Pontifícia Universidade Católica
do Rio de Janeiro

Sheriton Rodrigues Valim

Um serviço de Middleware para atuação genérica e remota de dispositivos na Internet das Coisas Móveis

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio.

Orientador: Prof. Markus Endler

Rio de Janeiro
Setembro de 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Sheriton Rodrigues Valim

Um serviço de Middleware para atuação genérica e remota de dispositivos na Internet das Coisas Móveis

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

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

Profa. Noemi de La Rocque Rodriguez
Departamento de Informática – PUC-Rio

Profa. Debora Christina Muchaluat Saade
Universidade Federal Fluminense – UFF

Rio de Janeiro, 20 de Setembro de 2019

Sheriton Rodrigues Valim

Graduou-se em Engenharia da Computação pela Pontifícia Universidade Católica do Rio de Janeiro (Rio de Janeiro, Brasil).

Ficha Catalográfica

Valim, Sheriton Rodrigues

Um serviço de Middleware para atuação genérica e remota de dispositivos na Internet das Coisas Móveis / Sheriton Rodrigues Valim; orientador: Markus Endler. – 2019.
48 f: il. color. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2019.
Inclui bibliografia

1. Informática – Teses. 2. Internet das coisas;. 3. Middleware;. 4. Bluetooth;. 5. Atuação remota;. 6. Computação móvel.. I. Endler, Markus. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

À minha esposa Isabelle Valim, com quem compartilho todos os momentos. Obrigado por todo o incentivo, ajuda e por ter me dado o bem mais importante de todos: o nosso filho Arthur, que em breve estará conosco.

À minha família, especialmente meus pais, por todo o apoio e confiança depositada em mim durante todos esses anos.

Ao professor Markus Endler. Muito obrigado por todo apoio e compreensão. Obrigado por toda a confiança e incentivo. Obrigado também por todos os conselhos e puxões de orelha.

A todos os amigos do LAC, em especial ao Bruno Olivieri, Matheus Zeitune e Felipe Nogueira, sem os quais este trabalho não teria sido concluído.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo

Valim, Sheriton Rodrigues; Endler, Markus. **Um serviço de Middleware para atuação genérica e remota de dispositivos na Internet das Coisas Móveis**. Rio de Janeiro, 2019. 48p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A Internet das Coisas (do inglês, IoT) está se popularizando cada dia mais, se expandindo para aplicações em quase todos os setores de nossa socidade, impactando a economia e a vida cotidiana dos cidadãos. Com o crescimento das aplicações IoT, também ocorre uma expansão na demanda por novos dispositivos com capacidades de atuação, como lâmpadas inteligentes, fechaduras inteligentes, aparelhos de climatização (HVAC), máquinas industriais com capacidade de comunicação, robôs, drones, etc. Muitas plataformas de *middleware* estão sendo desenvolvidas para suportar o desenvolvimento de aplicativos de IoT distribuídos e facilitar a comunicação de sensores para a nuvem e recursos de *edge processing*. Mas surpreendentemente, muito pouco foi feito para fornecer mecanismos de suporte genéricos e em nível de *middleware* para detectar dispositivos controláveis e executar comandos de atuação, ou seja, transferi-los para o dispositivo. Este trabalho apresenta uma extensão ao *middleware ContextNet* que provê suporte à atuação remota e genérica sobre dispositivos inteligentes conectados pela Internet das Coisas.

Palavras-chave

Internet das coisas; Middleware; Bluetooth; Atuação remota; Computação móvel.

Abstract

Valim, Sheriton Rodrigues; Endler, Markus (Advisor). **A Middleware Service for generic and remote actuation of IoT devices in the Internet of Mobile Things**. Rio de Janeiro, 2019. 48p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The Internet of Things (IoT) is becoming increasingly popular, expanding into applications in almost every sector of our society, impacting the economy and daily life of citizens. As IoT applications grow, so does the demand for new devices with actuation capabilities, such as smart light bulbs, HVAC devices, smart locks, communication-capable industrial machines, robots, drones, and so on. Many software platforms are being developed to support the development of distributed IoT applications and to facilitate cloud sensor communication and edge processing capabilities. But surprisingly, very little has been done to provide middleware-level support and generic mechanisms for discovering devices and their interfaces, and executing activation commands, i.e. transferring them to the device. This work presents an extension to the ContextNet Middleware to support generic and remote actuation on devices connected by the Internet of Things.

Keywords

Internet of Things; Middleware; Bluetooth; Remote Actuation; Mobile Computing.

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

Aplicações IoT para *SmartHomes*, *SmartBuildings*, Transportes e Logística, *Healthcare*, Indústria 4.0, entre outros segmentos estão em pleno crescimento nos dias atuais. Um elemento essencial comum a todas essas aplicações são os dispositivos inteligentes com atuadores, ou *Actionable Smart Things* (AST), pela sua capacidade de interferir ou alterar o ambiente físico em que se encontram.

Alguns exemplos desses ASTs são aparelhos de climatização (HVAC), lâmpadas e cadeados inteligentes, displays, regadores ou pulverizadores (*sprinklers*), alto-falantes, sirenes, robôs e *drones*. Em todos estes casos, o dispositivo geralmente se conecta a algum *gateway/hub* usando alguma interface de comunicação sem fio de curto alcance. O *gateway/hub*, por sua vez, fica encarregado de conectar os ASTs aos serviços executados em uma nuvem ou *cluster*.

Vários destes ASTs usam *Bluetooth Low Energy* (BLE) para comunicação sem fio devido ao baixo consumo de energia, descoberta rápida e emparelhamento com dispositivos, além do seu link de comunicação confiável. No entanto, a cobertura sem fio é restrita (aproximadamente 50 metros). Apesar disso, praticamente qualquer *smartphone*, *tablet* ou dispositivo de *streaming* moderno (p.ex. *Amazon Fire Cube*, *Nvidia Shield* e *MiBox*) possui interface *Bluetooth*, tornando-o elegível para desempenhar o papel de um *gateway* de *Internet* para as coisas inteligentes, contanto que estejam dentro da sua faixa de BLE.

A motivação deste trabalho surgiu da necessidade de se estudar a Internet das Coisas Móveis (IoMT), onde tanto os dispositivos inteligentes quanto os *gateways/hubs* podem ser movidos ou se mover de forma autônoma. Para implementar este suporte, foi desenvolvido o *Mobile Hub* (M-Hub), um serviço de *Middleware* para *Android* que é executado em segundo plano e é independente de outros aplicativos móveis [1].

O objetivo deste trabalho é permitir que a atuação sobre ASTs em aplicações IoT através de conjuntos de comandos simples de alto nível seja efetuada através do *Middleware ContextNet*. Os comandos de alto nível devem abstrair a complexidade dos protocolos nativos de cada dispositivo inteligente, além de permitir que desenvolvedores criem suas próprias APIs para atuação. Es-

ses comandos de alto nível são, então, traduzidos para sequências de *bytes* de acordo com a especificação de cada dispositivo, de forma que possam ser processados pelos ASTs. Para isso, dois novos componentes foram incorporados ao *Middleware ContextNet*: o *Mobile-Actuator* (M-Act), que é um microsserviço do M-Hub para cuidar da comunicação direta com um AST e o *Smart Objects Manager* (SOM), um microsserviço do SDDL responsável pelo registro e divulgação de dispositivos conectados. Esses dois serviços trocam mensagens MACTQuery no formato JSON para envio de comandos de atuação, notificações sobre conexão, desconexão e transferência de recursos necessários para que o M-Act possa efetuar a atuação sobre os ASTs em sua vizinhança.

Acredita-se que esse trabalho contribui para a área de IoMT, e especificamente para o tratamento da heteronegeidade no que tange à atuação. Isso é feito através do desenho, implementacão e avaliação de um par de serviços de *middleware* (na nuvem/*cluster* e nos M-Hubs) que facilitam a flexibilização, a extensibilidade e uma adaptação, online e em tempo real, para o controle de ASTs. Imagina-se que o resultado desta dissertação dê passos fundamentais para o apoio a atuação e controle na IoMT, ou seja, na IoT com mobilidade irrestrita de *hubs* e dispositivos inteligentes.

1.1 Objetivos

O principal objetivo deste trabalho é propor uma extensão ao *Middleware ContextNet* que possibilite a atuação remota sobre ASTs de forma genérica, ou seja, utilizando um conjunto de comandos de alto nível, definidos pela aplicação desenvolvida com base no *Middleware*, que possa ser aplicado a uma determina classe de dispositivos, independente dos protocolos específicos definidos pelos respectivos fabricantes. Esses comandos precisam ser traduzidos para o protocolo nativo definido pelos fabricantes de cada AST.

Os objetivos específicos são:

1. Desenvolver um serviço para lidar com o problema de interoperabilidade, ou seja, converter um comando genérico de atuação em um comando seguindo as especificidades do protocolo nativo do dispositivo a ser controlado.

2. Definir uma arquitetura de *driver* que contenha a especificação de um dispositivo e todas as etapas necessárias para converter um comando genérico de atuação no protocolo específico do dispositivo.

3. Desenvolver um serviço na nuvem que forneça um repositório de *drivers* de dispositivos que possam ser controlados por uma aplicação baseada no *ContextNet*.

4. Desenvolver um serviço que informe a lista de dispositivos conectados a rede e o endereço ou UUID de seus respectivos *gateways*.

5. Construir uma interface que permita ao usuário registrar um dispositivo e o respectivo *driver* na aplicação.

1.2 Organização da dissertação

Este trabalho está organizado da seguinte forma. No próximo capítulo, Capítulo 2, são apresentados os conceitos e tecnologias utilizadas para o desenvolvimento deste trabalho. No capítulo 3 é apresentada a solução proposta, detalhando a arquitetura da implementação que extende o *middleware ContextNet*. No capítulo 4 são apresentados os principais trabalhos relacionados e uma discussão sobre as semelhanças desses trabalhos com o trabalho apresentado. O capítulo 5 apresenta uma avaliação do trabalho através da implementação de uma aplicação de exemplo. Por fim, o capítulo 6 apresenta as conclusões e trabalhos futuros.

2
Fundamentos

Este capítulo apresenta, alguns conceitos e tecnologias fundamentais ao trabalho serão apresentados.

2.1
Atuação

Como muitas aplicações na Internet das Coisas objetivam o monitoramento, o controle e a automação de processos no mundo físico, é natural que além de sensores (os sentidos), sistemas de IoT também precisem interagir com atuadores.

Um atuador é um componente eletrônico de um equipamento ou máquina que é responsável por acionar, mover ou controlar um mecanismo do equipamento como, por exemplo, ligar, desligar ou colocar em modo *sleep*, modificar um potenciômetro, abrir uma válvula, etc. Em termos práticos, é um "controlador" do equipamento. Existem atuadores hidráulicos, pneumáticos, elétricos, termais, magnéticos ou mecânicos.

Sensores e atuadores são formas específicas de transdutores: dispositivos físicos que convertem uma forma de energia em outra. Assim, no caso de um sensor, o transdutor converte algum fenômeno físico em um impulso elétrico que pode ser interpretado para determinar uma leitura. Em contrapartida, o atuador opera na direção inversa de um sensor. Ele toma uma entrada elétrica e a transforma em ação física. Por exemplo, um motor elétrico, um sistema hidráulico e um sistema pneumático são todos tipos diferentes de atuadores.

Um atuador também pode ser visto como o "executor" de comandos de controle (ou atuação) recebidos do usuário ou de outra parte do sistema IoT. Nesse caso, os comandos remotos recebidos precisam ser interpretados e convertidos em sinais eletromecânicos mais simples.

2.2
ContextNet Middleware

Para suportar aplicações IoMT, o Middleware *ContextNet*[2] foi desenvolvido pelo *Laboratory for Advanced Collaboration* (LAC), na PUC-Rio. O *ContextNet* possui uma arquitetura de três camadas que consiste em: (a) um

núcleo SDDL, uma rede P2P de nós executando em uma nuvem ou *cluster* de máquinas estacionárias e interagindo com (b) *Hubs* móveis executando em dispositivos *Android* ou placas (*Raspberry PIs*, etc.), que usam sua interface WPAN para descobrir, selecionar, conectar e trocar dados ou comandos de atuação com sensores e atuadores, respectivamente, de dispositivos inteligentes (AST). Atualmente, o *Mobile Hub* é capaz de se conectar apenas a objetos inteligentes habilitados para *Bluetooth* clássico e *Bluetooth Smart* (BLE). A terceira camada (c) corresponde aos dispositivos IoT (sensores, atuadores e *Beacons*).

O núcleo do SDDL, descrito em [3], manipula a arquitetura de comunicação ativando uma plataforma de *middleware* orientada para mensagens móveis. Estas plataformas apresentam um protocolo móvel leve, o MR-UDP [3], que lida com conexões intermitentes, incluindo a transferência suave. Esse protocolo também pode manter fluxos de dados para os *Mobile Hubs* por meio do uso de mensagens de baixa taxa de manutenção (*heartbeat*).

O *Mobile Hub*, descrito em [4] é o nível de *middleware* de *smartphone* que suporta o M-Act como um microsserviço. O *Mobile Hub* fornece múltiplos acessos nativos do *Android* ao M-Act, como a conexão de nível inferior para os atuadores através de *Bluetooth*, *Bluetooth LE* e outros canais de comunicação.

2.3 *Mobile Hub*

O *Mobile Hub* (M-Hub) preenche a lacuna entre a conexão à *Internet* com o SDDL *Core* e as conexões sem fio de curto alcance com dispositivos inteligentes. É composto por microsserviços independentes que interagem através de um mecanismo *publisher/subscriber* usando o *EventBus*, ilustrado na Figura 2.1.

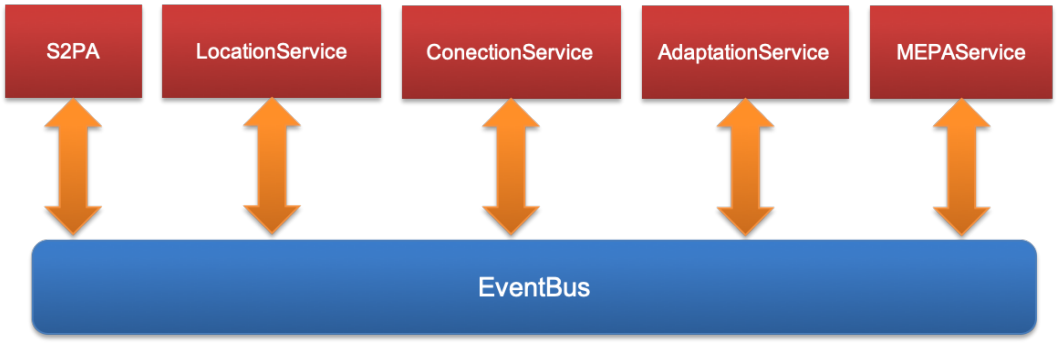

Figura 2.1: Eventbus.

O serviço ***Short-Range Sensor, Presence and Actuation* (S2PA)** implementa o *TechnologyListener* e interage com todos os elementos inteli-

gentes próximos a uma interface *Bluetooth Classic* ou BLE (mais detalhes na subseção 2.4).

O ***LocationService*** é responsável por amostrar a posição atual do M-Hub e anexá-la a qualquer mensagem enviada para o SDDL *Gateway* (GW), que pode ser um ponto geográfico inserido manualmente ou estático, ou a última geo-coordenada, obtida a partir do sensor GPS embutido do *smartphone*.

O ***ConnectionService*** executa o *ClientLib* para comunicação com o SDDL *Core*. Para otimizar a transmissão através do link de *Internet* de área ampla, esse serviço pode agrupar várias mensagens (comandos ou *acks*).

Para suportar o processamento local de dados de entrada ou de saída do SDDL *Core*, o M-Hub também apresenta o *Mobile EPA (M-EPA)*, que é um mecanismo completo de *Complex Event Processing* (CEP) para analisar fluxos de dados (comandos ou dados de sensores, por exemplo). Além disso, o mecanismo permite a realização de filtragem, agregação, sumarização e detecção baseada em janelas de padrões de eventos, que são descritos e implementados como regras independentes de *Event Condition Action* (ECA) em uma linguagem *Event Processing Language* (EPL) específica.

O ***AdaptationService*** é responsável por identificar o tipo dos dispositivos nas proximidades do M-Hub e instanciar objetos em memória que permitem ao S2PA interagir com esses dispositivos.

A periodicidade e a duração de muitas das ações desses serviços são influenciadas pelo nível de energia atual do dispositivo (que é classificado como BAIXO, MÉDIO e ALTO). Este nível é definido pelo último serviço, o ***Energy Manager***, que periodicamente verifica o nível da bateria do dispositivo, além de verificar se o *smartphone* está conectado a uma fonte de energia.

Com o propósito de gerenciar a atuação em dispositivos inteligentes, um novo microsserviço foi introduzido ao M-Hub neste trabalho: M-Act. Como a atuação envolve interações através do WPAN, é necessário maior detalhamento a respeito do S2PA, que fornece algumas funções básicas de interação do BLE para o M-Act.

2.4 S2PA

O *Service for Sensors, Presence and Actuators* (S2PA) [1] é um microsserviço central do M-Hub na tarefa de fazer a descoberta de, e a conexão com, M-OBJs (*Mobile-Objects*) próximos por meio de diferentes tecnologias sem fio de curto alcance (WPAN). A API do S2PA foi projetada para lidar com múl-

tiplas WPANs, implementando uma interface que pode ser mapeada para os recursos da tecnologia WPAN suportada.

Para suportar qualquer WPAN, o S2PA define alguns métodos e interfaces básicos que devem ser implementados em todas essas tecnologias:

1. Descoberta e conexão com M-OBJs através da varredura (*scan*) periódica, à busca de anúncios de dispositivos

2. Estabelecimento de uma conexão e consulta sobre os serviços fornecidos por cada M-OBJs (seus sensores ou atuadores)

3. Leitura e escrita de atributos (*capabilities*) de um ou mais serviços (leitura de um sensor ou escrita de um comando para um atuador)

4. Notificações sobre desconexão de M-OBJs sempre que a leitura ou escrita dos atributos resultar em falha

Precisamente, o S2PA disponibiliza as interfaces abstratas `Technology` e `TechnologyListener`, ilustradas na Figura 2.2, que devem ser implementadas por cada tecnologia WPAN que venha a ser usada. A interface `Technology` define os métodos necessários para executar as principais funcionalidades da tecnologia WPAN, como descoberta e conexão aos sensores e atuadores nas proximidades do M-Hub, descoberta de serviços disponíveis por cada dispositivo, leitura e envio de dados (por exemplo, leitura dos dados de sensores e envio de comandos para atuadores) e notificação sobre desconexão de dispositivos. Já a interface `TechnologyListener` é responsável por ouvir todos os eventos importantes disparados por cada tecnologia WPAN e publicá-los para outros serviços interessados.

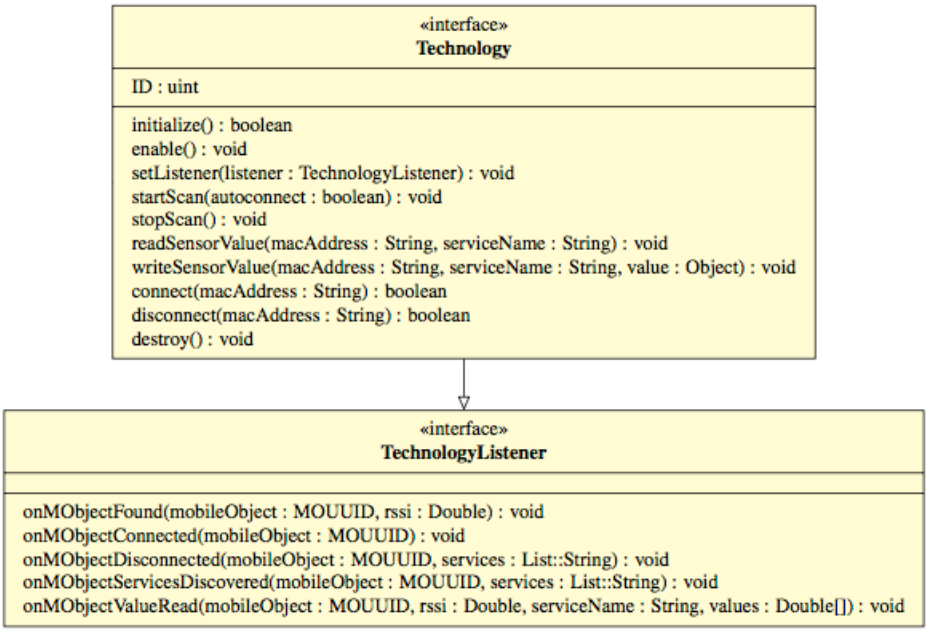

Figura 2.2: Interfaces `Technology` e `TechnologyListener` [5].

Atualmente, o M-Hub provê suporte às tecnolgias *Bluetooth Low Energy* (BLE) e *Classic Bluetooth* seguindo o padrão S2PA.

O S2PA também disponibiliza a interface *TechnologyDevice* que deve ser utilizada para modelar e representar os dispositivos físicos (sensores e atuadores) em objetos M-OBJ, permitindo com que o M-Hub interaja com esses dispositivos. Os M-OBJ são implementados diretamente no M-Hub no mesmo pacote da implementação da tecnologia WPAN correspondente. Por exemplo, um dispositivo físico que se comunique através de BLE deve ter a implementação do seu modelo M-OBJ implementado junto da implementação do BLE no M-Hub.

Este serviço S2PA também se encarrega de iniciar outros serviços do M-Hub, manter o registo dos sensores/atuadores encontrados até então e enviar os dados coletados (de sensores ou sobre a desconexão de M-OBJs) como mensagem de difusão a todos os componentes do M-Hub que a necessitem.

2.5 A WPAN *Bluetooth Low Energy*

Bluetooth Low Energy (BLE)[1], por vezes também chamado de *Bluetooth Smart*, é um subconjunto mais leve e eficiente do *Bluetooth* clássico e foi introduzido como parte da especificação *Bluetooth* 4.0. Se por um lado existe uma certa sobreposição com *Bluetooth* clássico, o BLE representa uma versão nova

[1] www.bluetooth.com

e bem diferente desse WPAN e foi concebido especialmente para transmissões frequentes, mas de poucos dados, o que é muito comum em IoT.

Há um grande espectro de protocolos WPAN para serem usados no projeto e implementação de produtos IoT, mas o que torna BLE tão interessante é que o BLE é um protocolo de ampla adoção e que permite ao periférico (e seu componente mestre) ser detectado e estabelecer uma conexão com qualquer plataforma *mobile* (iOS, *Android*, telefones *Windows*, etc).

O funcionamento do BLE é gerido pelo *Generic Access Profile* (GAP) e *Generic Attribute Profile* (GATT). O GAP controla as conexões e os anúncios de dispositivos e determina como dois dispositivos podem (ou não) interagir entre si. O GAP define várias funções/papéis para os dispositivos, mas os dois conceitos principais são: dispositivo central (mestre) e dispositivos periféricos (escravos). O GATT, por sua vez, define como ocorre a troca de dados usando atributos pré-definidos. Neste trabalho, os ASTs são exclusivamente os dispositivos periféricos e os *Mobile Hubs* fazem o papel de dispositivo mestre.

Na fase de anúncio (e descoberta mútua), o periférico periodicamente retransmite o seu pacote de anúncio a cada *advertisement interval*. Uma periodicidade maior (e.g. a cada 2 segundos) economiza mais energia do dispositivo periférico mas, em compensação, este será menos responsivo, demorando mais para ser descoberto e acessado pelo dispositivo mestre. O GATT permite com que o dispositivo periférico trafegue alguma informação (alguns *bytes*) como carga de transporte (*payload*) do pacote de anúncio.

Uma vez estabelecida a conexão entre um periférico e um mestre BLE, o processo de anúncio geralmente se encerra e não é mais possível enviar pacotes de anúncios. A partir desse momento, o dispositivo mestre irá consultar os serviços disponíveis no periférico, para que ambos usem os serviços e características GATT para se comunicar em ambas as direções, lendo e escrevendo dados em cada uma das características descobertas.

No nível físico, todos os dispositivos BLE operam em 40 canais espalhados na faixa de 2,4 GHz. Três destes canais são reservados aos anúncios e os demais canais são para troca de dados. Há apenas um tipo geral de pacote, especifico para anúncios e para os tipos de dados. Dependendo da potência de transmissão e da sensibilidade de recepção, o alcance varia entre 20 metros (um transmissor a 0,01mW e um receptor com sensibilidade de -70dBm) e 100 metros (TX a 10mW e RX -90dBm).

2.6 *Drivers* de dispositivos atuáveis

Existe uma grande variedade de dispositivos IoT atuáveis, que podem ser móveis ou não, podem ter controles discretos (liga/desliga) ou contínuos (velocidade) e podem ter diferenças significativas de projeto e do *hardware* utilizado (e.g. tipo de motores e eixos). Isso tudo faz com que a natureza dos comandos de atuação varie muito e esteja bastante vinculada à finalidade do dispositivo. Consequentemente, cada dispositivo tem microcontroladores que aceitam códigos (de máquina) bem específicos e de baixo-nível de abstração para ser controlado. Esse código de controle é chamado *driver* e, geralmente, é escrito pelo fabricante do dispositivo. Cada *driver*, normalmente, lida com um tipo ou, no máximo, uma classe de dispositivos intimamente relacionados.

Em sistemas operacionais, os *drivers* dos dispositivos periféricos geralmente fazem parte do núcleo (*kernel*) do sistema operacional. No entanto, é possível desenvolver sistemas onde os *drivers* são executados no espaço do usuário. Nesta forma de implementação, os *drivers* ficam isolados uns dos outros e fora do *kernel*, eliminando uma grande fonte de potenciais falhas no sistema. Segundo [6], esta é a melhor forma de se desenvolver sistemas confiáveis.

É preciso que se tenha um modelo bem definido do que um *driver* faz e como ele interage com o resto do sistema operacional. Geralmente, os *drivers* compõem a camada mais baixa do sistema operacional, como mostra a Figura 2.3, ficando mais próximos das portas de comunicação com os dispositivos de *hardware*.

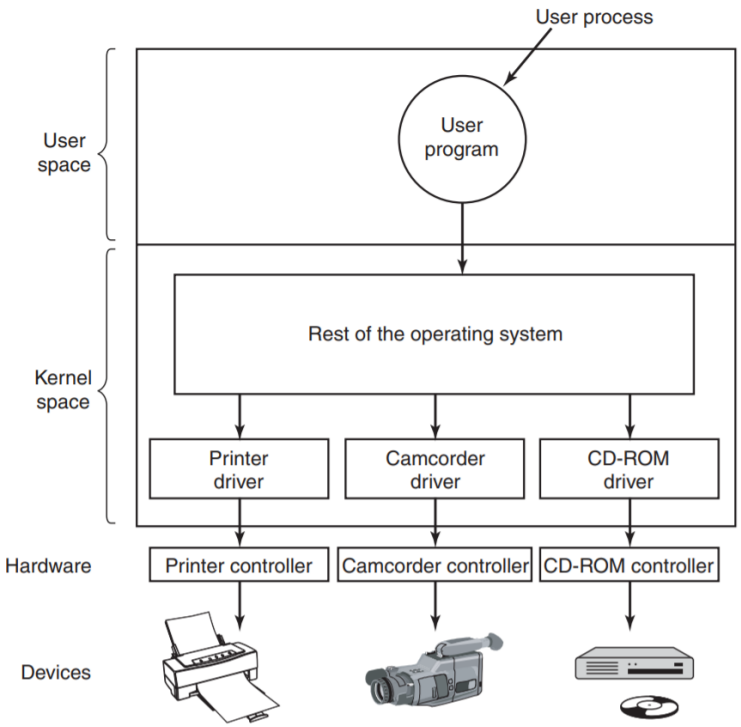

Figura 2.3: Posicionamento lógico de *drivers* de dispositivos [6].

Os *drivers* são geralmente classificados em duas categorias, de acordo com os dispositivos que controlam. Existe a categoria dos *drivers* de dispositivos de bloco (como discos, que contêm vários blocos de dados a serem endereçados independentemente) e os *drivers* dos dispositivos de caracteres (como teclados e impressoras, que geram ou aceitam fluxos de caracteres). Para cada tipo de dispositivo, o sistema operacional define uma interface de programação (API) padrão que todos os *drivers* devem prover. Estas interfaces consistem em procedimentos executados pelo sistema operacional para que os *drivers* possam ser utilizados. Tipicamente, os *drivers* são carregados dinamicamente no sistema durante a execução como se fossem *plugins* do núcleo do sistema operacional.

Um *driver* de dispositivo tipicamente possui diversas funções. A mais óbvia é ler/registrar as solicitações do sistema. Mas, além disso, um *driver* pode inicializar dispositivo, gerenciar seu estado, monitorar e controlar seu consumo de energia e tratar os eventos recebidos.

Em geral, os *drivers* de dispositivos possuem estruturas semelhantes. O *driver* é iniciado com a verificação e validação dos parâmetros de entrada. Caso não sejam parâmetros válidos, um erro é retornado. Se forem válidos, uma tradução de termos abstratos para concretos pode ser necessária. Em um *driver* de disco, por exemplo, isso pode significar a conversão de um número

de bloco linear para número de faixas ou setor.

Em seguida, o *driver* pode verificar se o dispositivo se encontra em uso. Se sim, a solicitação é registrada para posterior execução. Se o dispositivo estiver ocioso, ocorre uma verificação de *status* do *hardware* a fim de saber se a solicitação já pode ser executada.

Controlar um dispositivo significa transmitir (uma sequência de) comandos de máquina para ele e, eventualmente, fazer uma consulta sobre o estado de funcionamento do dispositivo. O *driver* é o lugar onde essa sequência é definida e transferida para o microcontrolador.

3
Proposta de extensão do Middleware

Na maioria dos sistemas IoT atuais, a lógica específica da atuação é geralmente codificada e entrelaçada com a lógica restante da aplicação. Isto ocorre porque, até o momento, os aplicativos de IoT são adaptados para aplicações com protocolos de atuação específicos (ou proprietários) e instruções de atuação de baixo nível. Muitas vezes, um mesmo tipo de AST desenvolvido por diferentes fabricantes possui diferentes protocolos de interação, isto é, comandos com características diferentes para um mesmo fim como, por exemplo, as lâmpadas inteligentes das empresas *MeeToo*[1] e *MagicBlue* [2]. Ambas as lâmpadas inteligentes utilizam BLE como meio de comunicação, possuem funções para ajustar a intensidade da luz, alterar as cores e possuem aplicativos para *smartphones* que permitem a um usuário controlar estas funções. Enquanto o protocolo da versão desenvolvida pela *MeeToo* exige que seja enviado um *array* de quatro *bytes* para definir a cor e a intensidade da luz em um único comando (**BB**, **GG**, **RR**, **WW**), o protocolo da versão desenvolvida pela *Magic Blue* requer um *array* de sete *bytes* para definir a cor (0x56, **RR**, **GG**, **BB**, 0x00, 0xF0, 0xAA) e outro *array* de sete *bytes* para definir a intensidade da luz (0x56, 0x00, 0x00, 0x00, **WW**, 0x0F, 0xAA), separadamente [7].

No entanto, quando são necessárias aplicações para interagir com (e agir com base em) muitos e diferentes tipos de ASTs ao longo do tempo, é indicado que soluções codificadas sejam substituídas por um mecanismo mais flexível, implementado na camada de *middleware*.

Considere, por exemplo, um aplicativo IoT que seja capaz de escurecer (ou alterar a cor de) todas as lâmpadas inteligentes dentro de várias salas de aula de uma instituição. Considere, ainda, que estas salas pertencem a departamentos diferentes e que as lâmpadas inteligentes são de fabricantes diferentes e, portanto, utilizam comandos GATT / BLE diferentes para os possíveis controles definidos pelo fornecedor. Assim, se um usuário precisa controlar as luzes em um espaço anteriormente não visitado, ele primeiro terá que inspecionar as lâmpadas e baixar o aplicativo móvel para, finalmente,

[1] https://pt.aliexpress.com/item/32659685459.html
[2] https://br.gearbest.com/smart-light-bulb/pp_230349.html

conseguir alterar a intensidade ou a cor da luz. Além disso, uma mesma sala pode conter lâmpadas inteligentes de diferentes fornecedores. Por outro lado, seria bom se o usuário tivesse acesso a comandos de alto nível como *setdim (50)*, *setRGB (20,10,50)* ou *turn (off)* , que seria aplicado uniformemente a qualquer lâmpada inteligente, em qualquer sala.

Outro exemplo hipotético é a utilização de robôs de aspiração futuristas, instalados para limpar o chão de uma casa. Muitos desses robôs aspiradores já aprendem a planta da casa a ser limpa para que, na maior parte do tempo, eles possam se mover e limpar bem todos os cômodos. Mas, de vez em quando, o robô pode enfrentar algum obstáculo que o impeça de prosseguir com o itinerário planejado. Neste caso, o robô pode obter informações sobre o tipo e identificação do provável obstáculo, através de câmeras ou sensores. Para isto, o robô poderia emitir comandos simples de alto nível como *door.open (X)*, *wheelchair.rotate (W, 30)* ou *serumSuppport.move (Z, 100)* para abrir um porta X, girando 30 graus em um obstáculo W ou avançando 1 metro em um obstáculo Z.

Uma alternativa para solucionar o problema da falta de padronização dos protocolos nativos e permitir que diversos ASTs se tornem compatíveis com o sistema é a utilização de componentes de *software* semelhantes aos *drivers* de dispositivos [6] nos sistemas operacionais. O objetivo desses componentes de *softwares* tipo (*drivers*) é fazer a tradução do protocolo de controle específico do AST para a API da plataforma na qual deseja-se utilizar o dispositivo e vice-versa. Desta forma, a infraestrutura de *software* (*middleware*) para IoT pode, sempre que identificar um novo AST, carregar automaticamente e sob demanda - em tempo de execução - o *driver* correspondente de um repositório do *middleware*. Assim, será capaz de converter comandos genéricos de atuação para o protocolo específico (sequência de *bytecode*) definido pelo fabricante do AST recentemente adicionado ou encontrado.

A utilização de *drivers* sob demanda viabiliza um cenário de atuação em grande escala, já que os *drivers* não precisam ser distribuídos e permanecer em todos os *gateways/edge devices*, mas sim podem ser ativados e desativados à medida que isso for necessário. Isso é, novos *drivers* podem ser baixados e *drivers* obsoletos podem ser descartados ao longo do tempo, permitindo com que os *gateways* se adaptem aos novos ASTs sem perder a capacidade de controlar os ASTs antigos.

Além disso, a utilização dos *smartphones* como *gateways/hubs* intermediários no processo de atuação, como ocorre com os *Mobile hubs*, permite com que diversos ASTs possam ser controlados remotamente por um mesmo cliente, de forma simultânea, mesmo que os ASTs estejam geograficamente localizados

em diferentes regiões, pois basta que diferentes *gateways/hubs* móveis estejam fazendo a conversão de comandos de atuação genéricos para os protocolos nativos dos ASTs.

Embora na IoMT o sistema tenha que lidar com a eventual desconexão sem fio do AST, o que pode acontecer quando o *smartphone* ou o AST se afasta da cobertura do BLE, isso é muito raro quando o usuário do *smartphone* é aquele que está controlando a atuação no AST. No entanto, se os comandos de atuação são gerados remotamente e o *smartphone* atua apenas como um intermediário, para que exista a atuação confiável é necessário que exista uma conexão estável entre o *gateway* e o AST. Ou seja, é preciso contar com a presença de vários M-Hubs próximos ao AST, ou é preciso que seja possível garantir que o M-Hub e o AST não possam se mover para além dos limites da cobertura BLE.

Ao contrário de outros trabalhos que assumem que os dispositivos inteligentes de IoT têm memória e recursos de processamento suficientes para permitir o fornecimento de uma API com operações complexas e de alto nível para controle de atuação, ou mesmo a execução de uma máquina virtual na AST, este trabalho apresenta uma abordagem mais desafiadora. A ideia é contemplar e dar apoio ao uso de dispositivos IoT periféricos mais simples e com menos recursos, em que a única forma de controlá-los é através de comandos e parâmetros básicos no formato de *bytecode* bruto (i.e. o protocolo nativo). A fim de possibilitar tal forma mais geral e fundamental de atuação, aplicável a qualquer periférico com interface *Bluetooth*, notou-se a necessidade de estender o *middleware* de IoT *ContextNet* [8, 3] com um serviço e um protocolo que suportam essa atuação genérica. Esse serviço é executado parcialmente na nuvem (ou *cluster*) e parcialmente em um *hub* móvel, que interage diretamente com o AST, que possui uma interface *Bluetooth* LE.

3.1 Arquitetura

A extensão proposta para o *middleware ContextNet* suportar o controle de dispositivos/coisas/objetos inteligentes acionáveis é baseada em dois microsserviços, denominados *Smart Objects Manager* (SOM) e *Mobile-Actuator* (M-Act).

O SOM deve ser executado em um nó de processamento do *ContextNet Core*, na nuvem ou *cluster*. o SOM contém um repositório de *drivers* para os dispositivos AST, bem como uma *connectedTable* que associa cada AST descoberto a um único M-Hub, que é o atual *gateway* para controlar este AST. É importante observar que esse M-Hub pode mudar, já que um M-Hub pode

ser executado em um *smartphone Android*. Desta forma, tanto o M-Hub pode ser movido para longe do AST quanto o AST pode mover-se ou ser movido para longe do M-Hub. Após a desconexão entre o AST e o M-Hub, um segundo M-Hub pode encontrar o AST sendo controlado e se conectar a ele. Cada M-Hub pode encontrar e se conectar a muitos dispositivos AST diferentes ao longo do tempo.

O M-Act, por sua vez, é um microsserviço em execucão no M-Hub, ou seja, em qualquer dispositivo baseado em *Android*. O principal objetivo do M-Act é converter o protocolo genérico definido pela API para o protocolo nativo do AST definido pelo fabricante. Para isso, o M-Act precisa solicitar o *driver* do AST encontrado ao SOM e informar ao SOM que o AST está conectado na rede e disponível para atuação.

O processo de conexão com o AST é ilustrado na Figura 3.1.

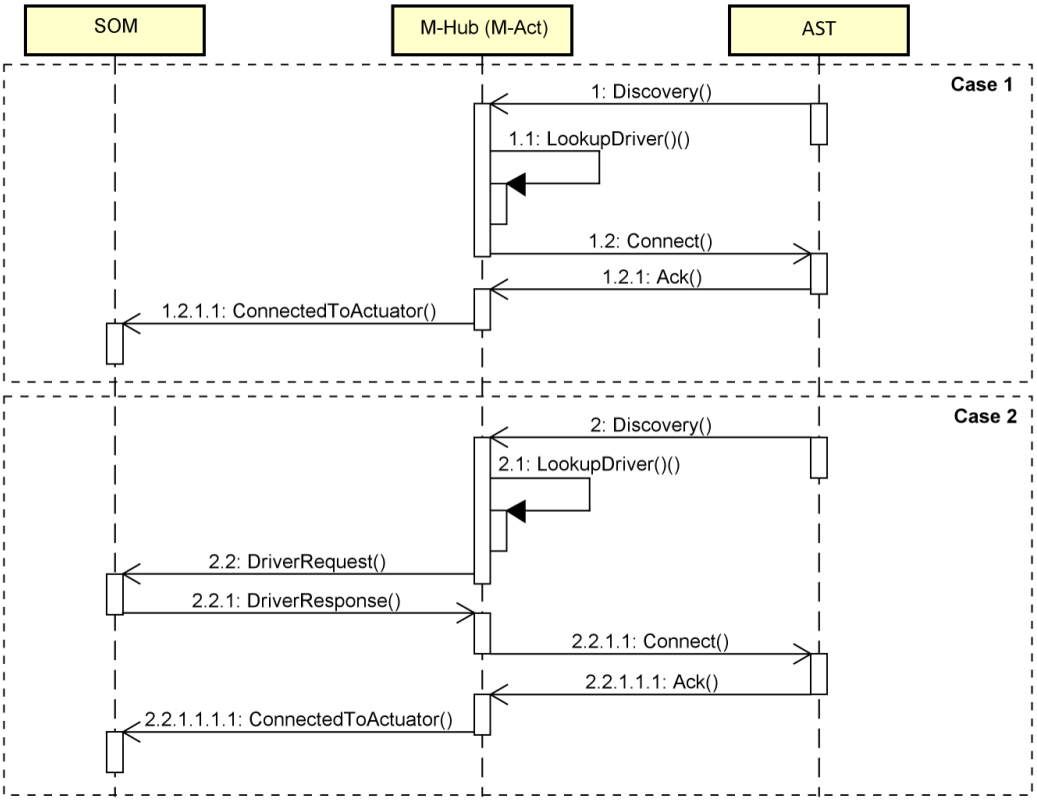

Figura 3.1: Etapas de conexão.

Periodicamente, o M-Hub procura dispositivos próximos e, imediatamente após a descoberta de um novo AST, verifica se existe um *driver* apropriado para o AST em seu *cache* de *drivers* (Figura 3.1 - Caso 1).

Se não encontrar um *driver* adequado em seu *cache*, o M-Act solicita esse *driver* ao SOM (Figura 3.1 - Case 2). Quando o SOM tem o *driver* para o

AST descoberto, ele responde enviando o *driver* solicitado para o M-Act. Caso contrário, ele notifica o M-Act que nenhum *driver* para o AST foi encontrado.

Quando o *driver* é carregado corretamente no M-Act, o M-Act se torna hábil para se conectar ao AST e notificar o SOM a respeito do status dessa nova conexão. Após esta etapa, o M-Act pode receber os comandos de atuação, traduzi-los em comandos nativos com o auxílio do *driver* e enviá-los ao AST.

Para enviar comandos de atuação para um AST, um cliente deve, inicialmente, consultar o SOM para saber qual M-Hub é o *gateway* atual do AST de destino, recebendo uma lista de comandos disponíveis na especificação do *driver* do AST. Depois que o cliente receber a resposta do SOM com o UUID (Identificador Único Universal) do M-Hub associado ao AST e a lista de comandos, ele está pronto para controlar o AST. Esses comandos de atuação genéricos são entregues ao M-Hub, que é o *gateway* atual do dispositivo de destino. Esses comandos são processados pelo M-Act local do M-Hub. O M-act, então, traduz os comandos de atuação genéricos de alto nível em comandos específicos do protocolo nativo, de baixo nível, do AST, de acordo com a especificação do *driver* do AST. Em seguida, o M-Act encaminha a mensagem do protocolo nativo ao serviço S2PA, que envia ao AST utilizando a tecnologia correspondente (atualmente, somente BLE).

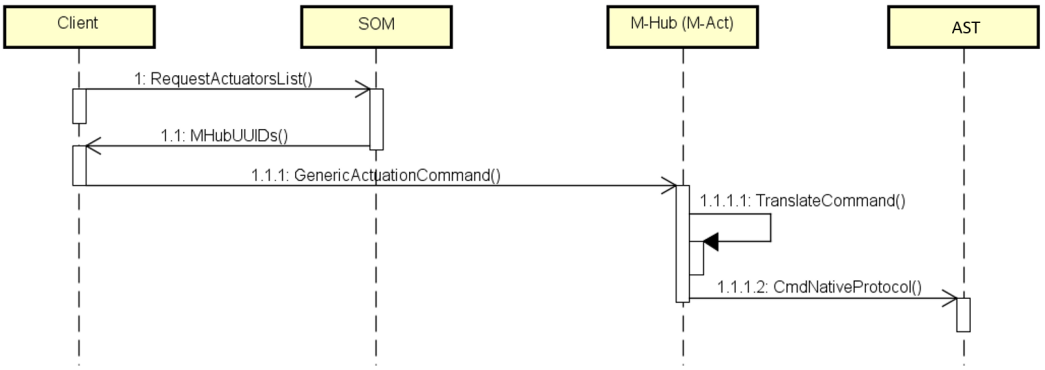

Figura 3.2: Sequências de atuação.

Seria desejável, entretanto, que os comandos de atuação pudessem ser endereçados a um AST de um determinado tipo em uma determinada região geográfica, abstraindo a existëncia do M-Hub como *gateway* do AST. Porém, a implementação atual do *ContextNet* não permite o endereçamento de mensagens para dispositivos IoT, obrigando, assim, o envio de mensagens para o M-Hub ao qual o AST está conectado.

Uma alteração possível no *ContextNet* para viabilizar essa funcionalidade é a inclusão de UUIDs dos dispositivos IoT e as informações de localização e UUID do M-Hub, a qual o dispositivo IoT está conectado, nos *Gateways*

do SDDL. Dessa forma, seria possível identificar os dispositivos IoT e suas respectivas características que estão em uma daterminada região geográfica. Assim, o *Gateway* do SDDL poderia redirecionar as mensagens endereçadas a um determinado dispositivo IoT para o M-Hub em que está conectado.

Todas as mensagens trocadas entre os serviços M-Act e SOM possuem o formato MACTQuery (Figura 3.3).

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

Figura 3.3: MACTQuery.

O campo ***type*** distingue uma mensagem de comando (cmd) de uma mensagem de solicitação de ***driver***. As mensagens de comando podem conter qualquer comando de atuação genérico destinado a um AST. Quando um cliente constrói um comando de atuação, ele deve preencher o campo ***target*** com o nome do AST a ser controlado.

Mensagens do tipo ***driver***, por sua vez, são utilizadas para transportar um *driver* de algum AST do SOM para o M-Act. Podem ser utilizadas, ainda, na solicitação de um *driver* necessário por parte do M-Act.

3.1.1 M-Act

O M-Act é um serviço do M-Hub destinado a gerenciar a comunicação sem fio destinada a controlar um atuador de ASTs. Resumidamente, o M-Act executa duas tarefas: (a) solicitar ao SOM um *driver* para o atuador e (b) traduzir um comando genérico de atuação (recebido de um nó remoto do *ContextNet*) para o protocolo nativo do atuador específico. Como já mencionado, a tradução do comando genérico para o protocolo nativo do atuador é realizada com a ajuda de um *driver*.

Inicialmente os *drivers* de dispositivos foram desenvolvidos no formato JSON [9]. Porém, observou-se que para realizar a tradução de comandos

genéricos para o protocolo nativo de alguns dispositivos, poderia ser necessário executar algumas funções mais complexas, como *checksums* e algoritmos de criptografia. Essa necessidade ficou evidenciada durante o desenvolvimento do projeto quando percebeu-se que o protocolo nativo de alguns dispositivos AST utilizados na pesquisa exigiam que fossem enviados *checksums* dos argumentos dos comandos a serem executados. Com isso, concluiu-se que tal necessidade similar também poderia existir em outros ASTs e, possivelmente, com funções ainda mais complexas do que simplesmente calcular um *checksum*. Por exemplo, algum AST poderia exigir que os comandos e argumentos de uma sequência de comandos fossem criptografados, compactados, convertidos para *little/big endian* ou numerados antes de serem enviados ao AST.

Devido a esses fatores, optou-se por utilizar uma linguagem de *script* que além de permitir ao desenvolvedor criar algoritmos complexos, como os citados anteriormente, também permite isolar a execução do *driver* através de *sanboxing* do interpretador do *script*, dando mais segurança para a aplicação final. Dentre as linguagens de *script* disponíveis, optou-se pela utilização de Lua por ser uma linguagem multiplataforma, enxuta, de fácil implementação e integração [10]. No entanto, nada impede que em implementações futuras outras linguagens de *script* como Javascript ou Python possam ser utilizadas.

Os *drivers* escritos em Lua tem o formato ilustrado na Figura 3.4, onde cada função do *driver* equivale a um comando de alto nível da API utilizada na aplicação.

```lua
function connection()
    local step1 = {}
    step1["service"] = <uuid_service>
    step1["characteristic"] = <uuid_characteristic>
    step1["value"] = <cmd_array>
    return { step1 }
end

local function checksum(cmd)
    local sum = 0
    for i=3, #cmd do
        sum = sum + cmd[i]
    end
    print(sum%256)
    return bit32.bnot(sum % 256) % 256
end

function roll(args)
    local speed = args["speed"]
    local heading = args["heading"]
    local state = args["state"]
    local seq = args["seq"]

    local heading1 = bit32.lshift(heading, 8) % 256
    local heading2 = bit32.band(heading, 0xFF) % 256
    local cmd =  {0xFF,  0xFE, 0x02, 0x30, seq, 0x05, speed,  heading1, heading2, state }
    table.insert(cmd, checksum(cmd))

    local result = {}
    result["service"] = <uuid_service>
    result["characteristic"] = <uuid_characteristic>
    result["value"] = cmd

    return { result }
end
```

Figura 3.4: Exemplo de *driver* em Lua com função *checksum*.

Cada função no *driver* deve receber apenas um argumento (`args`) que é uma tabela Lua com os parâmetros do comando a ser executado (p.ex.: o comando **roll** recebe os parâmetros `speed`, `heading`, `state` e `seq`). As funções do *driver* devem retornar um *array* de tabelas, onde cada tabela contém uma sequência de *bytecotes* que deverão ser enviadas para o AST e, no caso de dispositivos BLE, cada tabela também deve conter os UUIDs do serviço e característica onde a sequência de *bytecodes* deverá ser escrita. As funções dos *drivers* devem retornar várias tabelas pois, um único comando de alto nível na aplicação pode ser traduzido em um conjunto de comandos do protocolo nativo do dispositivo.

Além disso, o *driver* pode conter a função **connection** que não recebe argumentos. Essa função é necessária para dispositivos que precisem de alguma sequência de comandos para desbloquear o dispositivo, ou como forma de autenticação.

O uso de *drivers* permite com que o M-Act controle uma vasta variedade de dispositivos diferentes e oferece a capacidade de estender, sob demanda, sua capacidade de atuação para novos ASTs quando eles são atendidos pela primeira vez. Além disso, a API utilizada na aplicação é definida pelos comandos genéricos definidos pelo conjunto de *drivers* disponíveis, permitindo

com que o M-Act se adapte a diferentes contextos. No entanto, o M-Act também não mantém indefinidamente os *drivers* de todos os dispositivos AST aos quais já esteve conectado. Se algum *driver* baixado não for utilizado por algum tempo, este é descartado pelo M-Act para que o espaço de memória seja liberado, possibilitando o *download* de novos *drivers* necessários para o controle de ASTs recém detectados.

Como microsserviço do M-Hub, o M-Act interage com outros serviços em execução no M-Hub através do *EventBus*(Figura 3.5).

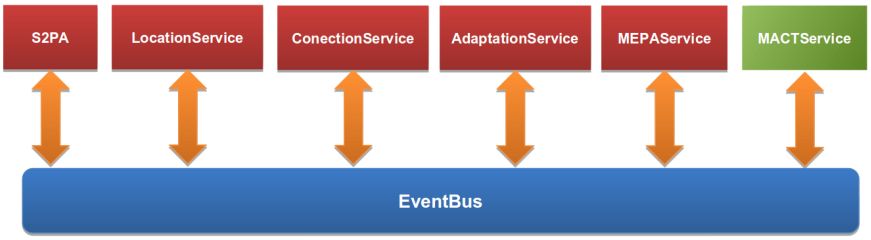

Figura 3.5: Eventbus.

Durante o processo de atuação em ASTs o M-Act se comporta como um elemento central na arquitetura do M-Hub (Figura 3.6) recebendo mensagens MACTQuery do *ConnectionService*, que manipula mensagens de entrada e saída do SDDL *Core*. Após receber a MACTQuery, o M-Act extrai o comando que deseja-se executar e o encaminha para uma instância de *GenericActuator* que implementa a interface *TechnologyDevice* e representa um AST qualquer na forma de um M-OBJ. O *GenericActuator* tem como objetivo realizar o gerenciamento de estado individual de um AST controlado pela aplicação. Por exemplo, um dos dispositivos utilizados exige que um dos parâmetros do comando seja um número de sequência que indica a ordem do comando a ser executado. Além disso, também é de responsabilidade do *GenericActuator* realizar a tradução dos comandos de alto nível para o protocolo nativo do AST com a ajuda do *driver* e enviar os *bytecodes* para o S2PA que, por sua vez, envia os *bytecodes* para o dispositivo físico através da interface WPAN adequada.

Vale ressaltar que dessa forma, um mesmo comando pode ser enviado para um conjunto de ASTs através de uma única MACTQuery.

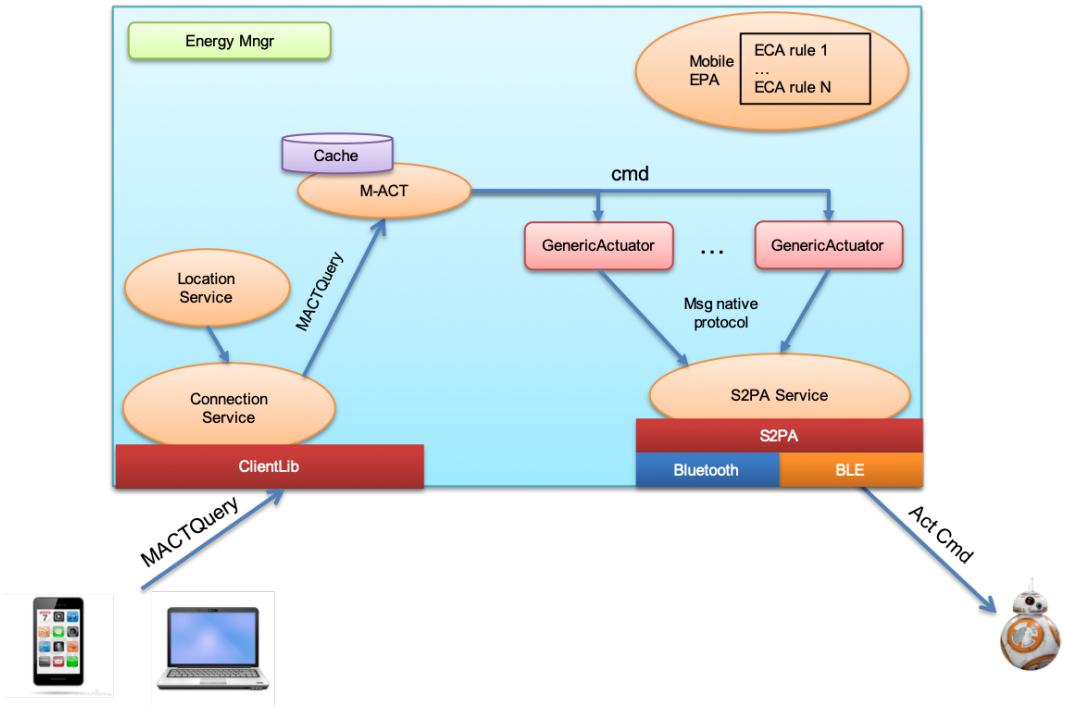

Figura 3.6: Arquitetura do M-Act com interpretador descentralizado.

Com esta proposta de suporte de atuação do *middleware*, existem basicamente duas maneiras de atuação remota em um AST:

1. através de uma aplicação remota operada por um humano. Por exemplo, controle remoto de braço mecânico controlado por alguém usando um *smartphone* ou PC.

2. o acionamento em um AST é iniciado de forma autônoma, assim que alguma regra CEP é executada localmente (no serviço MEPA do M-Hub) ou remotamente (em um nó de processamento do SDDL), pela detecção de algum padrão interessante. Por exemplo, depois que a temperatura local atingir 30 graus, alguns HVAC nos arredores são ativados para diminuir a temperatura do ambiente. Esta aplicação será desenvolvida em trabalhos futuros.

3.1.2 SOM

O *Smart Objects Manager* (SOM) é um microsserviço executado no SDDL (Core do ContextNet) e realiza duas tarefas principais: (a) serve de repositório de *drivers* para sensores e atuadores e (b) e mantém a informação sobre ASTs conectados a cada momento na `connectedTable`.

O repositório é responsável por fornecer *drivers* a pedido do M-Hub, controlar as atualizações dos drivers, além de controlar versões distintas de

um mesmo *driver*. A `connectedTable`, por sua vez, é uma tabela que registra todas as conexões BLE de M-Hubs com dispositivos atuadores.

Como já explicado através da Figura 3.1, todas as vezes em que o M-Hub se conecta a um AST, o M-Hub envia uma mensagem para o SOM, informando-o sobre esta conexão. O SOM, então, utiliza esta mensagem para atualizar a `connectedTable`, inserindo uma tupla [UUID M-Hub, Nome do AST]. Quando o M-Hub perde a conexão com o AST, devido ao amplo movimento entre eles ou um problema de conexão WPAN, o M-Hub envia uma mensagem para informar ao SOM sobre a desconexão. Esta informação é repassada para a `connectedTable`, removendo-se, assim, a tupla [UUID M-Hub, Nome do AST].

O *driver* AST contém vários parâmetros que incluem o ID do dispositivo, a versão do *driver*, o protocolo de comunicação usado pelo dispositivo (por exemplo, BLE, *Bluetooth* clássico), credenciais de autenticação quando aplicável e os comandos de dispositivos compatíveis. Todos esses parâmetros são armazenados e trocados no formato JSON, sendo interpretados pelo M-Act. Os comandos do dispositivo são a coleção de possíveis comandos catalogados para consultas ou acionamentos de dados/estado.

Uma utilidade alternativa para o SOM seria a execução da tradução dos comandos no protocolo nativo do AST e/ou manter os comandos enviados por um cliente, até que o AST que se deseja controlar se conecte a um M-Hub. No entanto, neste trabalho, optou-se por executar a tradução dos comandos no M-Hub para evitar a concentração do processamento na nuvem. Essa escolha permite com que parte do processo seja executado nas bordas, usando, assim, o poder de processamento do dispositivos Android que executam o M-Hub.

3.2 Aplicação

A solução proposta pode ser utilizada como base para aplicações que tenham como principal objetivo o controle de dispositivos de maneira remota. Exemplos de aplicações desse tipo são *SmartHomes*, Indústria 4.0 ou qualquer outra em que um usuário precise acionar um dispositivo à distância. Por exemplo, em uma *SmartHome* o proprietário pode especificar a temperatura desejada enquanto está fora acionando os aparelhos de ar condicionado pela Internet. Em uma Indústria 4.0 o engenheiro responsável pode acionar mecanismos de segurança após receber alertas indicando alguma anormalidade nos processos.

Tipicamente, uma aplicação desenvolvida com base na solução proposta é ilustrada na Figura 3.7, onde o usuário desenvolvedor precisa criar uma

aplicação cliente que troca mensagens com o SDDL através da *ClientLib* fornecida pelo *ContextNet*. O restante da arquitetura da aplicação fica por conta da arquitetura geral do *ContextNet* com o SOM executando no SDDL, na nuvem, e o M-Hub executando em dispositivos Android.

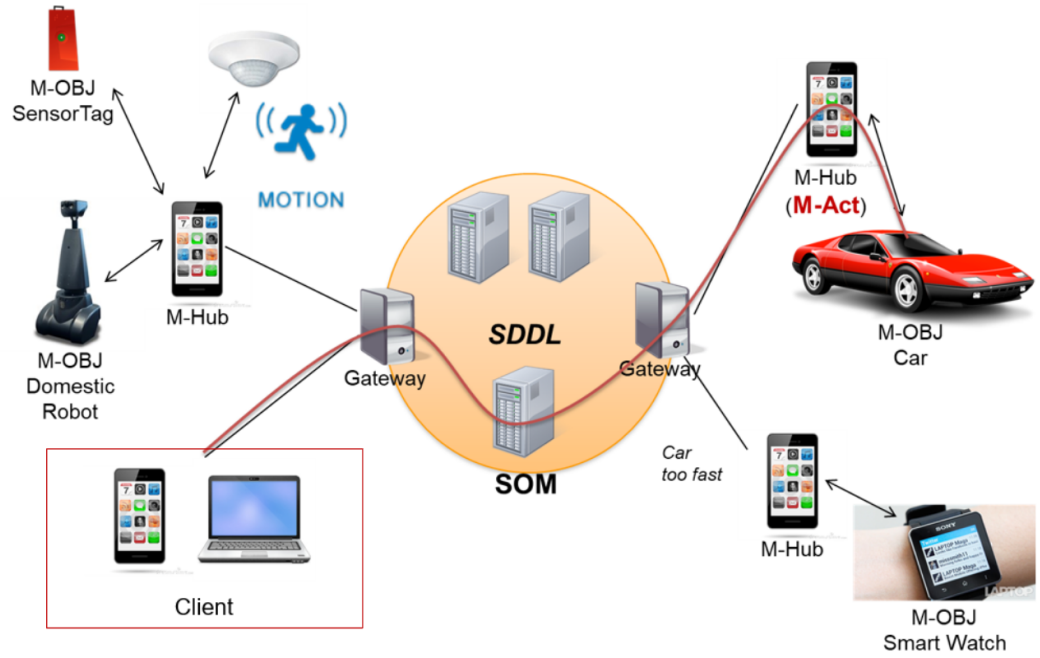

Figura 3.7: Visão geral de uma aplicação.

Durante o desenvolvimento do projeto, os *drivers* dos dispositivos foram desenvolvidos através de engenharia reversa, descobrindo os protocolos nativos de dispositivos. No entanto, nada impede que os *drivers* sejam disponibilizados pelo fabricante dos dispositivos. Nesse caso, entretanto, o desenvolvedor ficaria restrito à API definida pelos fabricantes dos dispositivos. Alternativamente, o desenvolvedor poderia implementar novos *drivers* para traduzir comandos da sua API para os comandos indicados nos *drivers* dos fabricantes.

4
Trabalhos Relacionados

Existem muitas abordagens de *middleware* para IoT. Em geral, o *middleware* pode facilitar um processo de desenvolvimento integrando dispositivos de computação e comunicação heterogêneos e suportando a interoperabilidade dentro dos diversos aplicativos e serviços. Nos trabalhos [11] [12] são apresentadas algumas propostas para *middleware* IoT que vêm sendo estudadas ao longo dos anos.

Alguns destes *middlewares* são utilizados em aplicações específicas como segurança e interoperabilidade [13], aplicações em *Smart Cities* [14] e IoT como serviço [15], enquanto outros fornecem um suporte mais abrangente para desenvolvimento de aplicações IoT. Existem, ainda, abordagens que enfatizam a integração de redes de sensores sem fio (WSNs) à Internet [16].

No entanto, muitos destes *middlewares* não consideram os nós móveis, não consideram objetos inteligentes móveis ou não os dimensionam. Não há relatos na literatura, até então, de uma abordagem sistemática de uma arquitetura de *middleware* redimensionável focado em Internet das Coisas Móveis e que ofereça suporte a um tratamento uniforme de atuação remota sobre dispositivos inteligentes com atuadores.

A utilização de *smartphones* para melhorar o monitoramento e o controle de instalações atraiu a atenção acadêmica devido à vasta possibilidade de aplicações e estudos a serem realizados. Em [17], por exemplo é apresentado um conceito arquitetônico diversificado para o *Wireless Sensor Actuator Mobile Computing in a Smart Home* (WiSAMCinSH). Este estudo consiste na utilização de sensores e atuadores em vários canais de comunicação, com diferentes capacidades e graus de confiabilidade na comunicação. No trabalho, os autores destinam a utilização da aplicação ao usuário final para controle de dispositivos domésticos e monitoramento do ambiente. Assim como esse trabalho, [18], [19] e [20] também exploram a utilização de *smartphones* no contexto de *Smart Home*.

Por outro lado, em [21] o autores apresentam a utilização de *smartphones* para o monitoramento do ambiente interno de máquinas de café, ajustando o sabor do café de acordo com a preferência de um cliente. Essa troca de informações é feita via *Bluetooth*.

No entanto, as propostas estudadas nos trabalhos citados acima apresentam baixo nível de abstração entre o *software* do *smartphone* e o *hardware* a ser detectado e controlado. É necessário que se conheça previamente as especificações protocolo nativo e do *hardware* do dispositivo a ser controlado e, além disso, o *smartphone* precisa coordenar o processamento totalmente, sem fornecer interação remota.

Em [22] é possível observar uma forma mais dinâmica para utilização de *smartphones*. Os autores propuseram uma arquitetura para comunicação entre clientes como *tablets* e *smartphones* e dispositivos IoT, como sensores e atuadores. A arquitetura proposta nesse trabalho sugere três camadas ou blocos: (a) Bloco de comunicação responsável por tratar a recepção e transmissão de mensagens por um meio de comunicação sem fio como BLE, ANT+, IEEE 802.15.4, ZigBee, NFC, Wi-Fi, Z-wave. (b) Bloco de gerenciamento de dispositivos IoT que atua como uma interface para os dispositivos IoT. Esse bloco é capaz de tratar diferentes dispositivos IoT, desde que seja implementada uma interface com o protocolo de alto nível utilizado para cada dispositivo. (c) Bloco gerenciador de coordenação, esse bloco é responsável pela coordenação da interação entre os blocos de comunicação e gerenciamento de dispositivos. Nesse trabalo é possível identificar a necessidade da utilização de um protocolo de alto nível para descrever as funcionalidades dos dispositivos, assim como uma interface capaz de traduzir esse protocolo para o protocolo nativo do dispositivo a ser controlado.

Já em [23], os autores apresentam o conceito de IoT sendo acessado pela *Internet* com o *smartphone* atuando como *gateways*. O objetivo desse trabalho é fornecer um *gateway* móvel e um nó IoT *Device Management* que possa manter uma lista de protocolos de dispositivos e atuar como um *buffer* de dados coletados de sensores. Nessa literatura, nada é mencionado a respeito de atuadores.

Tanto [22] quanto [23] apresentam a necessidade de um intermediário central para gerenciar os dispositivos conectados. Esses trabalhos apresentam, também, que é necessário um repositório central de *drivers* para conectar dispositivos IoT. Além disso, ressaltam a ideia do uso do *smartphone* como um *gateway* que controla o trabalho de interação da IoT.

4.1 Discussão

Os trabalhos apresentados neste capítulo possuem algumas semelhanças com o trabalho apresentado nesta dissertação. Dentre eles, os trabalhos [17], [22] e [23] possuem o maior grau de semelhança. A Tabela 4.1 compara esses

trabalhos ao trabalho apresentado nesta dissertação de forma resumida. Fica evidente a necessidade de um *driver*, ou algum outro mecanismo qualquer que seja capaz de descrever as peculiaridades de cada AST para auxiliar na interoperabilidade dos dispositivos, permitindo que a aplicação desenvolvida seja capaz de se comunicar com os dispositivos independentemente do protocolo nativo definido pelo fabricante do AST.

Além disso, nota-se que, apesar de permitir a inclusão de novos dispositivos na aplicação, apenas [23] implementa um mecanismo de repositório de *drivers*, chamados de configuração de atuador. Entretanto, este repositório fica limitado ao *gateway*, diferente do SOM que mantém o repositório com os *drivers* disponíveis para todos os *gateways* disponíveis.

Embora existam trabalhos semalhantes ao apresentado nesta dissertação, observa-se que nenhum deles permite a inclusão de qualquer AST e a definição de uma API em tempo de execução. Neste trabalho, a utilização de uma linguagem de *script* para a definição dos *drivers* possibilita aos usuários desenvolvedores definir seus próprios conjuntos de comandos de alto nível, ou seja, sua API de atuação, baseado nos problemas que sua aplicação deseja solucionar. Desta forma, uma mesma instância do *middleware* pode ser utilizada por mais de uma aplicação executando simultaneamente, desde que um mesmo AST não possua dois *drivers* diferentes ao mesmo tempo. Neste caso, essa segunda aplicação pode ser desenvolvida enquanto a primeira continua a executar.

Tabela 4.1: Comparação entre os trabalhos relacionados e o trabalho apresentado.

Trabalho	**Interação Remota**	**Drivers de ASTs**	**Respositório de *drivers***	**API**	**Suporta novos tipos de ASTs**
Datta et al. (2014)	✓	Sim, interface não definida	No próprio gateway.	Pré definida	Sim, com *drivers* já conhecidos.
Ghabar et. al. (2015)	✓	Não	Não	Pré definida	Não
Aloi et al. (2017)	✓	Sim, interface própria	Não	Pré definida	Sim, com *drivers* já conhecidos.
M-Act	✓	Sim, interface definida pela aplicação	SOM, no SDDL	API definida pelos comandos disponíveis nos *drivers*	Sim, *drivers* podem ser incluídos e removidos em tempo de execução.

5
Avaliação

Uma aplicação de exemplo foi desenvolvida a fim de verificar a viabilidade do *middleware* descrito no capítulo 3. Os dispositivos utilizados para os testes foram um BB8 da *Sphero*[1], um *Smart RobotCar* da ELEGOO[2] e um dispositivo desenvolvido pela equipe do LAC para acionar um aparelho de ar condicionado. Os dispositivos BB8 e *Smart RoboCar* possuem a capacidade de se movimentar. Isso permite com que o desenvolvedor crie um comando de alto nível (*move*, por exemplo) que acione essa funcionalidade nos dois dispositivos, mesmo eles apresentando caracteristicas e protocolos nativos diferentes.

Além disso, foram avaliadas duas métricas importantes quando trata-se de atuação remota: o tempo gasto para conectar a um atuador, ou seja, quanto tempo é gasto no processo de conexão ilustrado na Figura 3.1, e a latência no envio de comandos para um atuador, ilustrado na Figura 3.2.

A aplicação de exemplo, ilustrada na Figura 5.1, permite com que um usuário registre um dispositivo, faça o *upload* do respectivo *driver* e execute alguns comandos simples. Também é possível implementar *scripts* com sequências de comandos a serem enviadas para o dispositivo AST.

[1]https://www.sphero.com/
[2]https://www.elegoo.com/

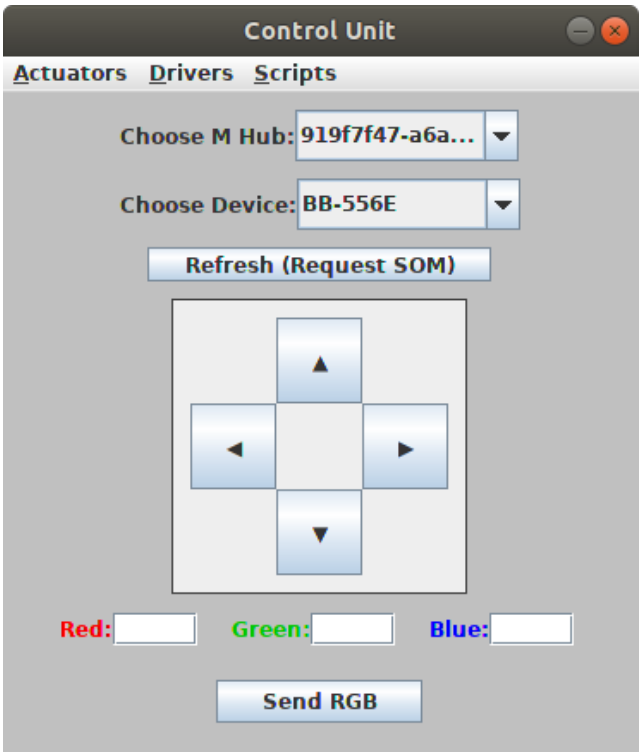

Figura 5.1: Aplicação de exemplo.

Pela capacidade dos ASTs de alterar o ambiente em que estão, não é adequado que qualquer AST possa ser controlado a qualquer momento. Ainda que seja possível a descoberta oportunística de ASTs, e eventualmente dos serviços e funcionalidades, o uso inadequado destes dispositivos poderia por em risco a integridade física de pessoas próximas a ele. Portanto, os ASTs disponíveis precisam ser registrados no SOM. Isto é feito através do menu *Actuators* > *Register*, então o usuário deve preencher o formulário ilustrado na Figura 5.2.

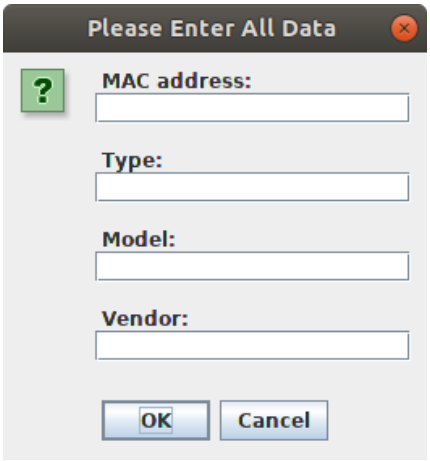

Figura 5.2: Registro de AST.

O identificador utilizado é o MAC *Address* do dispositivo, já os campos *Type*, *Model* e *Vendor* servem como chave composta para identificar o *driver*. Dessa forma, um mesmo *driver* pode ser utilizado para diversos dispositivos do mesmo tipo, modelo e fabricante.

O *upload* de *drivers* é realizado através do menu *Drivers* > *Upload*. Como mencionado no parágrafo anterior, os *drivers* são identificados pela tupla [*Type*, *Model*, *Vendor*]. Então, após selecionar o arquivo do *driver* o usuário deverá preencher um formulário com as informações *Type*, *Model*, *Vendor*, ilustrado na Figura 5.3.

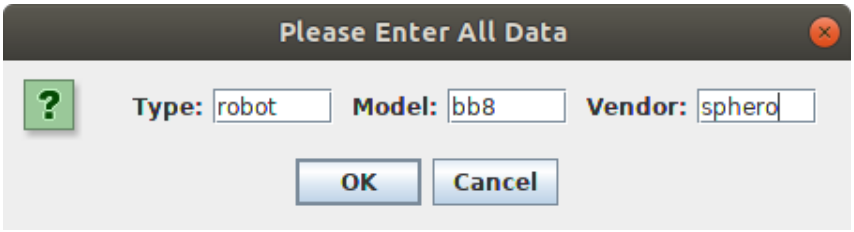

Figura 5.3: *Upload* de *driver* de AST.

A partir desta aplicação, foi possível verificar a viabilidade do *middleware* para o desenvolvimento de aplicações IoT com capacidade de atuação.

O processo mais crítico de conexão a um dispositivo é o caso 2 ilustrado na Figura 3.1, quando o M-Hub identifica a existência de um dispositivo AST na vizinhança, mas precisa fazer a solicitação do *driver* para este dispositivo ao SOM.

Para avaliar este processo, foram criados quatro cenários de testes, onde o SOM foi alimentado com o registro prévio de 1, 100, 1.000 e 10.000 dispositivos hipotéticos diferentes. O objetivo deste teste é identificar como a quantidade de dispositivos registrados no SOM influncia o tempo gasto para que um novo dispositivo, descoberto pelo M-Hub, esteja apto para atuação.

Para cada cenário de testes, foram realizadas 10 execuções. Os testes foram realizados numa intranet, com o M-Hub executando em um dispositivo Xiaomi Mi 9, com plataforma *Android* versão 9, conectado via WiFi na frequência de 5 GHz. O SOM, por sua vez, foi executado em um notebook sob Linux Ubuntu 18.04.2 LTS, com 8 GB de RAM, processador Intel Core i7-4700MQ, conectado via WiFi na frequência de 2,4 GHz.

As figuras 5.4, 5.5, 5.6 e 5.7 exibem os valores de tempo (em milissegundos) obtidos em cada execução.

Figura 5.4: Testes de conexão com 1 dispositivo no SOM.

Figura 5.5: Testes de conexão com 100 dispositivos no SOM.

Figura 5.6: Testes de conexão com 1000 dispositivos no SOM.

Figura 5.7: Testes de conexão com 10000 dispositivos no SOM.

Pode-se observar nos gráficos acima que, com o aumento do número de dispositivos registrados no SOM, houve um pequeno aumento no tempo para que o dispositivo (AST) ficasse pronto para atuação. No entanto, ao observar os gráficos comparando a média dos tempos obtidos na Figura 5.8 e o desvio padrão na Figura 5.9, percebe-se que a variação entre o melhor e o pior caso foi de cerca de 22,5 %, enquanto o desvio padrão dos tempos obtidos sofreu um variação de 57,3 % para os mesmos casos. Isso indica que o aumento no número de dispositivos registrados pode aumentar consideravelmente o tempo de conexão de ASTs ao sistema.

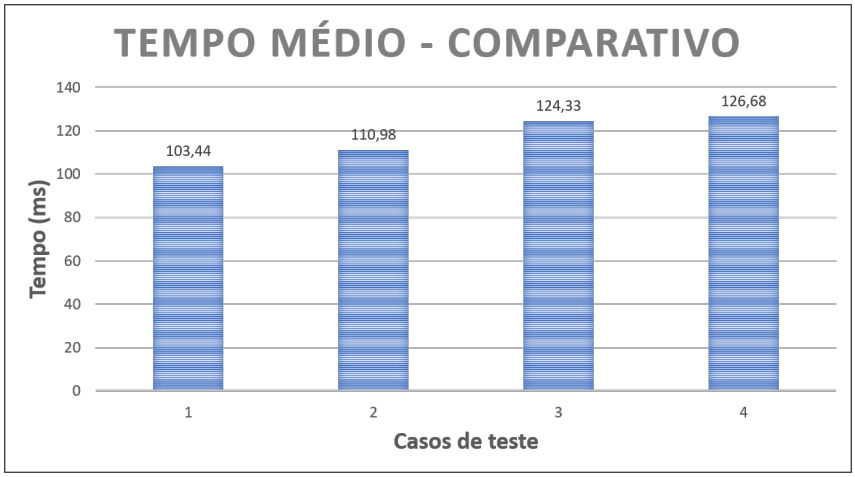

Figura 5.8: Comparativo do tempo médio.

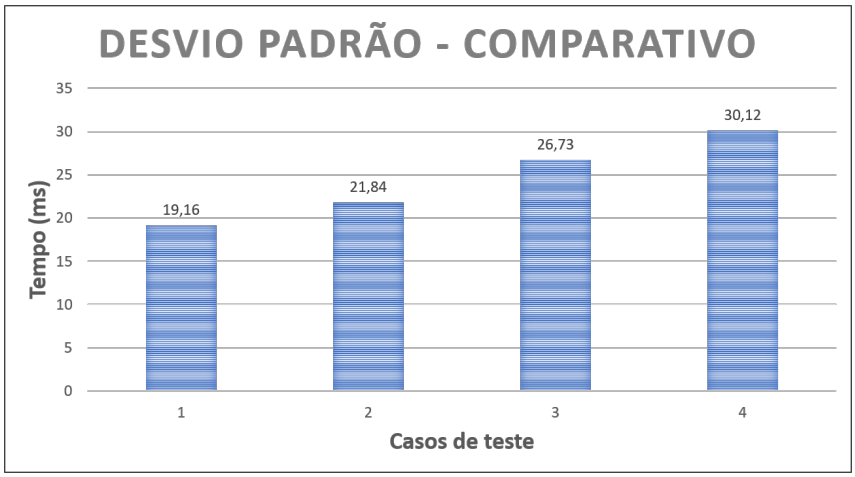

Figura 5.9: Comparativo do desvio padrão.

Para avaliar a latência no envio dos comandos, foi utilizada a aplicação de exemplo para controlar o dispositivo BB8 da *Sphero* através de uma WLAN BLE. A aplicação exemplo foi modificada para marcar o tempo imediatamente antes do envio da MACTQuery e logo após receber uma mensagem de confirmação do envio do comando para o AST. O M-Hub também foi modificado para enviar uma mensagem de confirmação logo após o envio do comando, já convertido para o protocolo nativo, para o BB8. Foram enviados 230 comandos, sendo que cada comando só foi enviado após o recebimento da confirmação do comando anterior. Após o recebimento da confirmação, a aplicação de exemplo calculou a variação do tempo entre o momento do envio do comando e o recebimento da confirmação (ida e volta da mensagem).

A latência, ou o tempo que leva para um comando trafegar do cliente até o dispositivo AST é, aproximadamente, a metade do tempo calculado. Nesse teste foi obtida uma média de 65,47 ms. Contudo, também foi observado um desvio padrão muito alto, cerca de 191.04 ms. No pior caso, o tempo entre o envio do comando e a resposta foi de 3284 ms, ou seja, latência de 1642 ms no envio de um comando.

6
Conclusão

A utilização de atuadores em sistemas IoT tende a se tornar cada vez mais comum, permitindo com que uma grande quantidade de aplicações sejam criadas, como sistemas de *SmartHomes*, *SmartBuildings*, *SmartCities*, Indústria 4.0 e, até mesmo, algumas aplicações inconcebíveis nos dias atuais. Isso trará novos desafios que precisarão ser pesquisados e desenvolvidos, além da necessidade do desenvolvimento de *middlewares* que sirvam como base para essas novas aplicações.

Este trabalho apresentou uma extensão de um *middleware* para suporte à atuação em IoMT, que pode ser utilizado para o desenvolvimento de aplicações que utilizem atuadores IoT, usando como *gateways* dispositivos *Android*. Foi desenvolvida uma aplicação de exemplo que demonstra a viabilidade do *middleware* para o desenvolvimento de aplicações que envolvam atuação sobre dispositivos acionáveis, móveis ou não, possibilitando a especificação de uma API própria da aplicação.

O estudo dos trabalhos relacionados demonstra a dificuldade no tratamento da heterogeneidade dos dispositivos IoT e a falta de um serviço de *middleware* que permita o desenvolvimento de aplicações IoT com suporte a atuação remota e que possam ser estendidas em tempo de execução, adaptando-se de forma a suportar novos dispositivos, não concebidos durante o desenvolvimento da aplicação.

Quanto à avaliação realizada, os resultados foram satisfatórios e mostram que quanto maior o número de dispositivos cadastrados no SOM, maior o tempo que o M-Hub leva para executar a solicitação do *driver*, devido ao maior número de operações realizadas pelo banco de dados para se identificar o dispositivo desejado. No entanto, o maior tempo gasto no processo não é relevante o suficiente a ponto de inviabilizar a utilização da ferramenta.

À respeito dos testes de latência, foi observado um tempo médio de 65,47 ms. No entanto, o desvio padrão se apresentou muito alto. Isto pode ser explicado devido à utilização da faixa de comunicação em 2,4 GHz, utilizada por diversos aparelhos que utilizam comunicação sem fio. Dessa forma, estes aparelhos costumam causar muita interferência na troca de mensagens.

A capacidade de expansão em tempo de execução, atrelada à liberdade

de desenvolvimento de APIs para atuação, que dependem exclusivamente das funções disponíveis nos *drivers* dos dispositivos registrados, torna este trabalho, de certa forma, inovador no contexto de *middlewares* para aplicações IoT.

6.1 Trabalhos futuros

Ao discutir atuação em IoT, diversos aspectos interessantes podem ser observados. Alguns foram tratados neste trabalho. No entanto, existem outros aspectos interessantes que podem ser iniciados a partir deste trabalho.

Um dos aspectos mais importantes quando se trata de atuadores é a segurança. Com a capacidade dos atuadores de alterar o ambiente onde se encontram, é desejável, também, que exista algum tipo de controle mais rígido sobre quais usuários têm permissão para controlar determinados ASTs, a fim de evitar acidentes, invasão de privacidade, etc.

A atuação automática através de regras CEP também pode ser investigada em trabalhos futuros. Através dela seria possível o acionamento de ASTs assim que alguma regra CEP for executada localmente (no microsserviço MEPA do M-Hub), remotamente (em um nó de processamento executando no SDDL) ou nos dois níveis, onde uma regra executada localmente no microsserviço MEPA dispara uma outra regra executada em um nó de processamento no SDDL. Nessas configurações os sensores podem se tornar *triggers* de atuação. Por exemplo, um conjunto de dados coletados de sensores podem indicar o aumento da temperatura e redução da umidade em uma determinada região, indicando um possível incêndio, disparando assim uma sequência de comandos de atuação para paineis de alerta e *drones* de monitoramento/contenção de incêndios. Tudo sem a intervenção humana.

Outro aspecto a ser investigado no futuro é a possibilidade dos próprios ASTs fornecerem seus *drivers*, permitindo com que o M-Hub tenha um papel mais ativo na extensão da aplicação, passando a registrar novos dispositivos automaticamente.

Além disso, também seria interessante o controle coordenado de dispositivos, permitindo com que usuários enviassem sequências de comandos, onde um comando depende da execução bem sucedida de um outro comando. Esses comandos podem, inclusive, serem executados por diferentes ASTs. Por exemplo, na indústria química, em muitos processos, é necessário que o reator seja despressurizado para que a válvula que escoa a produção possa ser aberta. Neste caso, seria interessante que uma sequência de comandos como “despressurizar o reator” e “abrir válvula” seja disparada. No entanto, por questões de

segurança, o segundo comando possui a restrição de só poder ser executado quando o primeiro estiver finalizado.

Um problema importante relacionado ao aspecto de mobilidade é o tratamento de *handover*, onde um AST se desconecta de um M-Hub **A** e se conecta a um M-Hub **B**. Isso pode acontecer tanto pelo movimento do M-Hub **A** se afastando do AST, enquanto o M-Hub **B** se aproxima do AST ou pelo movimento do próprio AST, que pode se movimentar para longe da área de conbertura do M-Hub **A** e entrar na área de conbertura do M-Hub **B**. Em ambos os casos, se houver uma sequência de comandos recebida pelo M-Hub **A**, que ainda não foi enviada ao AST, é necessário que essa sequência de comandos seja repassada ao M-Hub **B**. assim que este se conectar ao AST.

Seria interessante também investigar a possibilidade de usar a abordagem apresentada neste trabalho para o sensoriamento remoto. Nesse caso, a utilização de *drivers* permitiria que novos sensores fossem incluídos na aplicação em tempo de execução, assim como acontece com os atuadores. Os *drivers* seriam utilizados para converter os *bytecodes* lidos dos sensores em informações como temperatura, pressão ou luminosidade em suas respectivas unidades.

7
Referências bibliográficas

[1] TALAVERA, L. E.; ENDLER, M.; VASCONCELOS, I.; VASCONCELOS, R.; CUNHA, M. ; SILVA, F.. **The Mobile Hub concept: Enabling applications for the Internet of Mobile Things**. In: PERVASIVE COMPUTING AND COMMUNICATION WORKSHOPS (PERCOM WORKSHOPS), 2015 IEEE INTERNATIONAL CONFERENCE ON, p. 123–128, 2015.

[2] ENDLER, M.; BAPTISTA, G.; SILVA, L. D.; VASCONCELOS, R.; MALCHER, M.; PANTOJA, V.; PINHEIRO, V. ; VITERBO, J.. **ContextNet: Context Reasoning and Sharing Middleware for Large-scale Pervasive Collaboration and Social Networking**. In: PROCEEDINGS OF THE WORKSHOP ON POSTERS AND DEMOS TRACK - PDT '11, p. 1–2, New York, New York, USA, 2011. ACM Press.

[3] DAVID L., V. R. A. L. A. R. E. M.. **A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes**. Journal of Internet Services and Applications, 4(1):1–15, 2013.

[4] VASCONCELOS, R. O.; TALAVERA, L.; VASCONCELOS, I.; RORIZ, M.; ENDLER, M.; GOMES, B. D. T. P. ; SILVA, F. J. D. S. E.. **An Adaptive Middleware for Opportunistic Mobile Sensing**. In: 2015 INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING IN SENSOR SYSTEMS, p. 1–10. IEEE, 6 2015.

[5] TALAVERA, L.; ENDLER, M.. **An energy-aware iot gateway, with continuous processing of sensor data**. Master's thesis, Departamento de Informática, PUC-Rio, Março 2016.

[6] TANENBAUM, A. S.; BOS, H.. **Modern operating systems**. Pearson, 2015.

[7] SHAKED, U.. **Reverse engineering a bluetooth lightbulb**. https://medium.com/@urish/reverse-engineering-a-bluetooth-lightbulb-56580fcb7546. Acessado em: 11-11-2018.

[8] ENDLER, M.; BAPTISTA, G.; SILVA, L. D.; VASCONCELOS, R.; MALCHER, M.; PANTOJA, V.; PINHEIRO, V. ; VITERBO, J.. **ContextNet: Context Reasoning and Sharing Middleware for Large-scale Pervasive Collaboration and Social Networking**. In: PROCEEDINGS OF THE WORKSHOP ON POSTERS AND DEMOS TRACK, PDT '11, p. 2:1–2:2. ACM, 2011.

[9] VALIM, S.; ZEITUNE, M.; OLIVIERI, B. ; ENDLER, M.. **Middleware Support for Generic Actuation in the Internet of Mobile Things**. Open Journal of Internet Of Things (OJIOT), 4(1):24–34, 2018.

[10] IERUSALIMSCHY, R.; DE FIGUEIREDO, L. H. ; FILHO, W. C.. **Lua—an extensible extension language**. Software: Practice and Experience, 26(6):635–652, 1996.

[11] RAZZAQUE, M. A.; MILOJEVIC-JEVRIC, M.; PALADE, A. ; CLARKE, S.. **Middleware for internet of things: A survey**. IEEE Internet of Things Journal, 3(1):70–95, 2016.

[12] NGU, A. H.; GUTIERREZ, M.; METSIS, V.; NEPAL, S. ; SHENG, Q. Z.. **Iot middleware: A survey on issues and enabling technologies**. IEEE Internet of Things Journal, 4(1):1–20, 2017.

[13] XIAO, G.; GUO, J.; DA XU, L. ; GONG, Z.. **User interoperability with heterogeneous iot devices through transformation.** IEEE Trans. Industrial Informatics, 10(2):1486–1496, 2014.

[14] BELLAVISTA, P.; GIANNELLI, C.; LANZONE, S.; RIBERTO, G.; STEFANELLI, C. ; TORTONESI, M.. **A Middleware Solution for Wireless IoT Applications in Sparse Smart Cities**. Sensors, 17(11), 2017.

[15] AUBONNET, T.; BOUBENDIR, A.; LEMOINE, F. ; SIMONI, N.. **Controlled Components for Internet of Things**. Open Journal of Internet of Things (OJIOT), 2(iii):16–33, 2016.

[16] DELICATO, F.; PAULO, P. ; VASCONCELOS, T.. **Middleware Solutions for the Internet of Things**. Springer Briefs in Computer Science. Springer, 2013.

[17] GHABAR, O.; LU, J.. **Remote Control and Monitoring of Smart Home Facilities via Smartphone with Wi-Fly**. In: PROCEEDINGS OF THE FIFTH INTERNATIONAL CONFERENCE ON ADVANCED

COMMUNICATIONS AND COMPUTATION. INFOCOMP (2015). IARIA, BRUSSELS, BELGIUM, PP. 66-73. ISBN 978?1-61208-416-9, p. 66–73, Brussels, Belgium, 2015. IARIA.

[18] GHABAR, O.; LU, J.. **The Designing and Implementation of a Smart Home System with Wireless Sensor/Actuator and Smartphone**. INFOCOMP 2014, The Fourth International Conference on Advanced Communications and Computation, (c):56–64, 2014.

[19] RAJEEV PIYARE, S. R. L.. **Smart Home-Control and Monitoring System Using Smart Phone**. 24(April):83–86, 2013.

[20] WANG, H.; SABOUNE, J. ; EL SADDIK, A.. **Control your smart home with an autonomously mobile smartphone**. Electronic Proceedings of the 2013 IEEE International Conference on Multimedia and Expo Workshops, ICMEW 2013, 2013.

[21] KIM, K.; PARK, D.-H.; BANG, H.; HONG, G. ; JIN, S.-I.. **Smart coffee vending machine using sensor and actuator networks**. In: 2014 IEEE INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS (ICCE), p. 71–72. IEEE, 1 2014.

[22] ALOI, G.; CALICIURI, G.; FORTINO, G.; GRAVINA, R.; PACE, P.; RUSSO, W. ; SAVAGLIO, C.. **Enabling IoT interoperability through opportunistic smartphone-based mobile gateways**. Journal of Network and Computer Applications, 81(October):74–84, 3 2017.

[23] DATTA, S. K.; BONNET, C. ; NIKAEIN, N.. **An IoT gateway centric architecture to provide novel M2M services**. 2014 IEEE World Forum on Internet of Things, WF-IoT 2014, p. 514–519, 2014.