Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Menezes Cavalcante Lemos dos Passos

Uma implementação do MobileHub em Python compatível com Raspberry Pi

PROJETO FINAL DE GRADUAÇÃO

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Ciência da Computação

Rio de Janeiro
Dezembro de 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Menezes Cavalcante Lemos dos Passos

Uma implementação do MobileHub em Python compatível com Raspberry Pi

Projeto final apresentado ao Curso Bacharelado em Ciência da Computação como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Prof. Markus Endler
Orientador
Departamento de Informática – PUC-Rio

Rio de Janeiro, 10 de Dezembro de 2024

Resumo

Passos, D. M. C. L.; Endler, Markus. **Uma implementação do MobileHub em Python compatível com Raspberry Pi**. Rio de Janeiro, 2024. 27p. Relatório Final de Projeto Final II – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A solução proposta é o desenvolvimento de uma nova versão do sistema Mobile Hub na linguagem Python que seja compatível com dispositivos Raspberry Pi. Isso busca ampliar a utilização de toda a estrutura da ContextNet para projetos com utilizam essas tecnologias, que foram escolhidas justamente pela ampla utilização em projetos de IoT. O método utilizado consiste na análise do funcionamento e do código-fonte do já existente Mobile Hub, que foi desenvolvido com a linguagem Kotlin para a plataforma Android, e transpor para Python juntamente com o desenvolvimento das adaptações necessárias para sua execução no ambiente Linux padrão do Raspberry Pi. O trabalho inclui a discussão das diferenças entre os ambientes e a criação dos primeiros módulos do M-Hub Python além de descrever os passos necessários para que trabalhos futuros possam evoluir o que foi desenvolvido para atingir as mesmas capacidades do M-Hub Kotlin.

Palavras-chave

Computação Pervasiva; Internet das Coisas Móveis; Engenharia de Software.

Sumário

Lista de figuras

1
Introdução

Conforme ocorrem avanços na área da Internet das Coisas (IoT), a tecnologia digital é progressivamente integrada ao cotidiano do ser humano. Uma das formas como isso acontece é com a utilização de variados tipos de dispositivos espalhados pelos espaços físicos capazes de coletar e transmitir informações, tomar decisões e atuar em ambientes. Tal abordagem traz consigo diversos desafios que destacam a importância da evolução das técnicas de desenvolvimento da computação pervasiva.

O cenário se mostra mais complexo quando os dispositivos são capazes de se locomover no espaço, seja por conta própria ou acoplados a outro objeto móvel. À área de estudo que lida com essa dinâmica foi dado o nome de Internet das Coisas Móveis (IoMT). No cotidiano, por exemplo, milhões de pessoas utilizam diariamente serviços envolvendo telefones móveis, veículos e construções inteligentes munidos de múltiplos sensores e capacidade de comunicação. Com a mobilidade surgem muitos novos desafios como falhas imprevisíveis, autonomia energética, interferências, conectividade entre dispositivos e privacidade dos usuários (NAHRSTEDT et al., 2016).

Os custos necessários para lidar com todos esses desafios incentivam a criação de formas de simplificar o desenvolvimento de sistemas com dispositivos móveis inteligentes. Uma solução possível é a criação de uma plataforma middleware responsável por oferecer funcionalidades genéricas de utilização dos dispositivos e capaz de ser utilizada em diversas aplicações. Um middleware deve possuir uma arquitetura que permita reunir múltiplos mecanismos necessários para a interatividade entre todos os elementos de um sistema IoMT como, por exemplo, detecção e leitura de dados de objetos próximos, compartilhamento de informações entre nós, troca de dados e comandos entre nós móveis e servidores por meio do acesso à internet e obtenção de dados de contexto dos dispositivos.

O objetivo deste trabalho é o desenvolvimento de mecanismos (a) de detecção e comunicação entre um dispositivo móvel e sensores utilizando Bluetooth Low Energy (BLE) como tecnologia de rede de curto alcance (WPAN), (b) de leitura de dados de contexto do dispositivo e (c) de envio de mensagens com servidores via MQTT. A solução foi desenvolvida em Python e tem como objetivo ser uma contribuição com o esforço coletivo para criação de uma versão adicional do já existente Mobile Hub (M-Hub) da ContextNet (ENDLER; SILVA, 2018), um middleware IoMT desenvolvido pelo Laboratory

for Advanced Collaboration (LAC) [1] da PUC-Rio e que conta apenas com uma versão em Kotlin para dispositivos Android. Esses mecanismos disponibilizam funcionalidades básicas para os dispositivos que executam o M-Hub além de permitir novas contribuições que estendam essas ou novas funcionalidades em trabalhos futuros. Para isso, a implementação se baseou no funcionamento do atual M-Hub, seguindo seus padrões de arquitetura e de decisão de projeto com os devidos ajustes para a adequação às diferenças de linguagem de programação, de sistema operacional e de dispositivo.

[1] Laboratory of Advanced Collaborations - http://www.lac.inf.puc-rio.br/ Acesso em Maio/2023

2
Situação Atual

2.1
A ContextNet

A ContextNet é um middleware IoMT que possui uma arquitetura dividida em 3 principais partes: o ContextNet Core, o MobileHub e os Mobile Objects (M-OBJ). O ContextNet Core consiste em um grupo de um ou mais servidores que tratam os dados recebidos pelos M-Hubs e direcionam comandos a eles. Um M-Hub consiste em um dispositivo com acesso à internet e, potencialmente, a alguma tecnologia WPAN. M-OBJs são quaisquer dispositivos que possuam pelo menos uma tecnologia WPAN e, usualmente, dotados de sensores para leitura de dados, atuadores para execução de ações no ambiente e com menor poder de processamento. A ContextNet tem como um dos seus principais objetivos permitir que M-Hubs busquem e se comuniquem com M-OBJs espalhados pelo ambiente e transfiram esses dados para os servidores ContextNet Core, onde poderão ser armazenados e processados de acordo com as necessidades da aplicação. Os M-Hubs também podem adicionar dados de contextos às cargas transmitidas como, por exemplo, coordenadas geográficas obtidas pelo GPS do próprio aparelho que executa o M-Hub e o seu nível de bateria. Assim é possível que os dados enviados para o ContextNet Core contenham informações adicionais de contexto além daquelas que os M-OBJs podem gerar (ENDLER; SILVA, 2018).

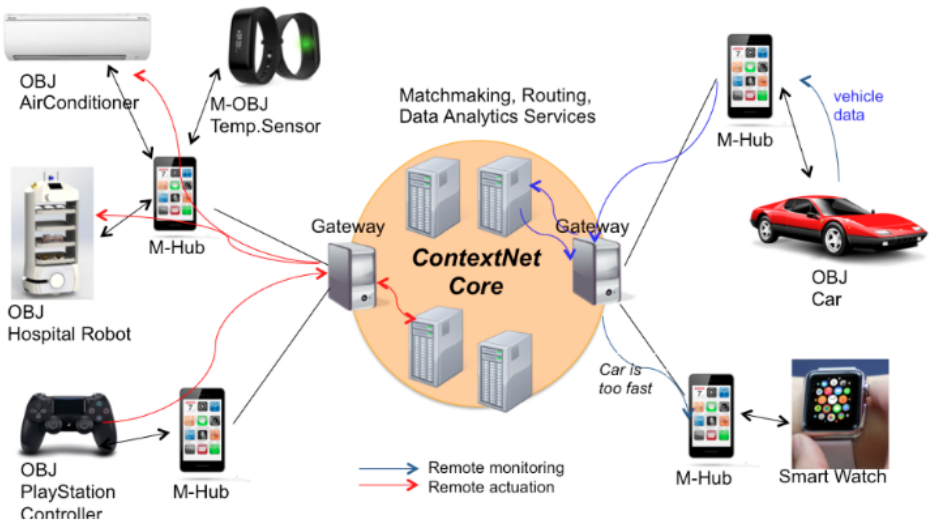

Figura 2.1: Arquitetura da ContextNet.
Fonte: (TALAVERA et al., 2023)

2.2 O Mobile Hub

O Mobile Hub é o principal componente móvel da ContextNet, que serve como intermediário entre os M-OBJs e os servidores do ContextNet Core. Sua arquitetura consiste em 3 serviços: (1) o Short-range Sensor, Presence and Actuation (S2PA), responsável pela descoberta e comunicação do M-Hub com os M-OBJs por meio de alguma tecnologia WPAN (como Bluetooth Classic, Bluetooth Low Energy, Wi-Fi Direct, Zigbee, entre outras) e dos drivers de cada M-OBJ; o (2) Connection gateway, responsável por se conectar com o ContextNet Core para receber e publicar mensagens por meio de uma tecnologia WLAN como o MQTT e (3) Complex Event Processing Gateway (CEP), responsável por interpretar e processar cadeias de eventos de acordo com regras predefinidas.

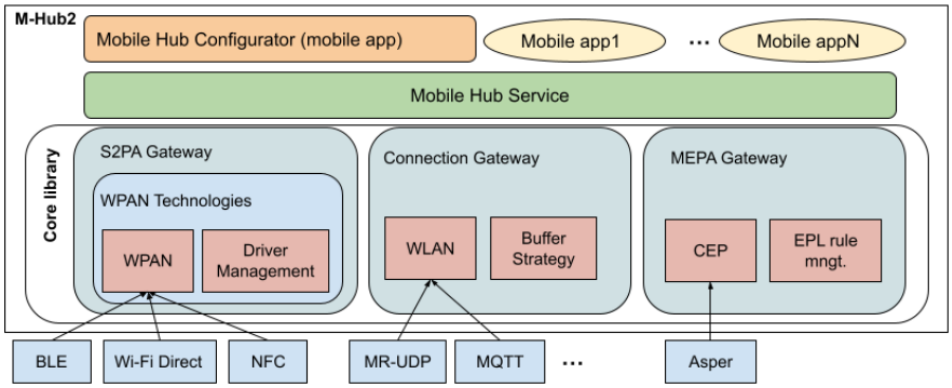

Figura 2.2: Arquitetura do Mobile Hub.
Fonte: (TALAVERA et al., 2023)

Enquanto se locomove por um espaço físico, o M-Hub pode buscar por dispositivos de sua vizinhança utilizando as tecnologias WPAN incorporadas ao sistema. Ao encontrar um novo dispositivo, se verifica se existe um driver correspondente instalado ou disponível para download e, caso exista, inicia uma conexão com o dispositivo e utiliza as funcionalidades do seu driver para se inscrever na leitura dos seus dados. Cada uma dessas etapas gera um evento que é notificado por um barramento do sistema que pode ser observado pela aplicação. A aplicação, por sua vez, pode tomar decisões e executar as funcionalidades do M-Hub.

Para permitir compatibilidade com variadas tecnologias WPAN, tecnologias WLAN, M-OBJs e tipos de informações de contexto, o M-HUB foi projetado com uma arquitetura extensível. Dessa forma desenvolvedores de aplicações podem, harmoniosamente, integrar tecnologias originalmente não disponibilizadas ou, até mesmo, proprietárias simplesmente estendendo as devidas interfaces. O serviço responsável por essas características é o S2PA, que

desempenha um papel crucial na comunicação e coordenação entre dispositivos heterogêneos, ao mesmo tempo em que fornece uma interface unificada para os desenvolvedores. Isso favorece o controle de dados de forma transparente, eficiente e escalável.

O Connection Gateway é responsável pela comunicação com os servidores da ContextNet. Atualmente suporta dois protocolos: MR-UDP e MQTT. O serviço permite se conectar, receber, escrever e responder mensagens dos tópicos mqtt. A aplicação pode fazer uso dessas funcionalidades, assim como outros módulos podem utilizá-las para notificar os servidores de informações geradas durante a execução.

O CEP permite analisar os eventos gerados pelos outros módulos do M-Hub e, com base em regras predefinidas, avaliar se a ordem dos eventos acontecidos permite concluir um evento mais complexo.

Para um sistema como esse, que visa ser efetivamente utilizado em inúmeros projetos que impactam diretamente a sociedade, é importante que sua utilização seja adaptável o suficiente para atender os mais variados cenários e dispositivos que abarcam a interconectividade do mundo atual. Compatibilidade com as tecnologias mais amplamente empregadas juntamente com a possibilidade de integrar soluções customizadas ao seu ecossistema além da capacidade de coletar, processar e armazenar dados são características essenciais que um Middleware IoMT deve considerar.

2.3 O Mobile Hub em outras plataformas

A modelagem da ContextNet não pressupõe muitas características acerca dos dispositivos que executam o M-Hub. Portanto, em uma aplicação, é possível imaginar uma variedade de aparelhos móveis capazes de assumir a função de M-Hubs como laptops, computadores Raspberry Pis, carros e drones. Para isso, novas implementações do Mobile Hub precisam ser criadas para dar suporte aos sistemas operacionais e hardwares desses dispositivos, já que as implementações atualmente existentes utilizam as APIs do sistema Android, impossibilitando a utilização imediata em outros sistemas.

3
Objetivos do trabalho

A solução proposta é o desenvolvimento de uma nova versão do sistema Mobile Hub em Python 3 para o sistema operacional Linux. Isso permitirá a utilização de toda a estrutura da ContextNet em projetos com essas tecnologias, que foram escolhidas justamente pela ampla utilização em projetos de IoT com destaque para os dispositivos Raspberry Pi. Essa versão visa manter a mesma arquitetura e, futuramente, as mesmas capacidades que o M-Hub para Android possui atualmente, garantindo compatibilidade total com a ContextNet. Como o M-Hub é um sistema complexo com muitos módulos, foi escolhido um conjunto de módulos e funcionalidades compatíveis com o tempo designado para este trabalho.

Os objetivos são (1) implementar o módulo S2PA, responsável por gerenciar as tecnologias WPAN e agregar os dados dos M-OBJs conhecidos garantindo a operabilidade da comunicação com múltiplos dispositvos; (2) Desenvolver um módulo WPAN de BLE funcional e integrável ao S2PA; (3) Garantir correto funcionamento do que foi desenvolvido com o Raspberry Pi; (4) Criar um driver para capaz de ler, interpretar e escrever características de um Sensor Tag CC2650 via BLE e (4) desenvolver um modelo de execução compatível com os mecanismos da linguagem Python.

4
Atividades Realizadas

4.1
Estudos conceituais e de tecnologia

O desenvolvimento deste trabalho exigiu o aprofundamento no estudo de diversas tecnologias, cuja compreensão foi fundamental para o planejamento e execução dos objetivos propostos. Os principais tópicos estudados foram as linguagens de programação Kotlin e Python, a biblioteca que implementa o padrão de programação reativa ReactiveX, o funcionamento da tecnologias BLE.

Inicialmente foi necessário o estudo da linguagem Kotlin e das APIs de desenvolvimento do Android para o correto entendimento do funcionamento geral do M-Hub e da ContexNet baseado na análise do código-fonte e da execução do exemplo de aplicação do M-Hub. Essa etapa foi fundamental pois permitiu detectar todas as tecnologias utilizadas e planejar o que deveria ser estudado em seguida.

Com a análise do código-fonte, foi possível perceber que o M-Hub possui uma arquitetura orientada ao paradigma de programação reativa utilizando, extensivamente, a biblioteca ReactiveX[1] para o gerenciamento de execução das tarefas assíncronas. Essa biblioteca é implementada em diversas linguagens de programação, incluindo Python e Java. A versão de Java é utilizada na implementação do M-Hub para Android, já que Kotlin garante interoperabilidade com bibliotecas de Java.

A ReactiveX permite simplificar o consumo de uma sequência de eventos assíncronos por vários consumidores concorrentemente. Para isso, os consumidores se inscrevem no fluxo de eventos ao fornecer callbacks ou ao fornecer um objeto Observer com métodos que deverão ser invocados a cada novo dado, evento ou erro gerado e também uma callback para quando o fluxo for encerrado. A Figura 4.1 mostra o diagrama de classes simplificado de uma sequência observável.

O método subscribe pode, por sua, vez agendar rotinas assíncronas que realizarão a chamada aos métodos dos Observers assim que cada elemento do fluxo de dados estiver disponível. Esses dados podem ser, por exemplo, os M-OBJs encontrados durante o escaneamento de dispositivos WPAN. Assim, a cada novo dispositivo encontrado o método onNext(M-OBJ) de cada um dos

[1] https://reactivex.io/

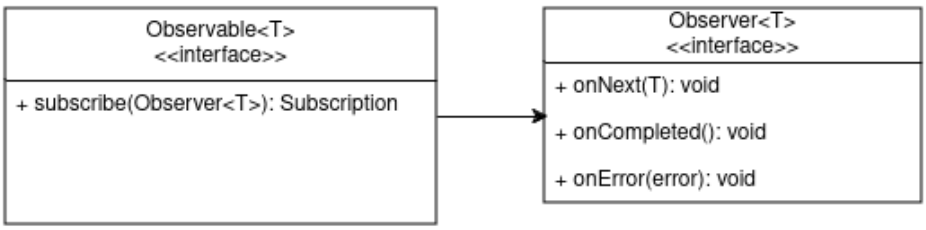

Figura 4.1: Diagrama de classes de uma sequência observável no ReactiveX

Observers será chamado, enviando os dados do dispositivo para a aplicação cliente.

Para a compreensão do serviço S2PA foi necessário o estudo da tecnologia BLE. Para isso foi utilizado principalmente o livro (HEYDON, 2012) e o guia (Bluetooth SIG, 2022). O estudo foi importante principalmente para aprender como funciona a interface do Perfil de Atributos Genéricos (GATT) oferecida por dispositivos periféricos para a leitura e escrita de dados via BLE.

Em uma conexão entre dois dispositivos BLE, existe a distinção entre o dispositivo Master (cliente GATT) e o dispositivo Slave (servidor GATT). Essa distinção é importante, pois define as capacidades de cada um deles na comunicação. Em uma máquina de estado BLE, Masters são dispositivos mais complexos e podem manter conexões com múltiplos dispositivos Slaves, enquanto Slaves devem manter sua conexão com um e apenas um Master e geralmente são dispositivos com menor poder de processamento. Também é determinado que um dispositivo não pode ser, simultaneamente, Master e Slave. Portanto é natural que o M-Hub se comporte como Master nas conexões com M-OBJs, já que, comumente, será o dispositivo com maior poder de processamento e com capacidade de se conectar com múltiplos pares.

O principal elemento conceitual em uma comunicação BLE é o atributo, que representa uma informação armazenada no servidor GATT que possui um tipo nomeado, um endereço e um valor. Os atributos são armazenados no banco de dados do servidor e são organizados hierarquicamente entre serviços e características, de forma que cada serviço agrupa um conjunto de características que podem ser acessadas. A coleção de todos os atributos de um servidor GATT é considerada seu banco de dados e , como representado na Figura 4.2. Após iniciar uma conexão com o servidor, um cliente GATT pode enviar requisições solicitando a leitura ou escrita dos seus atributos. Dessa forma, o M-Hub consulta dados e envia comandos aos M-OBJs conectados.

Attribute Handle	Attribute Type	Attribute Value
0x0001	Primary Service	GAP Service
0x0002	Characteristic	Device Name
0x0003	Device Name	"Proximity Tag"
0x0004	Characteristic	Appearance
0x0005	Appearance	Tag
0x0006	Primary Service	GATT Service
0x0007	Primary Service	Tx Power Service
0x0008	Characteristic	Tx Power
0x0009	Tx Power	−4dBm
0x000A	Primary Service	Immediate Alert Service
0x000B	Characteristic	Alert Level
0x000C	Alert Level	
0x000D	Primary Service	Link Loss Service
0x000E	Characteristic	Alert Level
0x000F	Alert Level	"high"
0x0010	Primary Service	Battery Service
0x0011	Characteristic	Battery Level
0x0012	Battery Level	75%
0x0013	Characteristic Presentation Format	uint8, 0, percent
0x0014	Characteristic	Battery Level State
0x0015	Battery Level State	75%, discharging
0x0016	Client Characteristic Configuration	0x0001

Figura 4.2: Banco de dados de atributos de um servidor GATT.
Fonte: (HEYDON, 2012).

4.2 Testes e Protótipos para aprendizado e demonstração

Para a primeira observação do funcionamento do M-Hub, foi feito um teste executando o exemplo de aplicação que é distribuído juntamente com a biblioteca do M-Hub. Para isso foi necessária a instalação do ambiente de desenvolvimento com o Android Studio[2], um servidor MQTT, um aparelho smartphone Android e um SensorTag CC2650[3]. Com isso, foi feita a instalação da aplicação de exemplo no aparelho Android e configurado as permissões das tecnologias wireless e de servidor MQTT. Os resultados foram observados simultaneamente pela tela do aparelho Android e pelo log do servidor MQTT enquanto a aplicação executava. Nesse teste foi possível notar os dispositivos encontrados pelo M-Hub, a detecção do SensorTag como M-OBJ devido a presença do driver do SensorTag na aplicação, a leitura dos dados dos sensores e o envio dos dados para o servidor MQTT.

Para aprender sobre a utilização da biblioteca de BLE para Python que foi escolhida para o projeto, foi criado um protótipo com o objetivo de ler as características de um dispositivo. Para isso foram utilizados um

[2]https://developer.android.com/studio. Acesso em 31/10/2024.
[3]https://www.ti.com/tool/TIDC-CC2650STK-SENSORTAG. Acesso em 31/10/2024.

ESP32-WROOM-32E[4] e um SensorTag CC2650. O ESP32 foi programado utilizando o framework ESP-IDF[5] para fornecer textos em formato de string como características GATT. Com um script em Python foi possível encontrar os dois dispositivos, ler os textos do ESP32 e ler os sensores do SensorTag.

[4]https://www.espressif.com/en/products/socs/esp32. Acesso em 31/10/2024.

[5]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html. Acesso em 31/10/2024.

5
Projeto e Especificação do Sistema

Apesar dos esforços para replicar fielmente a arquitetura original do M-Hub na implementação em Python, nem todos os aspectos puderam ser mantidos. Isso se deve tanto a diferenças fundamentais entre as linguagens quanto às eventuais necessidades de simplificação para priorizar o desenvolvimento de uma versão inicial funcional em tempo hábil. Portanto essas diferenças serão explicitadas e justificadas em cada caso para destacar as decisões tomadas e identificar pontos de evolução para trabalhos futuros.

O M-Hub se trata de um sistema reativo, portanto o gerenciamento de tarefas concorrentes é de fundamental importância para o seu funcionamento. Essa condição exige um modelo de execução cuidadosamente planejado para garantir a convivência harmoniosa entre os diversos módulos do sistema e da aplicação levando em consideração as características do sistema operacional e do hardware utilizado. Por esse motivo foi necessário analisar o modelo de execução do M-Hub para Android e planejar um modelo que atenda às mesmas necessidades em Python para sistemas baseados em Linux.

A solução escolhida foi a utilização da biblioteca asyncio[1], que é uma biblioteca padrão do Python com ampla compatibilidade com o escossistema do Python. A asyncio permite a criação e execução de laços de rotinas assíncronas e concorrentes juntamente com um conjunto de funcionalidades para gerenciamento das tarefas como mecanismos de sincronização e filas de tarefas. A possibilidade de conciliar a asyncio com o ReactiveX foi fator crucial para essa escolha[2] [3].

Com o asyncio, o modelo de execução do M-Hub Python se baseia cria uma fila de tarefas e adiciona à fila as rotinas assíncronas dos módulos Gateways que são, então, executadas pelo escalonador do asyncio. As demais tarefas são adicionadas pelos próprios módulos à fila para a execução de suas respectivas rotinas necessárias. Como um exemplo disso, o módulo WPAN é capaz de criar uma tarefa que inicia uma busca por dispositivos BLE e, por meio do operador 'await', pausar a execução da tarefa até que a notificação de um dispositivo encontrado seja recebida, o que permite que outras tarefas do sistema executem durante essa pausa.

[1] https://docs.python.org/3/library/asyncio.html. Acesso em 31/10/2024.

[2] https://github.com/ReactiveX/RxPY/tree/master/examples/asyncio. Acesso em 31/10/2024.

[3] https://oakbits.com/rxpy-and-asyncio.html. Acesso em 31/10/2024.

Outra recurso importante utilizado pelo M-Hub para Android é o framework de injeção de dependências Dagger[4] [5]. Esse framework facilita o desenvolvimento de módulos com baixo acoplamento, já que permite que as classes de um módulo, que implementam as interfaces das quais outro módulo depende, sejam instanciadas e providas automaticamente pelo framework. Para isso, basta que seja definido um mapeamento de quais classes devem ser fornecidas para cada interface, além de também ser permitido alterar dinamicamente o mapeamento. Assim, os módulos podem depender apenas das interfaces necessárias ao invés de depender diretamente dos módulos que implementam tais interfaces. Isso traz diversos benefícios para o sistema, como, por exemplo, a compilação independente de cada módulo, a possibilidade de modificar qual implementação será fornecida para uma interface em tempo de execução e possibilidade de múltiplas implementações diferentes para uma mesma interface.

Python, como uma linguagem de script interpretada, impossibilita a existência de uma biblioteca nos mesmos moldes da Dapper, que depende fundamentalmente do modelo de tipagem estática. Entretanto é possível utilizar a dinamicidade do Python para implementar padrões de injeção automática de dependências em tempo de execução sem a necessidade de incluir bibliotecas externas. Seguindo essa ideia foi incluída no M-Hub Python um módulo de injeçao de dependências nomeado "di". Esse módulo define a classe "Injection"responsável por gerenciar a instanciação das dependências. Para isso foi definido que as classes podem ser mapeadas a strings. Ao se invocar o método "inject(name)", a classe associada ao nome "name"será instanciada, instanciando também as classes associadas aos nomes de cada um dos parâmetros do seu construtor recursivamente.

Código 1: Classe Injection

```
1  from inspect import isclass, signature
2  from typing import Any, Dict
3
4
5  class Injection:
6      _class_binds : Dict[str, type] = {}
7      _instances : Dict[str, Any] = {}
8
9      @staticmethod
10     def bind(bindings:Dict[str, Any]):
11         for name, obj in bindings.items():
12             if isclass(obj):
13                 Injection._class_binds[name] = obj
```

[4]https://dagger.dev/. Acesso em 31/10/2024.

[5]https://developer.android.com/training/dependency-injection/dagger-android. Acesso em 31/10/2024.

```
14              else:
15                  Injection._instances[name] = obj
16
17      @staticmethod
18      def inject(name:str):
19          if name in Injection._instances:
20              return Injection._instances[name]
21          obj = Injection._class_binds[name]
22          injected_params = [Injection.inject(param) for param in
                signature(obj).parameters]
23          Injection._instances[name] = obj(*injected_params)
24          return Injection._instances[name]
```

A implementação desse forma instancia as classes e as armazena para reutilização, caracterizando o comportamento de uma classe singleton. Consequentemente todas as injeções de uma dependência de um mesmo nome compartilhará a mesma instância da classe associada ao nome. Futuramente essa classe poderá ser expandida para admitir outros comportamentos como uma nova instância a cada injeção.

O S2PA é o módulo do M-Hub responsável por gerenciar os M-OBJs e as tecnologias WPAN além de fornecer ao resto do sistema um fluxo único de eventos contendo dados sobre a descoberta e a comunicação dos M-OBJs. A sua arquitetura é projetada para que variadas tecnologias WPAN possam ser integradas sem que haja interferência na forma em que outros módulos do M-Hub e a aplicação consomem os dados gerados. O S2PA é completamente agnóstico da forma que as tecnologias WPAN operam, o funcionamento é garantido desde que elas implementem a interface WPAN esperada. A aplicação cliente pode estender a classe abstrata WPAN e adicioná-la na inicialização do M-Hub para serem administradas automaticamente pelo S2PA.

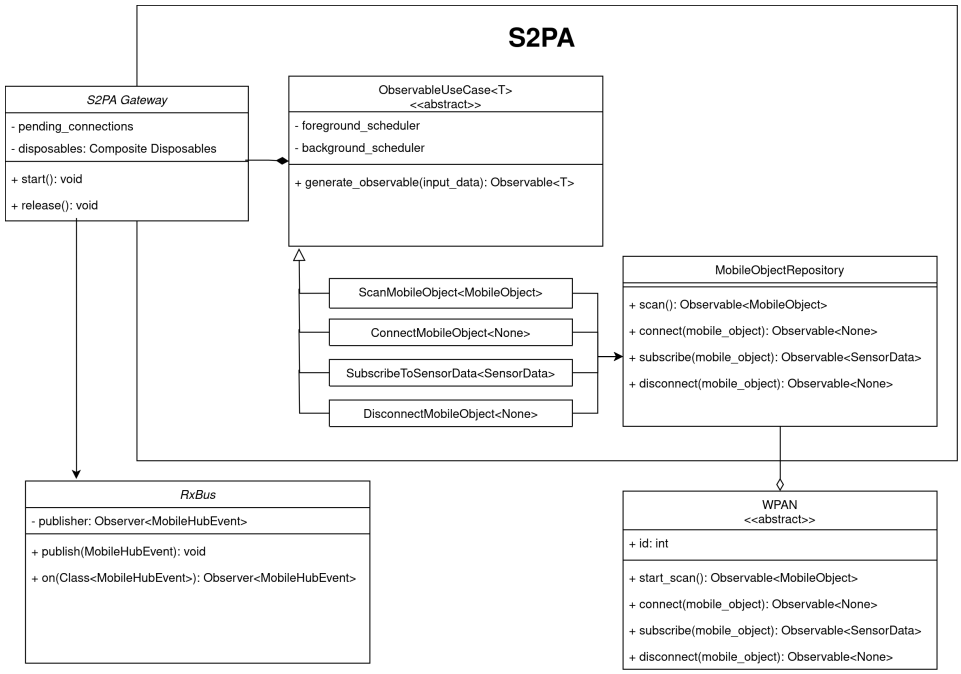

Figura 5.1: Arquitetura de classes do S2PA.

O M-Hub se encarrega de iniciar e encerrar o S2PA durante o início e o término de uma execução, mantendo o funcionamento do S2PA de forma continua durante toda a execução do M-Hub no caso de ausência de erros. Durante sua execução, o S2PA se conecta a todos os M-OBJs que possuem algum driver instalado e se inscreve para receber atualizações de mudanças em seus atributos. O fluxo de eventos gerado pelo M-Hub é implementado pela classe RxBus que utiliza um Observable do ReactiveX internamente. Dessa forma os eventos relacionados ao encontro e comunicação com M-OBJs são notificados a todos os objetos que se inscreveram para recebê-los. A aplicação cliente pode filtrar os eventos e inscrever Observers para cada um deles individualmente, recebendo, juntamente com o evento, um objeto com informações relevantes. Como exemplo, ao receber uma notificação de M-OBJ descoberto, recebe-se, também, um objeto MobileObject com as suas informações.

O M-Hub para Android acompanha uma implementação da tecnologia WPAN BLE. Essa implementação depende da biblioteca RxAndroidBLE[6], uma biblioteca que simplifica o uso do BLE no Android e oferece uma interface baseada no ReactiveX, permitindo execução de operações do BLE em segundo plano com notificação dos resultados por meio de fluxos observáveis (Observables).

[6]https://github.com/dariuszseweryn/RxAndroidBle. Acesso em 31/10/2024.

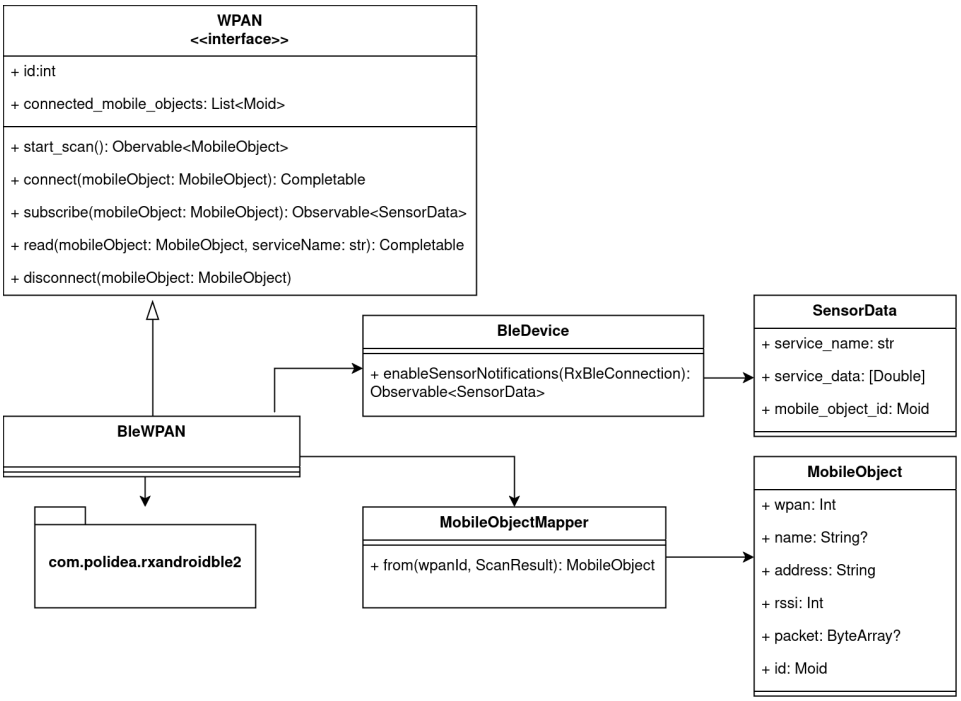

Figura 5.2: Arquitetura de classes do BleWPAN.

Na Figura 5.2 mostra as interações entre as classes do BleWPAN e a biblioteca RxAndroidBLE. O BleWPAN obtém os dados das buscas do RxAndroidBLE e as transforma por meio das classes MobileObjecMapper e BleDevice (driver do dispositivo) para criar, respectivamente, MobileObject e SensorData. Os dados transformados são, em seguida, propagados para observadores inscritos do S2PA.

Idealmente a existência de uma biblioteca BLE para Python similar à RxAndroidBLE, isto é, com uma interface que oferta os dados solicitados por meio de fluxos observáveis do ReactiveX, permitiria manter a arquitetura de classes praticamente idêntica a do BleWPAN em Kotlin. A busca por uma biblioteca como essa não encontrou nenhum resultado satisfatório, então a decisão foi utilizar uma outra biblioteca BLE compatível com buscas assíncronas por meio de tarefas do asyncio e criar uma classe wrapper que utiliza essa biblioteca e oferece uma interface similar a da RxAndroidBLE.

A biblioteca escolhida foi a Bleak[7]. Essa biblioteca fornece classes para a busca de dispositivos BLE, para a conexão com os dispositivos e para leitura, escrita e inscrição na mudança de valores de atributos dos dispositivos. Todas essas interações são compatíveis com o asyncio e podem ser executadas concorrentemente com outras tarefas do M-Hub. Após a inclusão da biblioteca bleak e substituição da RxAndroidBLE, o diagrama de classe do BleWPAN em Python segue a Figura 5.3.

[7]https://bleak.readthedocs.io/en/latest/index.html. Acessado em 31/10/2024.

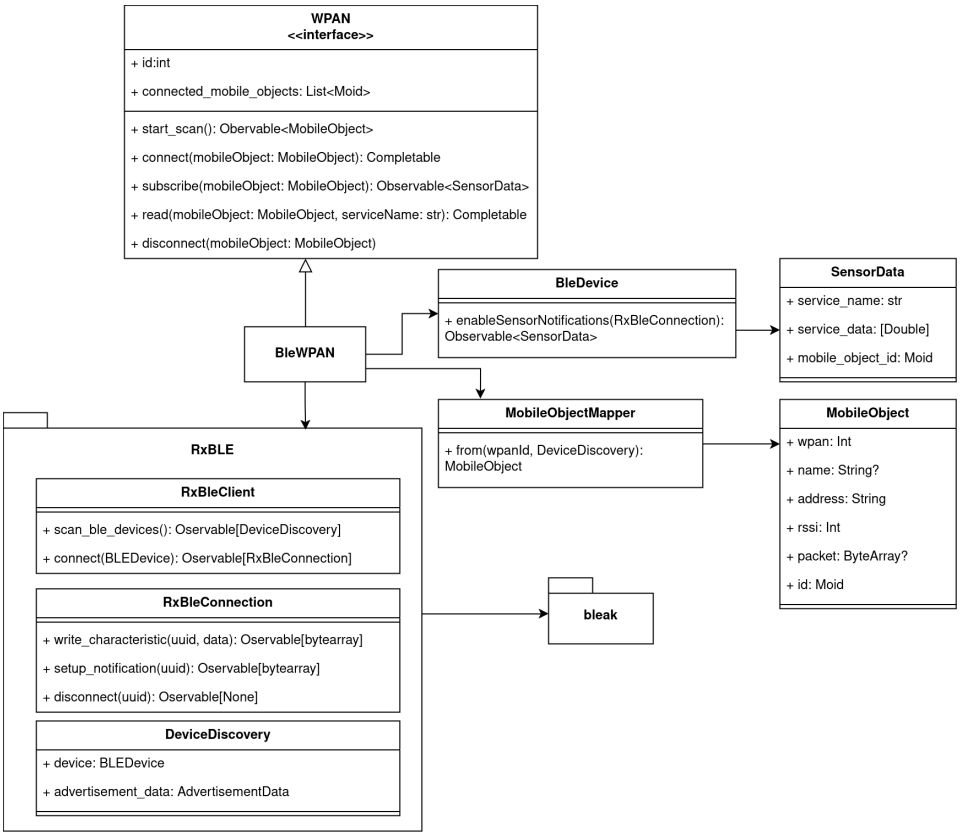

Figura 5.3: Arquitetura de classes do BleWPAN Python com o módulo wrapper RxBle.

6
Implementação, avaliação e considerações finais

6.1
Comentários Sobre a Implementação

Durante a execução do M-Hub foi possível perceber que, ao interromper o sistema e executá-lo novamente, os dispositivos uma vez conectados anteriormente não são mais enxergados pela aplicação. Após minuciosa análise, foi notado que os SensorTags haviam se conectado automaticamente com o Raspberry Pi, mas a biblioteca bleak não oferece meios de acessar dispositivos já conectados. Em seguida, ao pesquisar sobre isso, foi encontrado um relato do criador da biblioteca citando que essa funcionalidade está fora do escopo do bleak.

O BlueZ, sistema responsável pelo gerenciamento do Bluetooth e do BLE em sistemas Linux como o Raspberry Pi 4, permite encontrar esses dispositivos. A API do BlueZ funciona através do DBus, um sistema de barramento entre vários subsistemas e aplicações de usuários em sistemas Linux. Portanto, uma possível resolução dessa falha seria substituir a bleak por uma biblioteca como a dbus-fast [1], que possibilita uma interação direta com a API do BlueZ. Essa abordagem oferece maior controle e eficiência na interação do BLE, incluindo a capacidade de gerenciar os dispositivos já reconhecidos pelo BlueZ. Entretanto a API do BlueZ é mais complexa e trabalhar diretamente com ela exige maior esforço devido ao nível de detalhe necessário. Por conta dessas demandas, não foi possível concluir essa implementação dentro do tempo disponível.

Também é importante destacar que a utilização da RxPy conta com algumas desvantagens em relação à RxJava que é utilizada em Kotlin. A versão de Python conta com apenas o tipo básico de sequência Observable, enquanto a RxJava oferece um conjunto de variados tipos de sequências que são amplamente utilizados no M-Hub. O tipo Observable pode emitir itens, por meio da callback 'onNext(item)', pode emitir uma notificação de que a sequência foi encerrada, por meio da callback 'onCompleted()' e pode notificar um erro encerrando a emissão, por meio da callback 'onError()'.

O primeiro deles é o Completable[2] que é um Observable que não produz itens, apenas notifica que uma operação foi concluída, portanto não aceita uma callback 'onNext()' e apenas notifica que uma operação foi concluida através da

[1] https://github.com/Bluetooth-Devices/dbus-fast. Acesso em 20/12/2024

[2] http://reactivex.io/RxJava/2.x/javadoc/io/reactivex/Completable.html. Acesso em 20/12/2024.

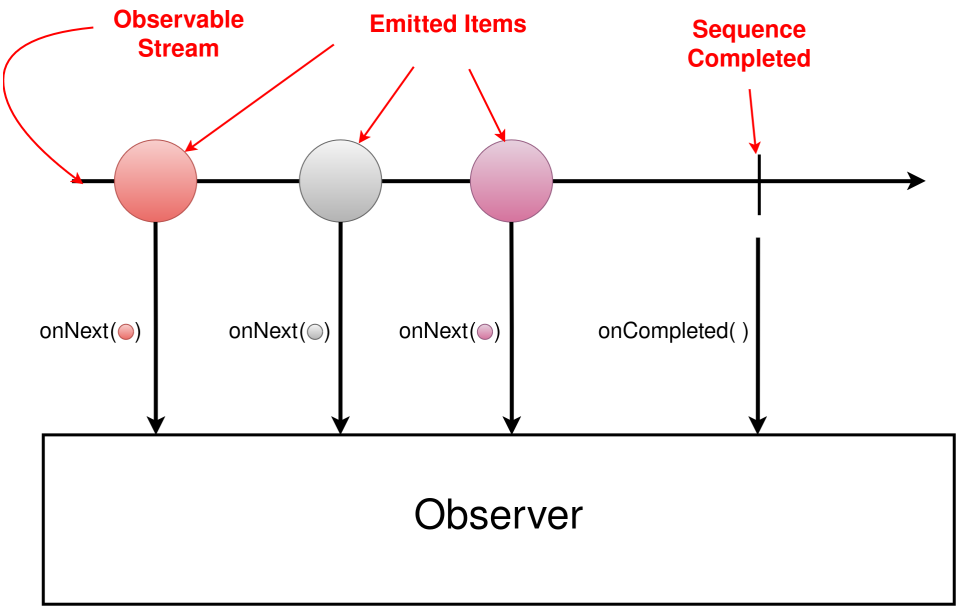

Figura 6.1: Sequência observável no ReactiveX

callback 'onCompleted()'. A sua substituição foi pelo Observable convencional mas que recebe None ao invés de não receber a callback 'onNext()'.

O segundo é o Flowable[3], que é uma sequência que produz itens e monitora a taxa de consumo dos consumidores inscritos. Sua utilidade se dá, principalmente, para evitar erros de limite de memória quando é esperado que a taxa de emissão de um certo tipo de item seja alta. Se a taxa de consumo for inferior à taxa de emissão, o Flowable aplica a estratégia de backpressure, definida em sua criação, para lidar com os itens acumulados. Entre essas estratégias estão, por exemplo, a buffer, que armazena todos os itens em um buffer intermediário, a latest, que descarta os itens não consumidos mais antigos e a drop que descarta todos os itens que não foram consumidos imediatamente. A alternativa adotada para substituir o uso do Flowable foi utilizar o Observable convencional e emitir erro ao atingir o limite. Essa alternativa não é a ideal, pois o sistema deve ser capaz de adotar diferentes estratégias para lidar com o excesso de itens. Existem bibliotecas como a rxbp[4] que fornecem uma extensão para abarcar estratégias de back pressure ao Observable do RxPy. Entretanto, para essa versão inicial do M-Hub Python essa opção foi deixada para ser analisada futuramente em virtude de priorizar o desenvolvimento de outros aspectos do sistema.

[3]http://reactivex.io/RxJava/2.x/javadoc/io/reactivex/Flowable.html. Acesso em 20/12/2024.

[4]https://github.com/MichaelSchneeberger/rxbp. Acesso em 20/12/2024.

6.2 Planejamento e execução de testes funcionais

Os testes funcionais foram executados executando uma aplicação M-Hub para detectar e ler sensores de um dispositivo SensorTagCC2650 a partir de um Raspberry Pi 4 Model B rodando o sistema operacional Linux raspberrypi 6.6.31, ambos no mesmo ambiente. O método utilizado foi criar uma aplicação que executa o M-Hub e se inscreve nos eventos de descoberta, conexão e recebimento de dados de sensores e mede os intervalos de tempo entre o recebimento dos eventos com a função 'time.time()' da biblioteca padrão do Python. Os dados gerados são o Tempo de Descoberta do SensorTag (DT), o Tempo de Conexão (CT) e o Tempo para receber o primeiro Dado de algum Sensor (SDT). O experimento foi executado 25 vezes para se obter medidas de média e desvio padrão. Entre cada execução foi utilizado o software D-Feet, um cliente D-Bus, que permite enviar comandos diretamente para o BlueZ, para desconectar o SensorTag e contornar os erros de reconexão discutidos na seção 6.1.

	DT (s)	**CT (s)**	**SDT (s)**
Mean	0.189	2.473	1.635
Standard Deviation	0.245	1.290	0.173

Tabela 6.1: Desempenho da comunicação com Mobile Objects.

Vale ressaltar que em uma conexão BLE, a etapa mais demorada é a descoberta dos serviços que o dispositivo oferece. Esse tempo pode ser consideravelmente reduzido quando o dispositivo já é reconhecido pela biblioteca utilizada, como demonstrado por (TALAVERA et al., 2015). Isso depende de uma biblioteca e de um hardware que forneçam a possibilidade de reconexão. O BlueZ, como implementando no Raspberry Pi, fornece essa possibilidade, além de permitir, também, ajustes nos parâmetros de periodicidade e tempo de duração das buscas. Isso evidencia a importância de utilizar uma biblioteca mais robusta que a experimentada neste trabalho.

6.3 Contribuições deste trabalho para a comunidade

Este trabalho contribui para a expansão da abrangência da ContextNet, sendo um esforço em direção a disponibilizar a execução do M-Hub em outros dispositivos além de smartphones Android. Isso significa, também, um avanço na exploração das técnicas de computação pervasiva no desenvolvimento da computação na borda com dispositivos móveis conectados utilizando a linguagem Python e Raspberry Pis.

Outra contribuição deste trabalho é servir como uma análise e documentação que explora os detalhes de arquitetura e funcionamento dos módulos S2PA e WPAN do M-Hub. O texto deste trabalho também aponta algumas melhorias que podem ser feitas ao M-Hub Python para que se aproxime ainda mais do M-Hub Kotlin.

6.4 O que aprendi com o trabalho

Uma alteração significativa foi a substituição da biblioteca RxBleAndroid por uma solução própria que utiliza a biblioteca bleak e converte os resultados para Observables que interagem com o asyncio de modo a criar tarefas que produzem os itens e notificam os Observers que se inscreveram. Essa alteração foi possível de ser realizada afetando minimamente o módulo WPAN e sem afetar de nenhuma forma os outros módulos do M-Hub. Com essa mudança, que se mostrou descomplicada, aprendi ainda mais sobre a importância da modularidade e do baixo acoplamento em projetos que precisam ser flexíveis.

Com o trabalho também aprendi sobre a tecnologia Bluetooth e Bluetooth Low Energy. Aprendi que possue diferenças de objetivos entre as duas tecnologias, dado que o BLE foi uma forma de atingir uma conectividade a dispositivos de baixo poder de processamento e que precisam consumir pouca energia. Foi possível entender os modelos GATT e GAP do BLE e os passos necessários para interagir com eles para descobrir os serviços de um dispositivo BLE e ler e escrever as características dos dispositivos BLE.

Outro aprendizado muito relevante foi sobre o ReactiveX como uma forma de facilitar o desenvolvimento de aplicações reativas. O ReactiveX sugere uma forma diferente de estruturar o fluxo de dados de um sistema e promove uma abordagem que simplifica e minimiza falhas de sincronia entre os diferentes fluxos além de reforçar a importância de pensar cuidadosamente nas etapas do processamento dos dados da aplicação desde suas aquisições até atingir o objetivo final de cada um deles.

Também aprendi sobre como aplicativos de Android são estruturados e um pouco sobre a API disponível para isso, além de aprender sobre a linguagem Kotlin. O estudo do aplicativo exemplo me ensinou como desacoplar a interface de usuário com os mecanismos de lógica e os serviços que executam em segundo plano no Android.

6.5 Oportunidades futuras

Este trabalho, como uma etapa inicial do desenvolvimento do M-Hub Python, abre caminho para diversas oportunidades futuras. Primeiramente, há o potencial para expandir a compatibilidade do sistema com outras formas de comunicação, como integração da tecnlogia Wi-Fi Direct no S2PA e inter-comunicação direta com entre dispositivos M-Hub ao se encontrarem em um mesmo espaço físico por meio de uma das tecnologias WPAN, o que ampliaria ainda mais a versatilidade e o escopo de aplicações do M-Hub.

Outra oportunidade está no aprimoramento da implementação do BLE, já que a versão desenvolvida possui problemas de conectividades que não são possíveis de contornar com a biblioteca escolhida bleak. Como sugerido na Seção 6.1, um trabalho futuro que substitua a bleak por outra com suporte a mais funcionalidades como a interação direta com o BlueZ via D-Bus seria muito benéfico para o M-Hub.

Além disso, o sistema também pode ser aprimorado com a integração de mais mecanismos de análise de dados em tempo real nos dispositivos M-Hub. O uso de técnicas de inteligência artificial e aprendizado de máquina, por exemplo, em conjunto com a CEP Engine que já existe no M-Hub Kotlin, ampliaria as possibilidades de interpretação dos dados extraídos do ambiente do M-Hub.

Por fim, se faz necessário o desenvolvimento, em Python, dos módulos restantes que o M-Hub Kotlin já possui e que não foram abarcados por este trabalho. Estes são o Connection Gateway, que fornece conectividade ao ContextNet Core; o MEPA Gateway, que itengra processamento de eventos com CEP Engines; e os módulos de obtenção de drivers de M-OBJs dinamicamente dos repositórios da ContextNet.

Essas possibilidades apontam para o aprimoramento contínuo do M-Hub e demonstrar a sua capacidade como uma plataforma versátil e poderosa, além do potencial da ContextNet como uma abordagem inovadora para a pesquisa e desenvolvimento nas áreas de internet das coisas móveis e computação pervasiva.

7
Referências

Bluetooth SIG. **The Bluetooth Technology for Linux Developers Study Guide**. 2022. Acesso em 31/10/2024. Disponível em: <https://www.bluetooth.com/blog/the-bluetooth-for-linux-developers-study-guide/>. Citado na página 13.

ENDLER, M.; SILVA, F. S. E. Past, present and future of the contextnet iomt middleware. **Open J. Internet Things**, v. 4, p. 7–23, 2018. Disponível em: <https://api.semanticscholar.org/CorpusID:51689202>. Citado 2 vezes nas páginas 6 e 8.

HEYDON, R. **Bluetooth Low Energy: The Developer's Handbook**. Pearson Education, 2012. ISBN 9780132888400. Disponível em: <https://books.google.com.br/books?id=aDReS05LxNoC>. Citado 3 vezes nas páginas 5, 13 e 14.

NAHRSTEDT, K. et al. Internet of mobile things: Mobility-driven challenges, designs and implementations. In: **2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)**. [S.l.: s.n.], 2016. p. 25–36. Citado na página 6.

TALAVERA, L. E. et al. The mobile hub concept: Enabling applications for the internet of mobile things. **2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)**, p. 123–128, 2015. Disponível em: <https://api.semanticscholar.org/CorpusID:15916548>. Citado na página 24.

TALAVERA, L. E. et al. Design and implementation of a flexible architecture for mobile edge devices. **Monografias em Ciência da Computação - Pontifícia Universidade Católica do Rio de Janeiro.**, 2023. Disponível em: <https://bib-di.inf.puc-rio.br/ftp/pub/docs/techreports/23_04_talavera.pdf>. Citado 3 vezes nas páginas 5, 8 e 9.