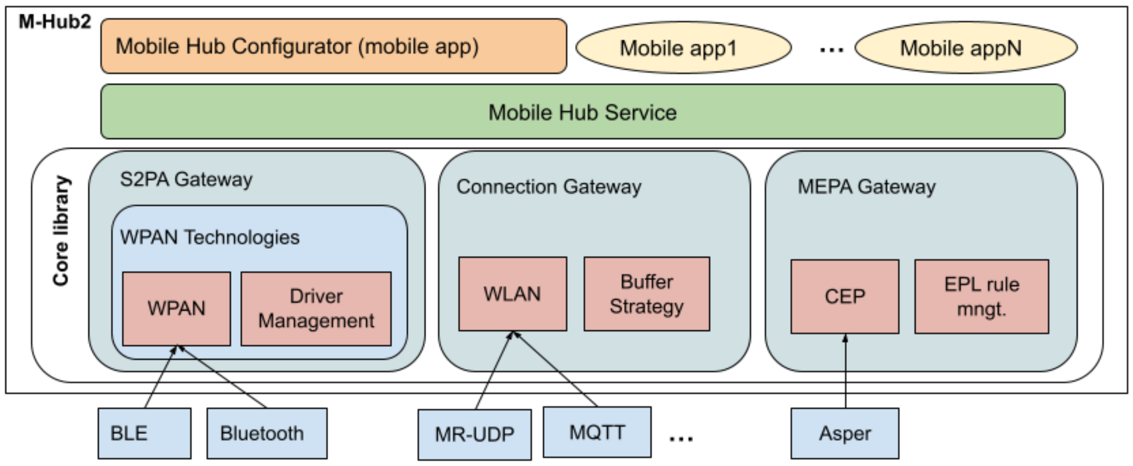

Figure 2: The M-Hub2 library architecture

the configuration of the M-Hub2.

Aiming to group the different use cases and technologies, some (enabler) components called gateways are the starting point of communication between the system and its use cases. The **Connection Gateway** manages the cloud communication technologies and acts as an entry point for incoming messages. The **S2PA Gateway** uses short-range communication technologies with M-OBJs to retrieve any sensed information. Finally, the **MEPA Gateway** processes all the data from multiple sources with the use of CEP. Each one of these gateways has multiple use cases that interact with the previously described interfaces, see Figure 2. As for storage, the repository pattern is used to separate the multiple sources for each entity (e.g. CEP Queries, Devices).

The core library must be configured using a builder object that defines the plugins that should be included for the execution. For example, if BLE is intended to be used, then its corresponding library must be added as part of the dependencies and set in the builder object. All the dependencies are managed by the Mobile Hub Configurator, which also starts a background service that contains all the gateways. Such service is responsible for keeping the M-Hub2 executing in background and starting/stopping the enabler components. A more detailed description of the gateways is provided in the following sections.

4.1.1 Connection Gateway

The connection gateway is the entry point for any incoming communication, hence it maps any message to a local controller. A controller is a component that has CRUD (Create, Read, Update, and Delete) methods to manage operations related to a specific entity such as processing rules (CEP queries), sensor data types, mobile objects, etc. Thus, it allows, for example, the addition of new processing rules, or the removal of previously defined ones.

The technology used for wide-range wireless communication (WLAN) is defined in the configuration of the core library. Today, it could be MR-UDP protocol [14], by default, or MQTT by including its library (i.e. WLAN-MQTT). To build a new WLAN technology library, the only requirement is to implement the *WLAN* interface, which is part of the core library.

Besides that, a component defines the buffering strategy for the messages towards the ContextNet Core. The reason is that not all outbound messages are equally relevant and urgent, and it is preferable not to continuously send data to save the hub's energy, by only turning on the WiFi or 4G interfaces occasionally. The **buffering strategy component** reads the network information to see if it is currently being used, if that is the case then there is no problem with sending the information. Otherwise, it will queue the information and only a worker object may be able to transmit all the queued messages. A Worker is an Android component that can execute periodic tasks. It is managed by the WorkerManager which has good energy consumption management, and thus it will start the worker every 15 minutes if necessary.

4.1.2 S2PA Gateway

This gateway has the only responsibility of the management of multiple M-OBJs to receive/send information from/to them. As we already mentioned before, the M-Hub2 must be able to interact with multiple WPAN technologies at the same time to support a wide range of devices. To seamlessly provide such behavior, the gateway maintains a unique flow with all of the underlying technologies, that starts with a constant discovery of M-OBJs. The scan use case will request to all the available WPAN technologies to discover M-OBJs, where the specific implementation of the scan resides on the corresponding plugin library.

Once an M-OBJ is discovered, the connection use case will attempt to init with the device, unless a driver is required (we will discuss this point later). If the connection is successful, then other use cases may enter into play, such as the subscription for sensor data, which requests the M-OBJ to provide any new sensor data whenever it is available. Or to request sensor data from a specific sensor, or execute an action in an actuator. It is also important to mention that all information, like the discovered M-OBJs and sensor data, is routed to other components for later processing (which will be detailed later on). It is also important to note that the core library is not aware of the underlying technologies, which makes the gateway interact only with a list of abstractions, and not with specific implementations.

There are also use cases that handle other sections of the process, such as the driver's loading. Being able to communicate under multiple WPAN technologies, is not the only requirement to support multiple M-OBJs. Each one of the M-OBJs types may need specific algorithms (**drivers**) to translate (transcode) the information that arrives from its sensors. These drivers are similar among devices of the same technology and require to be available at connection time. Thus, to make this possible it is important to separate the logic to load and store the driver, from the specific implementation for each WPAN technology.

The core library through the S2PA gateway may receive a driver error from a connection attempt to an M-OBJ. In that case, the core library takes care of loading the driver from either, the cloud or the local database and sends it to the appropriate WPAN technology. From that point, the technology has to take care of the driver transcoding to retry a connection with the M-Obj.

A transcoder is a software component that takes care of building the M-Obj driver from a string script that arrives from the core library (S2PA gateway). As of today, Android supports two scripting languages that can evaluate at runtime, Lua and javascript. As for our first transcoder implementation we chose Lua since it is very lightweight, simple, and easy to learn. Nevertheless, it is very simple to extend the WPAN-BLE library to include a javascript implementation of the transcoder.

The LuaTranscoder builds a driver, a BleDevice object, from the lua script received from the core. That BleDevice is stored in a Last-Recently-Used (LRU) cache. The size of the cache is 20 by default, but its size may be changed at configuration time. The idea is that in a static environment, drivers are not going to be changed at all, but in a mobile environment they are going to be changed most of the time.

4.1.3 MEPA Gateway

For local processing, we decided to use Complex Event Processing (CEP) to handle all the sensor data events, and thus generate complex events that are sent to the cloud with the WLAN technology. These complex events are very important, and thus they are not queued as happens with other data types. The idea behind the local processing is to reduce the transmission of events to the cloud, and hence only send data that the system considers relevant.

4.2 MQTT, BLE, and Asper Libraries

The MQTT library opens a dedicated channel of an MQTT broker running in a backend server and publishes data items to the cloud. The BLE library implements all the methods for discovering BLE peripheric devices, exchanging the GATT profile, and setting up BLE Characteristics reading and writings.

Asper is a library that contains the Asper-CEP engine [4] used by the MEPA Service to process CEP rules for filtering, aggregating, or checking event patterns over any stream of sensor data, either from a peripheral, BLE-connected IoT device, or from smartphone-local embedded sensors (e.g. the geographic location). From the viewpoint of Asper-CEP, all sensor data is defined as a primitive event type, which contains the sensor's names and their respective values and a timestamp.

4.3 EdgeSec service

EdgeSec is a security service that uses different cryptographic mechanisms to ensure the authenticity, authorization, confidentiality, and integrity of all data exchanged between an M-Hub2 and M-OBJs. It is based on a security architecture created to protect IoT middlewares that use mobile nodes as gateways, such as ContextNet [8]. The motivation for developing this solution was the growing risks and potential threats that IoT systems usually face, including:

- Man-in-the-middle attacks: because data exchanged between M-Hub2 and M-OBJ is not encrypted, malicious actors could eavesdrop on the communication channel and get access to confidential information. Additionally, they could alter data in transit to disrupt service or measurement;

- Device impersonation: any malicious or infected device could join the ContexNet network and impersonate a regular device, generating false data and negatively affecting the whole system.

- Denial of service attacks: it is common for malicious actors to infect and control a large number of IoT devices, creating a botnet network and using this distributed processing power to perform denial of service attacks against specific targets on the internet.

All these threats are mitigated by using the EdgeSec service. When an M-Hub2 connects to an M-OBJ that supports this service, a handshake and authentication process takes place to negotiate session keys and one-time passwords (OTPs).

To prevent eavesdropping, all data sent to and received from M-OBJs is encrypted using the session key and signed using OTPs through message authentication code (MAC) algorithms. This signature ensures that devices never use false identities to impersonate others and prevents data from being altered in transit. Denial of service attacks is also not possible because the connection to each device is authorized by an external authorization server that runs in the cloud. This server holds data related to each registered device and responds to authorization requests, sending the cryptographic elements needed to establish a secure connection.

EdgeSec service was built on top of other WPAN plugins, creating an abstraction layer where the cryptographic operations take place. This approach facilitates integration with M-Hub2 Core library components, leveraging the use of interfaces. EdgeSec runs as a plugin, implementing the interfaces required by the Core library to operate a WPAN data exchange, stamping the data with the additional security features just described before sending it to M-OBJs.

4.4 Access to smartphone's local sensors

The M-Hub2 also takes advantage of the large - and expanding - array of smartphone-embedded sensors to get more information about its current environment. More specifically, it supports activating and deactivating any smartphone sensor and collecting sensor/input data from them. The sampled data from these sensors (i.e., Context-Data) is collected in regular time intervals and stored in a *Context Record*, which has a key-value store and is updated constantly upon the arrival of new samples from active sensors. This *Context Record* is then attached to any message addressed to some server in the ContextNet Core in the cloud. Moreover, the *Context Record* may also be the input to the M-EPA/Asper library, where some event and data patterns are verified. For example, accelerometer and gyroscope data from the device may be fed to the M-EPA service to detect what kind of movement the smartphone (running M-Hub) is undergoing.

5 M-Hub2 Applications

For IoMT applications, the most relevant information about M-OBJs are their current location and state of movement (e.g., drones, intelligent tags at parcels, wearables, etc.) However, most of these objects are simple peripheral devices that lack GPS, are agnostic about their location and state of movement, and are not connected to the Internet. Thus, there is the problem of how remote clients can learn the whereabouts of their M-OBJs, track their movements, or else interact directly with them to sense or modify the environment. For this, smartphones - or equivalent gadgets with GPS and other sensors - running our M-Hub2 are a viable and economical solution for tracking and providing enriched context information about the M-OBJs (location, speed, surrounding sound, etc.) WPAN-connected to the M-Hub2.

Applications dealing with mobile objects may require more sophisticated information, e.g. if the objects of interest are being transported - or carried by - another trackable object, such as a vehicle, a conveyor belt, etc. In particular, *Co-movement* appears as a fundamental relation among objects in IoMT applications, since it may reveal information about the current state of transportation, of goods, machines, and people [16] and can be applied in logistics, personal and public security, m-health, tourism, among others.

We define a set of objects are in a mutual co-movement state at time t if, and only if, they have stayed within a given distance interval D from each other, and shared very similar speeds and accelerations during the time interval [t δ, t] (N most recent data probes).

With information about the kind of objects involved (say, X and Y), the knowledge of co-movement may reveal important information about them, such as X is being carried by Y, then X and Y are on the same path or X and Y should arrive at the same destination.

Using the scenario above as background, a Mobile Package Tracker(MPT) was developed as an application to support logistics operations with the use of BLE sensors and beacons attached to delivery packages and transport containers in logistics. The MPT captures context information through communication between the Smartphone and nearby M-OBJs, and co-related information among them to improve its short-range signal.

A system to track the co-movement of smart mobile objects is composed of two main components, a *Server Component*, running in a ContextNet Processing Node, and an M-Hub2 running on a smartphone. The M-Hub2 keeps on discovering new, and connecting to, Mobile Objects (e.g. parcels), and sending sensor data of the M-OBJ and context information (obtained by the smartphone's internal sensors) to the Server. Based on the collected sensor and context information, and the signal strength of the WPAN connection it is then possible to deduce the proximity between M-OBJ and the M-Hub2 smartphone, thus allowing to correlate their data so as to detect co-location and similar conditions from other M-OBJs in the surroundings. For example, consider an M-Hub2 in a truck that has a constant WPAN connection to an M-OBJ$_1$ with a temperature sensor, and to an M-OBJ$_2$ that is a smart tag on a parcel with a sensitive vaccine. Then this system could be used to track the whereabouts of the vaccine load and if it is constantly with the required temperature.

6 Tests and Performance Evaluation

In this section, we present the results of experiments where we measured the total time spent by the M-Hub2 for discovering, connecting, and receiving data from sensors through BLE. The ability of M-Hub2 to dynamically adapt to different WWAN and WPAN connectivity was also evaluated.

For all experiments we had the same setup: a smartphone Samsung Galaxy A51, running Android 12L Snow Cone was used for the M-Hub2; a SensorTag[2] was used as the M-OBJ, and their WPAN connection with the M-Hub2 is through BLE; for the MR-UDP and MQTT WWAN connectivity of the M-Hub2 with the Gateway we used Wi-Fi IEEE 802.11n. Finally, a PC intel(R) Core (TM) i5-10400F CPU 2.90GHz with 4GB of

[2]Texas Instruments CC2650 SensorTag - https://www.ti.com/lit/ml/swru410a/swru410a.pdf?ts=1678778276260

RAM running Ubuntu 20.04.5 emulated the ContextNet core Gateway and a back-end Processing Service. The latter was used for measuring the delay of the arrival of each sensor data packet in the cloud.

6.1 M-Hub2 Connection Time Evaluation

In this experiment, a scenario with one M-Hub2 and one sensor tag was used to evaluate the connection time considering different internet-based session protocols. The sensor used in this experiment was already paired with the mobile device before the connection attempt. The evaluation parameters considered were: Connection Time (CoT); Time to Receive the first M-OBJ data after the connection was established (TR) for both MQTT and MR-UDP protocols; and Total Time to Receive the first data (TTR), which corresponds to CoT + TR. All values were measured in seconds. Each experiment was repeated 10 times and calculated the mean value and the standard deviation. The results are shown in Table 2, for MQTT, and in Table 3, for MR-UDP.

It is worth noting that if the M-OBJ's driver is not in cache, the M-OBJ needs to be discovered by the S2PA Service before the data is sent to the Gateway, adding, on average, 1.814 seconds to TTR for both MQTT and MR-UDP, with 0.60426 of standard deviation.

Table 2: Performance of M-Hub connection with ContextNet Core for MQTT protocol

MQTT	**CoT (s)**	**TR (s)**	**TTR**
Mean value	1.185	4.694	5.879
Std. Deviation	0.09594	2.29174	

Table 3: Performance of M-Hub connection with ContextNet Core for MR-UDP protocol

MR-UDP	**CoT (s)**	**TR (s)**	**TTR**
Mean value	0.592	4.17	4.762
Std. Deviation	0.17623	0.60303	

These performance results indicate that MR-UDP is twice as fast as MQTT at establishing connectivity with the ContextNet Core. From previous experiments[14] we also know that MR-UDP is very robust to intermittent connectivity and sporadic communication failures, and thus probably much better than MQTT. This suggests that MQTT is probably a good choice when the WWAN connectivity is of good quality, while the M-Hub2 should better use MR-UDP in regions with weak or unstable Internet connectivity.

6.2 M-Hub2 Handover Evaluation

In this experiment, we considered a scenario where a sensor leaves a M-Hub2 network area and enters another M-Hub2 network area. For this, we used a second smartphone, already connected to the Gateway, to simulate the second M-Hub2 that received the M-OBJ's data after the first smartphone's Bluetooth is turned off. The smartphone used was a Samsung Galaxy A01 Core, running Android 10 Quince Tar, on the same Wi-Fi, for both MQTT and MR-UDP connections. The evaluation parameter considered was the Time to Receive the first data after handover (TR).

Each experiment was executed 10 times and we calculated the mean value and the standard deviation. The results are shown in Table 4, for MQTT, and in Table 5, for MR-UDP.

Table 4: Performance of M-Hub2 handover for MQTT protocol

MQTT	**TR (s)**
Mean value	5.272
Std. Deviation	0.42565

Table 5: Performance of M-Hub2 handover for MR-UDP protocol

MR-UDP	**TR (s)**
Mean value	5.541
Std. Deviation	0.27787

These results indicate the increase of TR due to handover. As it is more prominent in MR-UDP, the value of TR obtained in this protocol was greater than using MQTT.

7 Conclusion and Future Work

In this paper, we presented a second - and restructured - version of Mobile Hub (M-Hub2) implemented as multiple library-based architectures, preserving the main principles, services, and benefits from the original version. The modular architecture allowed more flexibility, efficiency, and configurability to the mobile middleware. As a result, it also provides co-existing multi-technology support with much less effort, extending the Mobile Hub Concept to have a native heterogeneity. With the newly listed features, especially the co-existing multi-technology support, M-Hub2 can detect and connect to nearby BLE M-OBJs, or use a different WWAN protocol.

Using two different implementations of WWAN protocols (MR-UDP and MQTT), and case tests alternating the connection protocols available, the M-Hub2 has been dynamically adapted to Internet-based session protocols in WWAN networks. Using M-Hub2 executing in different smartphone configurations and two BLE SensorTags, experiments were executed to measure the time interval for sensor discovery until the reception of the first sensor data.

In preliminary experiments, M-Hub2 exhibited excellent results for discovery and handover. The time to receive the sensor data, considering the discovery time, was below 5 seconds. These results represent about 50% better performance than the previous architecture [20] [17].

Following the milestones defined for the M-Hub2, the actual research is working to incorporate other WPAN protocols (ZigBee) and test them side-by-side with our BLE plugin.

Over the next year, we plan to make further tests and evaluations regarding, the M-Hub2's performance, Dynamic Adaptability, Extensibility, Energy consumption, and full support for further WWAN and WPAN heterogeneity.

Moreover, to better support intermittent connectivity and the fact that the M-Hub middleware service on the smartphone might be awakened only periodically (due to the mobile platform constraints), as future steps we will also consider other design and implementation options.

References

[1] Almutairi, J., and Aldossary, M. A novel approach for iot tasks offloading in edge-cloud environments. *Journal of Cloud Computing* (2021).

[2] C. Perera, P. Jayaraman, A. Z. P. C., and Georgakopoulos, D. “dynamic configuration of sensors using mobile sensor hub in internet of things paradigm. *IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing* (2013), 473–478.

[3] Chettri, L., and Bera, R. A comprehensive survey on internet of things (iot) toward 5g wireless systems. *IEEE Internet of Things Journal 7*, 1 (2020), 16–32.

[4] Eggum, M. *Smartphone Assisted Complex Event Processing.* dissertation, University of Oslo, 2014.

[5] Endler, M., and Silva, F. S. Past, Present and Future of the ContextNet IoMT Middleware. *OJIOT 4*, 1 (2018), 7–23.

[6] K. Zhang, Y. Mao, S. L. e. a. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access 4* (2016), 5896—-5907.

[7] L. Yang, H. Zhang, M. L. J. G., and Ji, H. Mobile edge computing empowered energy efficient task offloading in 5g. *IEEE Transactions on Vehicular Technology 67*, 7 (2018), 6398—-6409.

[8] M. Endler, A. S., and Cruz, R. An approach for secure edge computing in the internet of things. pp. 1–8.

[9] M. Yuyi, Z. J., and Letaief, K. B. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. In *Proceedings of the Wireless Communications and Networking Conference (WCNC)* (March 2017), IEEE, pp. 1–6.

[10] Maray, M., and Shuja, J. Computation offloading in mobile cloud computing and mobile edge computing: Survey, taxonomy, and open issues. *Mobile Information Systems 2022* (2022), 17.

[11] O. Munoz, A. P.-I., and Vidal, J. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Transactions on Vehicular Technology 64*, 10 (2015), 4738—-4755.

[12] Pereira, P. P., Eliasson, J., Kyusakov, R., Delsing, J., Raayatinezhad, A., and Johansson, M. Enabling cloud connectivity for mobile internet of things applications. In *2013 IEEE seventh international symposium on service-oriented system engineering* (2013), IEEE, pp. 518–526.

[13] Rahmani, A. M., Mohammadi, M., Mohammed, A. H., Karim, S. H. T., Majeed, M. K., Masdari, M., and Hosseinzadeh, M. Towards data and computation offloading in mobile cloud computing: taxonomy, overview, and future directions. *Wireless Personal Communications 119*, 1 (2021), 147–185.

[14] Silva, L. D. N., Endler, M., and Roriz Jr., M. MR-UDP: Yet Another Reliable UserDatagram Protocol, now for Mobile Nodes. Tech. rep., Departamento de Informática, PUC-Rio, 2013.

[15] Talavera, L. E., Endler, M., and Colcher, S. An Energy-aware IoT Gateway, with Continuous Processing of Sensor Data. In *Proc. of the Brazilian Symposiym for Computer Netwoks and Distributed Systems (SBRC 2016)* (May 2016).

[16] Talavera, L. E., Endler, M., and Silva, F. S. E. Demo abstract: Monitoring co-movement of smart objects using accelerometer data. *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)* (2015), 214–216.

[17] Talavera, L. E., Endler, M., Vasconcelos, I., Vasconcelos, R., Cunha, M., and da Silva e Silva, F. J. The Mobile Hub concept: Enabling applications for the Internet of Mobile Things. In *PerCom Workshops* (Mar. 2015), pp. 123–128.

[18] Valim, S., Nogueira, F., Pisani, F., and Endler, M. Middleware Support for Generic and Flexible Actuation in the Internet of Mobile Things. In *IEEE World Forum on Internet of Things (WF-IoT 2020* (2020).

[19] Valim, S., Zeitune, M., Olivieri, B., and Endler, M. Middleware support for generic actuation in the internet of mobile things. *Open Journal of Internet Of Things (OJIOT) 4*, 1 (2018), 24–34.

[20] Vasconcelos, R., Talavera, L., Vasconcelos, I., Roriz, M., Endler, M., Gomes, B., and Silva, F. An adaptive middleware for opportunistic mobile sensing.

[21] Wu, X., Brown, K. N., and Sreenan, C. J. Snip: A sensor node-initiated probing mechanism for opportunistic data collection in sparse wireless sensor networks. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2011), pp. 726–731.

[22] X. Zheng, D. E. P., and Julien, C. “braceforce: A middleware to enable sensing integration in mobile applications for novice programmers. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems* (2014), pp. 8–17.

[23] Z. Wang, Z. Zhao, G. M. X. H. Q. N., and Wang, R. User mobility aware task assignment for mobile edge computing. *Future Generation Computer Systems 85* (2018), 1–8.

[24] Zachariah, T., Jackson, N., and Dutta, P. The internet of things still has a gateway problem. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications* (New York, NY, USA, 2022), HotMobile '22, Association for Computing Machinery, p. 109–115.

[25] Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., and Dutta, P. The internet of things has a gateway problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (New York, NY, USA, 2015), HotMobile '15, Association for Computing Machinery, p. 27–32.