Alysson Cirilo Silva

Descoberta e Desconexão de Objetos Inteligentes (*Smart Objects*) em Ambientes Oportunísticos de IoMT

São Luís – MA

2019

Alysson Cirilo Silva

Descoberta e Desconexão de Objetos Inteligentes (*Smart Objects*) em Ambientes Oportunísticos de IoMT

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal do Maranhão

Orientador: Francisco José da Silva e Silva

São Luís – MA

2019

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Silva, Alysson Cirilo.
Descoberta e desconexão de objetos inteligentes smart objects em ambientes oportunísticos de IoMT / Alysson Cirilo Silva. - 2019.
40 f.

Orientador(a): Francisco José da Silva e Silva.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2019.

1. IoMT. 2. IoT. 3. Middleware. I. Silva, Francisco José da Silva e. II. Título.

Alysson Cirilo Silva

Descoberta e Desconexão de Objetos Inteligentes (*Smart Objects*) em Ambientes Oportunísticos de IoMT

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. São Luís – MA, 19 de Julho de 2019:

Francisco José da Silva e Silva
Doutor - UFMA

Carlos de Salles Soares Neto
Doutor - UFMA

Marcelo Henrique Monier Alves Júnior
Mestre - IFMA

São Luís – MA
2019

Agradecimentos

Em primeiro lugar, à Deus, pelo dom da vida. Ele conhece bem as minhas limitações e me permitiu superá-las em diversos momentos da minha trajetória até aqui.

Agradeço a Universidade Federal do Maranhão (UFMA) e a todos seus professores e colaboradores por me proporcionarem a oportunidade de uma boa educação no curso que faz parte de mim.

Ao meu orientador, Prof. Francisco Silva, pela confiança, disponibilidade, paciência e a tudo que me ensinou. Também pela oportunidade de participar de projetos tão interessantes e desafiadores. Será sempre um exemplo que vou seguir.

À minha família pelo apoio, amor e compreensão incondicional que me deram nesta fase tão importante. Se não fosse por ela não estaria onde estou agora, são a parte mais importante da minha vida e devo tudo a vocês.

À Caroline, minha namorada, por ser um farol que me guia no mar turbulento da vida. Obrigado por ser uma pessoa tão especial em minha vida, pela compreensão e todos os momentos bons que me deu, não teria conseguido sem seu apoio. É um imenso prazer para mim dividir um planeta e uma época com você.

Aos membros da banca examinadora, pela dedicação e por aceitarem a missão de avaliar este trabalho.

À todos meus colegas do Laboratório de Sistemas Distribuídos Inteligentes (LSDi) da UFMA, por terem compartilhado comigo tantas horas do dia e ter feito esta experiência muito menos penosa. Os considero como uma família.

À todos os amigos que encontrei durante essa jornada, nada disso teria valido a pena se não estivessem estado aqui para compartilhar comigo todos os desafios e alegrias. Obrigado.

The road to wisdom? Well, it's plain
And simple to express:
Err
and err
and err again,
but less
and less
and less.
(Piet Hein)

Resumo

À medida que a IoT se expande no dia a dia e em diversos setores da economia, e dado o caráter oportunístico das interações em ambientes de IoMT, uma extensão da IoT, torna-se claro a importância do mecanismo de detecção de eventos com *smart objects*. Em particular, eventos de descoberta, conexão e desconexão com objetos inteligentes se fazem essenciais para uma quantidade considerável de aplicações que interagem com esses dispositivos sem leitura de dados de sensores (*e.g.*, localização *indoor*). Este trabalho visa apresentar um módulo de notificação de eventos de descoberta, conexão e desconexão de objetos inteligentes no middleware M-Hub/CDDL.

Palavras-chave: IoT. IoMT. Middleware.

Abstract

As the IoT expands itself in daily life and in various sectors of economy, and given the opportunistic character of interactions in IoMT environments, an IoT extension, it becomes clear the importance of a smart object event detection mechanism. In particular, discovery, connection and disconnection events with smart objects are essential to a considerable number of applications that interact with such devices without the need for sensor data reads (*e.g.*, indoor location). This work aims to present a discovery, connection and disconnection event notification module in the M-Hub/CDDL middleware.

Keywords: IoT. IoMT. Middleware.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 IoT

A Internet das Coisas (*Internet of Things* – IoT) é um paradigma de comunicação recente, onde objetos do dia a dia são equipados com equipamentos e protocolos que permitem que se comuniquem com seus usuários e com outros objetos, se tornando parte integral da Internet (ATZORI; IERA; MORABITO, 2010). Pode ser definida como a interconexão de sensores e atuadores que fornece a capacidade de compartilhar informações entre plataformas através de um framework unificado (GUBBI et al., 2013).

No contexto da IoT, as "coisas" (do inglês: *things*) são denominados "*smart objects*" (BANDYOPADHYAY; SEN, 2011). Seu conceito propõe tornar a Internet ainda mais pervasiva e imersiva, ela promoverá o desenvolvimento de aplicações que utilizarão a grande quantidade e variedade de dados produzidos por esses objetos, de forma a prover serviços para os usuários.

No âmbito desse trabalho, é necessário definir o conceito de middleware, componente essencial para aplicações de IoT.

> O middleware é uma camada de software ou conjunto de subcamadas interposta entre os níveis tecnológicos e de aplicação. Sua característica de esconder os detalhes de diferentes tecnologias é fundamental para isentar o programador de problemas que não são diretamente pertinentes para seu foco, que é o desenvolvimento da aplicação específica habilitada pelas infraestruturas de IoT (ATZORI; IERA; MORABITO, 2010, tradução nossa).

A IoT geralmente lida com *smart objects* estáticos, frequentemente presentes na infraestrutura do ambiente, como: sensores de uma sala, leitores RFID de prédios inteligentes, etc. A Internet das Coisas Móveis (*Internet of Mobile Things* – IoMT) é uma extensão da IoT clássica, onde os objetos e os gateways são livres para se locomover, gerando uma maior dinamicidade de interações.

Exemplos de *smart objects* móveis incluem: dispositivos vestíveis, veículos, e robôs móveis. Visto o contexto de mobilidade, smartphones são dispositivos adequados para assumir o papel de fornecedor de Internet e serviço de localização—ou seja, um gateway—para *smart objects* que tenham disponíveis apenas tecnologias de comunicação sem fio de curto alcance e, dessa forma, não implementam a pilha TCP/IP (TALAVERA et al., 2015).

De acordo com Nahrstedt et al. (2016), a IoMT se diferencia da IoT nos seguintes aspectos:

a) *contexto*, exemplo:

   – onde e com quem o dispositivo móvel se encontra.

b) *acesso à Internet e conectividade*, exemplo:

   – estado de conexão (conectado/desconectado);
   – se conectado, em qual rede.

c) *disponibilidade de energia*, exemplo:

   – onde o dispositivo pode ser recarregado;
   – quanta energia o aplicativo necessita.

d) *segurança e privacidade*, exemplo:

   – que tipo de infraestrutura de segurança o dispositivo encontra ao mudar de localização.

1.2 M-Hub/CDDL

O Mobile Hub/Context Data Distribution Layer (M-Hub/CDDL) é um middleware IoT para aquisição, processamento e distribuição de dados de contexto, com amplo suporte para criação de aplicações de IoT cientes de contexto que possuam requisitos de qualidade de contexto (GOMES et al., 2017).

Surgido de uma parceria entre o Laboratório de Sistemas Distribuídos Inteligentes (LSDi) da UFMA e o Laboratory for Advanced Collaboration (LAC) da PUC-Rio, o middleware combina o M-Hub (TALAVERA et al., 2015) como gateway móvel para aquisição dos dados de sensores, internos e externos, com o CDDL, uma camada de distribuição responsável por registrar e descobrir serviços de contexto disponíveis, prover e monitorar dados de contexto e garantir a qualidade dos serviços de distribuição dos dados de contexto.

O M-Hub/CDDL adota o MQTT como único protocolo de comunicação, possuindo inclusive um microbroker interno. Assim, a captura dos dados de sensores obedece a arquitetura *publish/subscribe*.

1.3 Caracterização do problema

Dado a dinamicidade característica da IoMT, onde a topologia da rede muda constantemente e os *smart objects* e gateways interagem de forma oportunística, torna-se claro a necessidade da existência de um mecanismo de notificação que informe à aplicação a ocorrência de eventos de descoberta, conexão e desconexão de *smart objects* com os smartphones.

Um problema semelhante, entretanto mais desafiador, é a construção de tal mecanismo de forma que a aplicação seja notificada sobre os eventos que ocorrem no âmbito de outras aplicações que estão sendo executadas em outros dispositivos.

1.4 Objetivos

O trabalho terá como objetivo a proposta de um modelo do mecanismo de propagação de eventos de descoberta, conexão e desconexão de *smart objects* no middleware M-Hub/CDDL. A solução deverá se adequar aos requisitos de ambientes de IoMT.

1.4.1 Objetivos específicos

a) projetar a arquitetura do mecanismo de notificação;

b) implementar a solução no middleware M-Hub/CDDL;

c) realizar análises de performance e acurácia na solução proposta.

1.5 Organização do texto

O restante deste trabalho é estruturado da seguinte maneira: No Capítulo 2 será apresentada a fundamentação teórica das tecnologias habilitadoras do trabalho. No Capítulo 3 a proposta do trabalho é definida. A avaliação da solução proposta é assunto do Capítulo 4. As conclusões estão presentes no Capítulo 5.

2 Fundamentação teórica

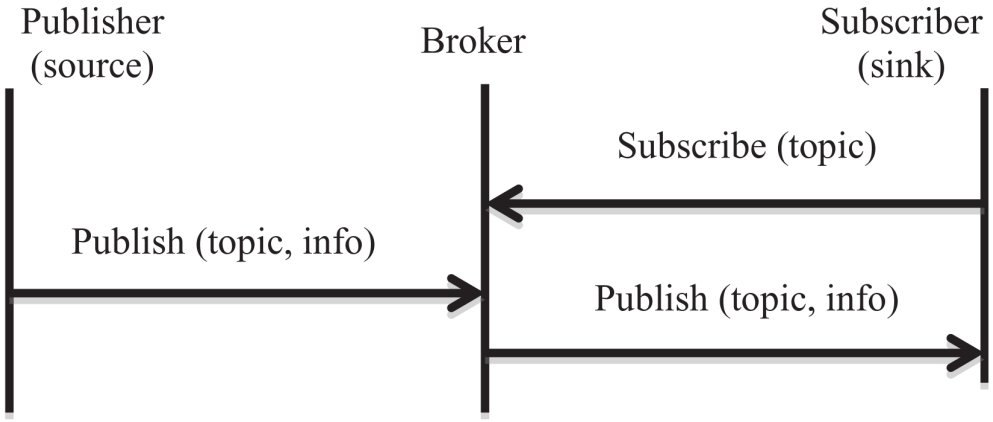
Este capítulo visa realizar uma abordagem de toda a fundamentação teórica e tecnológica necessária para o entendimento da solução proposta neste trabalho.

2.1 MQTT

O *Message Queue Telemetry Transport* (MQTT)[1] é um protocolo baseado em mensagens e seguindo o modelo *publisher/subscriber* , utiliza o conceito de tópicos gerenciados por componentes chamados de *brokers*. Originalmente projetado para o uso em redes não confiáveis e com recursos limitados, consiste em um servidor *broker* e dois tipos distintos de clientes, *publishers* e *subscribers* (LEE et al., 2013).

O *broker* funciona como um intermediador entre as mensagens que são enviadas por *publishers* e recebidas por *subscribers*. Toda a comunicação é feita por intermédio de tópicos, que funcionam como uma hierarquia de diretórios onde as mensagens são entregues. Os *publishers* publicam mensagens no *broker* em um determinado tópico, e este se encarrega de realizar a entrega da mensagem para os *subscribers* que registraram interesse naquele tópico, fazendo com que a orquestração da comunicação entre as entidades produtoras e consumidoras de dados seja feita exclusivamente pelo *broker*. O processo de entrega de mensagens pode ser visto na Figura 1.

Figura 1 – Diagrama de sequência do processo de entrega de mensagens no MQTT



Fonte: Al-Fuqaha et al. (2015)

Este protocolo permite o desacoplamento total entre os produtores e consumidores de dados, deste modo nenhum dos dois componentes têm ciência da existência mútua entre si.

[1] <http://mqtt.org>

2.1.1 Tópicos MQTT

Existem dois elementos em uma mensagem MQTT—o dado e o tópico. O dado é a informação que o *publisher* precisa enviar, este dado será então enviado para o *subscriber* que assinou o tópico onde a mensagem foi publicada (TANTITHARANUKUL et al., 2017).

O *publisher* pode definir qualquer tópico para enviar uma mensagem, com um ou mais níveis de hierarquia, cada nível é separado por uma barra (e.g., `thailand/humidity` ou `thailand/bangkok/traffic`).

Também existe a conveniência da utilização de caracteres coringas para a assinatura de tópicos, como por exemplo `thailand/+/traffic` onde o caractere "`+`" corresponde a qualquer padrão de um único nível de uma hierarquia, neste caso terceiro nível, podendo ser utilizado para receber os dados de tráfego de qualquer cidade da Tailândia. Outro caractere coringa que pode ser utilizado é o "`#`" que corresponde a todos os níveis subsequentes de uma hierarquia, este só pode ser utilizado como o último caractere de uma subscrição (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

Dado o seguinte tópico `a/b/c/d`, as seguintes assinaturas irão receber os dados publicados nele (LIGHT, 2019):

a) `a/b/c/d`;

b) `#`;

c) `a/#`;

d) `a/b/#`;

e) `a/b/c/#`;

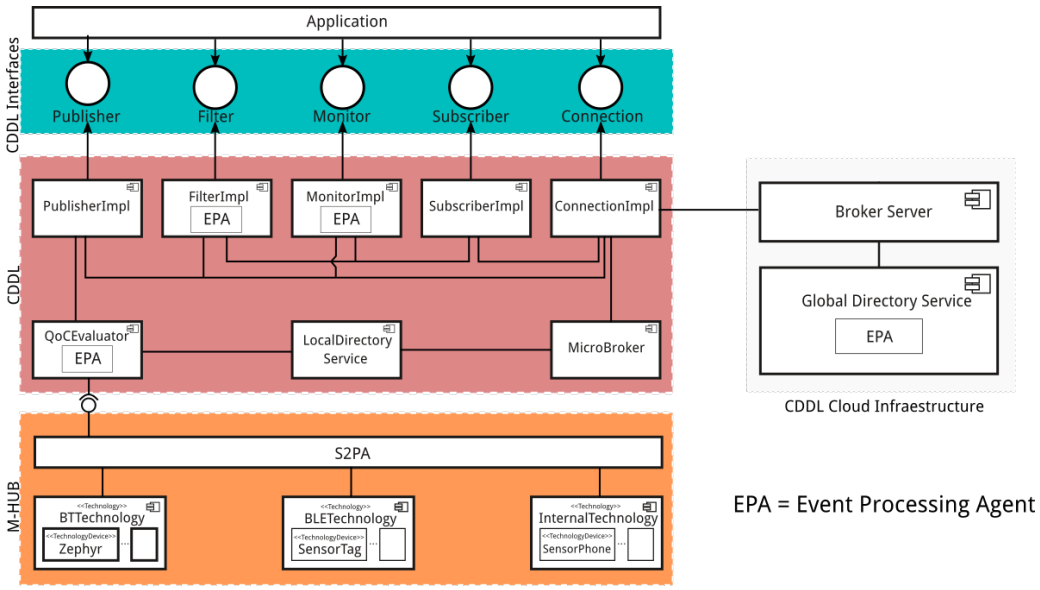
f) `+/b/c/#`.

O MQTT é o protocolo padrão utilizado pelo CDDL para realizar troca de mensagens entre diferentes componentes, como será discutido na subseção 2.2.2.

2.2 M-Hub/CDDL

O M-Hub/CDDL é uma composição de um gateway (M-Hub) e um middleware de IoMT (CDDL). Enquanto o M-Hub transforma o dispositivo Android que está em execução em um gateway IoT móvel responsável pela descoberta e aquisição de dados diretamente dos *smart objects*, o CDDL funciona como middleware provendo serviços locais e remotos de descoberta de provedores de serviços, processamento de eventos complexos, publicação e assinatura de dados e eventos com qualidade de serviço. A Figura 2 apresenta a arquitetura do middleware.

Figura 2 – Arquitetura do M-Hub/CDDL



Fonte: Gomes et al. (2017, p. 15)

O M-Hub faz o papel de gateway móvel entre os *smart objects* e a Internet, onde os componentes mais inferiores encapsulam a heterogeneidade e especificidade das tecnologias WPAN suportadas. O componente `BTTechnology` é voltado para a comunicação com dispositivos que utilizam a tecnologia *Bluetooth*; a comunicação com *smart objects* que utilizam *Bluetooth Low Energy* é feita pelo componente `BLETechnology` enquanto a comunicação com sensores internos do dispositivo móvel é realizada por `InternalTechnology`.

O S2PA é o serviço do M-Hub que gerencia e oferece uma API única para comunicação com as tecnologias suportadas. Os dados coletados pelo S2PA são pré-processados e enriquecidos com metadados de Qualidade de Informação (*Quality of Information* – QoI), e então encaminhados ao CDDL para distribuição.

O `QoCEvaluator` é o componente do CDDL que recebe os dados adquiridos pelo M-Hub. Uma de suas responsabilidades é calcular dinamicamente o valor de alguns parâmetros de QoI que não puderam ser fornecidos na etapa de aquisição. Após esta avaliação, esses parâmetros são adicionados como metadados aos dados de contexto que são encaminhados automaticamente ao `PublisherImpl`.

Todos os dados—oriundos do S2PA ou providos pela camada de aplicação—são então publicados pelo `PublisherImpl` em uma estrutura de tópicos de um *broker* MQTT por intermédio do `ConnectionImpl`, componente que gerencia conexões e sessões das aplicações clientes com *brokers*. Desta forma os dados podem ser entregue às aplicações que registrarem, neste *broker*, interesse em tais dados.

Também nota-se a presença de um componente chamado `MicroBroker`, este é uma

versão reduzida do *broker* MQTT Moquette voltado para dispositivos Android[2]. Pode ser utilizado como alternativa a um *broker* externo. Desta forma, os dados coletados ficam disponíveis apenas entre aplicações que executem no mesmo dispositivo.

2.2.1 M-Hub

O middleware M-Hub pode ser definido, de acordo com Talavera et al. (2015), como um serviço de middleware de IoMT geral executado em um dispositivo móvel pessoal, responsável por descobrir e oportunisticamente conectar à uma miríade de *smart objects* acessíveis apenas através de tecnologias WPAN de curto alcance. Por estar envolvido com cenários de IoMT, este componente de software tem que lidar com situações que apresentam muito mais indeterminismo, devido à fatores como a menor garantia de disponibilidade de sensores e atuadores, confiabilidade reduzida, maior volatilidade em conexões, etc.

O smartphone executando uma instância do M-Hub, funciona como gateway para *smart objects*, fornecendo acesso à Internet para dispositivos que não podem se conectar. Outro recurso importante que pode ser explorado pelas aplicações é a habilidade de enriquecer os dados de sensores com dados de contexto obtidos dos sensores internos do M-Hub.

Para que o M-Hub tenha suporte à determinada tecnologia de comunicação, é necessário que um novo módulo seja implementado. Este módulo será responsável por gerenciar quaisquer operações que sejam necessárias para garantir o funcionamento desta tecnologia.

2.2.1.1 S2PA

Para gerenciar a descoberta e conexão com dispositivos que trabalham com diferentes tecnologias de comunicação, além dos sensores internos do smartphone, o M-Hub utiliza o *Short-range Sensing, Presence & Actuation* (S2PA), um protocolo que fornece uma API comum para realizar a comunicação com diferentes tecnologias WPAN.

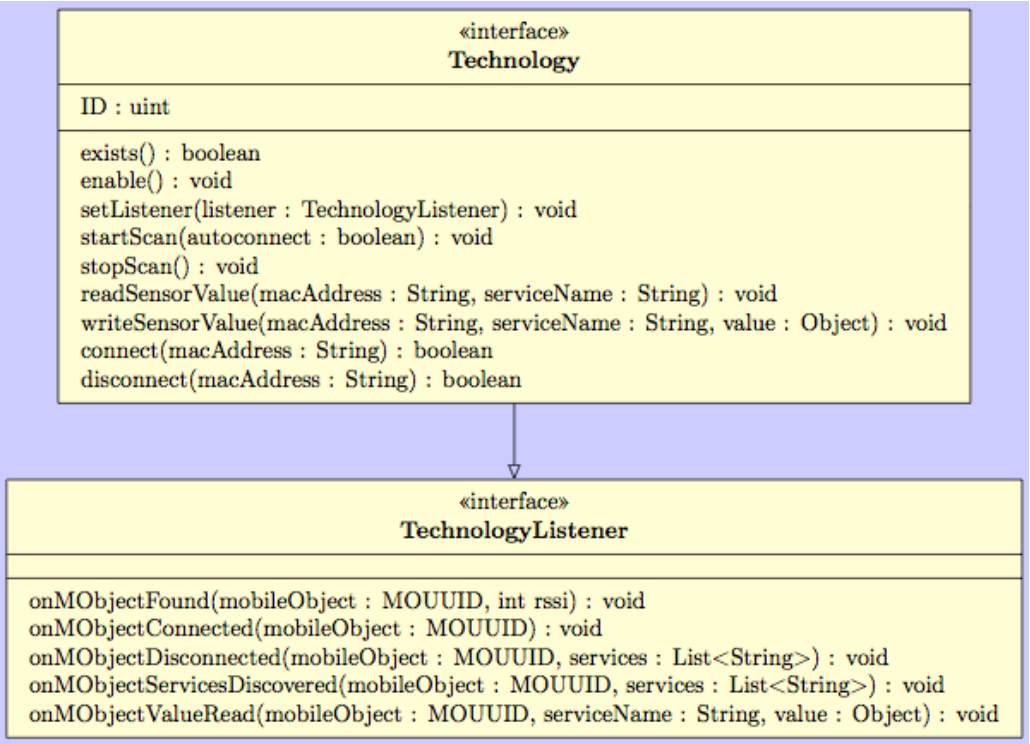
Implementado como um módulo na arquitetura do middleware, o S2PA define um conjunto de métodos e interfaces que os módulos responsáveis por determinada tecnologia de comunicação devem implementar. Isto permite que o S2PA se comunique com todas as tecnologias WPAN suportadas, gerenciando-as, e assim fornece uma API unificada para todas as camadas superiores da arquitetura que precisam se comunicar com tais tecnologias.

É possível encontrar na Figura 3 a interface `Technology` definida pelo S2PA que declara os métodos padrões que realizam as funcionalidades básicas, a qual todas as

[2] <https://github.com/technocreatives/moquette>

tecnologias devem suportar.

Figura 3 – Interface `Technology`



Fonte: Talavera et al. (2015, p. 125)

A interface `Technology` inclui inicialmente um identificador numérico `ID`, definido em tempo de programação, este define unicamente a tecnologia. Dentre os métodos métodos definidos pela interface, destaca-se:

a) `exists()`: Informa se o suporte à determinada tecnologia existe no dispositivo móvel onde o middleware está executando;

b) `enable()`: Habilita o uso da tecnologia para a aplicação;

c) `startScan()`: Inicia o processo de escaneamento desta tecnologia. Recebe como parâmetro um valor booleano que indica se o componente responsável pela tecnologia deve tentar realizar o processo de conexão automaticamente ao encontrar um dispositivo durante o escaneamento;

d) `stopScan()`: Finaliza o processo de escaneamento desta tecnologia;

e) `connect()`: Recebe como parâmetro o endereço MAC de um *smart object*, e tenta realizar conexão com ele;

f) `disconnect()`: Recebe um endereço MAC de um *smart object* como parâmetro, se conectado à ele, realiza a desconexão;

g) `readSensorValue()`: Recebe como parâmetro o endereço MAC e o nome de um serviço fornecido por um *smart object* e realiza o processo de leitura de dados fornecido por aquele serviço.

Outro método importante é o `setListener()`, que recebe como parâmetro uma instância de uma classe que implemente a interface `TechnologyListener`, também apresentada na Figura 3. Todas as informações relevantes sobre *smart objects* descobertos pelas tecnologias geram eventos, estes são capturados através de `TechnologyListener`. Esta interface define métodos que serão acionados de acordo com o acontecimento de certos eventos por parte de alguma tecnologia de comunicação. Dentre os métodos, tem-se:

a) `onMobjectFound()`: Executado quando uma tecnologia encontra um *smart object*;

b) `onMobjectConnected()`: Executado quando uma tecnologia se conecta à um *smart object*;

c) `onMobjectDisconnected()`: Executado quando uma tecnologia se desconecta à um *smart object*;

d) `onMobjectValueRead()`: Executado quando uma tecnologia realiza uma leitura por parte de um *smart object*;

O S2PA é uma classe que implementa a interface `TechnologyListener`. Ela é responsável por iniciar cada tecnologia e se cadastrar através do método `setListener()` como o *listener* daquela tecnologia. As tecnologias se comunicam com o S2PA através dos métodos de `TechnologyListener` descritos acima, desta forma o S2PA recebe todas a interações com os *smart objects*, implementando assim o padrão de projeto *observer* (GAMMA et al., 1994).

Quando o S2PA recebe de alguma tecnologia, um evento de descoberta, conexão, desconexão ou leitura de dados de *smart objects*, ele o encapsula em um objeto do tipo `SensorData`. Esta classe possui os seguintes atributos:

a) `mouuid`: Uma combinação entre o *id* da tecnologia que gerou tal evento com o endereço MAC do *smart object*;

b) `signal`: Número em ponto flutuante onde o RSSI (*Receiver Signal Strength*) daquela interação com o *smart object* é armazenado. O RSSI representa a intensidade de sinal naquele momento;

c) `action`: Indica que tipo de evento uma instância de `SensorData` está encapsulando. Os valores possíveis são as constantes:

- `FOUND`: Indica que um *smart object* foi descoberto pelo M-Hub no ambiente;
- `CONNECTED`: Indica que o M-Hub realizou uma conexão com o *smart object*;

  - `READ`: Indica que uma leitura foi realizada, ou seja, o *smart object* enviou dados ao M-Hub;
  - `DISCONNECTED`: Indica que a conexão com o *smart object* foi desfeita.

d) `sensorName`: Representa o nome do *smart object*;

e) `sensorValue`: Um vetor de pontos flutuantes que armazena os valores lidos do *smart object*. Logicamente este atributo só possui valores quando o atributo `action` assume o valor `READ`.

Os objetos do tipo `SensorData` gerados no S2PA a partir de leitura de dados de *smart objects* são enviados para o CDDL para distribuição.

O M-Hub fornece diversos serviços que não serão tratados neste trabalho, dentre eles, destaca-se (GOMES, 2017):

a) *protocolo de transcodificação*: Os pacotes de dados recebidos dos sensores podem ter diferentes formatos e codificações. Assim, o M-Hub deve transcodificá-los e serializá-los, antes de transmiti-los. A transcodificação de dados é altamente dependente do tipo, marca e fabricante do sensor;

b) *caching de dados*: A fim de otimizar a transmissão para a nuvem através da Internet móvel, o M-Hub pode agrupar várias amostras de dados obtidas a partir de vários sensores próximos antes de enviá-las ao gateway em rajada única. Para isso, o M-Hub armazena em cache as amostras de dados recebidas dos sensores;

c) *configuração e controle dos sensores*: Dependendo do tipo de sensor, o M-Hub pode, eventualmente ou periodicamente, enviar comandos, definições de parâmetros ou requisições de consultas de dados através da WPAN para os sensores conectados;

d) *pré-processamento de dados do sensor*: Antes de enviar os dados, pode ser necessário aplicar uma função de pré-processamento (e.g. transcodificação, formatação, agregação, filtragem, ou comparação com leituras anteriores etc.). Esse pré-processamento é feito no M-Hub;

e) *carregamento dinâmico de módulos do sensor*: Uma vez que não é possível ter módulos internos para todos os sensores que podem estar disponíveis a medida que o gateway se move, o M-Hub oferece um mecanismo de implantação de módulo em tempo de execução e gerenciamento do ciclo de vida desses módulos;

f) *processamento nas pontas*: O M-Hub oferece mecanismos que permitem aos desenvolvedores de aplicações distribuírem as funcionalidades do seu código entre o dispositivo móvel e nodos da nuvem, movendo parte do processamento das informações de contexto para as pontas do sistema. A isso se dá o nome

de In-Network Processing, uma técnica que ajuda a diminuir a quantidade de informações a ser transmitida para a nuvem, podendo melhorar a escalabilidade do sistema. O processamento ao qual esta funcionalidade se refere pode ser implementado em código Java convencional ou utilizando EPL Esper;

g) *processamento ciente de energia*: Por meio de um componente de gerenciamento de energia, o M-Hub monitora o nível da bateria do dispositivo móvel, disparando ações adaptativas que ajustam os comportamento dos seus serviços, de acordo com a disponibilidade de energia do dispositivo móvel. Por exemplo, a frequência de publicação dos dados por ser reduzida quando o nível da bateria está baixo (por exemplo, menor que 20%).

2.2.2 CDDL

O CDDL é uma camada de distribuição de dados que provê mecanismos que permitem a especificação, controle e monitoramento de requisitos de qualidade de informação e do serviço de distribuição de dados (GOMES, 2017).

Como middleware, o CDDL permite que as aplicações clientes assumam o papel de produtoras ou consumidoras de dados de contexto, fornecendo suporte à políticas de qualidade de serviço de distribuição de dados. Desta forma as aplicações podem especificar parâmetros que expressam seus requisitos de QoC para envio e recebimento de mensagens. A fim de fornecer suporte para o desenvolvimento de aplicações de IoT e IoMT, o CDDL atende aos seguintes requisitos (MUNIZ, 2017):

a) suporte à distribuição de dados local e remota;

b) suporte ao registro e descoberta distribuída de serviços;

c) modelo de programação uniforme e independente de localização;

d) suporte à entrega confiável de dados em cenários de mobilidade;

e) provisionamento e monitoramento de qualidade da informação;

f) provisionamento e monitoramento de qualidade do serviço de distribuição;

g) filtragem das informações.

A interação entre produtores e consumidores de dados de contexto se dá através do modelo *publisher/subscriber* com a utilização de *brokers*, implementando uma comunicação distribuída com o MQTT. O CDDL também fornece um *microbroker* que executa internamente no dispositivo móvel, o que permite que certas aplicações possam publicar e receber dados sem a dependência de conexão à Internet.

O CDDL recebe através de EventBus objetos do tipo `SensorData` vindos do S2PA. Este objeto é convertido em um objeto do tipo `Message`, todos os dados publicados pelo

CDDL são instâncias desta classe. Este novo objeto é então anotado com alguns parâmetros de QoI que não puderam ser informados no momento de captura do dado. Esta classe encapsula diversos atributos relevantes aos dados de contexto que representam, dentre eles, destaca-se:

a) `serviceName`: Nome do *smart object* que está relacionado a essa `Message`;

b) `serviceValue`: Valor da leitura dos dados do *smart object*;

c) `accuracy`: Acurácia do dado;

d) `measurementTime`: O *timestamp* em que a `Message` foi gerada;

e) `sourceLocationLatitude`: Latitude do M-Hub que gerou este dado;

f) `sourceLocationLongitude`: Longitude do M-Hub que gerou este dado;

g) `sourceLocationAltitude`: Altitude do M-Hub que gerou este dado;

h) `signal`: Intensidade do sinal no momento em que o *smart object* interagiu com o S2PA.

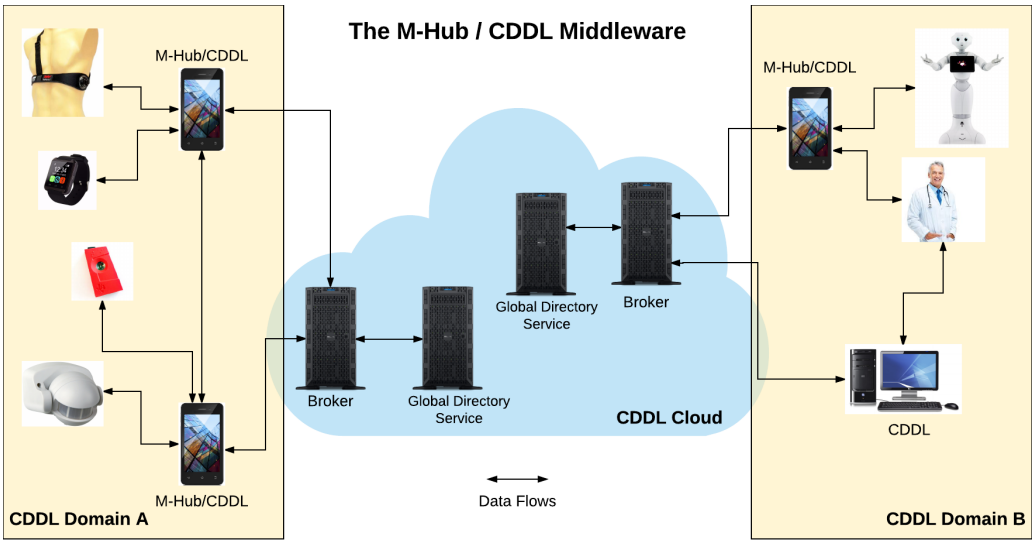
Ao receber um dado de contexto vindo do S2PA, o CDDL automaticamente o serializa em formato JSON e o publica em um *broker* MQTT definido no momento da inicialização do middleware. O atributo `serviceName` é utilizado para gerar parte do tópico.

A Figura 4 ilustra o uso da infraestrutura M-Hub/CDDL em um cenário de *Ambient Assisted Living*. Neste cenário os pacientes têm seus dados vitais monitorados por uma rede de sensores corporais. Os dados são coletados por smartphones equipados com aplicações M-Hub/CDDL que distribuem os dados em um conjunto de *brokers* em nuvem. Estes dados podem ser entregues aos médicos ou cuidadores e familiares do paciente.

Como já mencionado, o modelo de programação do CDDL respeita o padrão *publisher/subscriber* , o middleware oferece então duas classes: `Publisher` e `Subscriber` para realizar publicações e inscrições em tópicos, respectivamente. Uma aplicação que registra o interesse por algum tipo de dado de contexto será notificada através de um objeto `Message`, que representa este dado. O Código 2.1 apresenta um exemplo de como o desenvolvedor pode consumir dados de temperatura capturados pelo middleware.

Nas linhas 1 e 2, uma instância de `Subscriber` é criada e uma conexão com um *broker* é feita. Na linha 3 o programa registra interesse em dados do serviço de temperatura, internamente o `Subscriber` irá se inscrever em um tópico onde todos os dados de contexto deste serviço são publicados. Nas linhas 5 até 13 um *listener* para este objeto é definido, é a partir deste que os dados serão capturados. Todos os dados de contexto provenientes do serviço temperatura, chegarão ao método `onMessageArrived` como uma instância de `Message`.

Figura 4 – Visão geral do M-Hub/CDDL em um cenário de IoMT



Fonte: Gomes et al. (2017)

```
Subscriber subscriber = SubscriberFactory.createSubscriber();
subscriber.addConnection(connection); //connection with mqtt broker
subscriber.subscribeServiceByName("Temperature");

subscriber.setSubscriberListener(new ISubscriberListener() {
    @Override
    public void onMessageArrived(Message message) {
        String name = message.getServiceName();
        long signal = message.getSignal();

        Log.d(TAG, "Service name: " + name + "\tRSSI: " + signal);
    }
});
```

Código 2.1 – Modelo de programação *publisher/subscriber* do CDDL

3 Solução proposta

Visto que o CDDL somente exporta à camada de aplicação, eventos de leitura de dados de contexto. Significando que, dentre todos os objetos do tipo `SensorData` criados no S2PA, apenas aqueles com atributo "`action`" assumindo valor "`READ`" eram propagados ao `QoCEvaluator` no CDDL podendo assim serem percebidos pela aplicação (vide subseção 2.2.1.1).

Este fato implica em algumas limitações no desenvolvimento de aplicações. Imagine um cenário onde uma aplicação necessite de certos dados providos por um *smart object*—alocando recursos computacionais para processá-los. Em uma eventual desconexão com o *smart object*, o fluxo de dados do sensor cessaria de ser entregue à aplicação, contudo, nenhum fluxo ou notificação do evento de desconexão seria entregue. A aplicação não poderia decidir se o interrompimento do fluxo de dados se deu, devido a uma mudança na latência do envio de dados por parte do sensor ou por uma desconexão, não podendo então decidir sobre a necessidade da desalocação de recursos.

Esta limitação do CDDL dificultava ainda o desenvolvimento de aplicações que interagem com *smart objects* no ambiente por outros meios além de conexões. É o caso de aplicações de localização *indoor* baseadas em *beacons bluetooth*, onde os *beacons* são dispostos no interior de ambientes físicos e permanecem realizando *broadcast* de sua presença. Os dispositivos móveis não realizam tentativas de conexão, apenas percebem a presença destes *smart objects* e utilizam a intensidade do sinal capturado no momento.

O trabalho em questão tem como objetivo a implementação do mecanismo que propaga os eventos de descoberta, conexão e desconexão que ocorrem no S2PA do M-Hub até a camada de aplicação do CDDL, de forma que os desenvolvedores possam projetar aplicações que se adaptem a tais eventos.

Vale ressaltar que a implementação também deve permitir que estes eventos possam estar disponíveis para outras aplicações que estejam interessadas, utilizando para isso o MQTT. Ou seja—caso configurado desta forma—uma aplicação M-Hub/CDDL pode mudar seu comportamento baseado em eventos que foram disparados a partir de interações entre *smart objects* e smartphones remotos.

3.1 Requisitos de software

Requisitos funcionais

Os requisitos funcionais definidos para a solução são:

a) notificação de eventos de descoberta, conexão e desconexão de *smart objects* para a camada de aplicação do software;

b) separação dos fluxos de eventos, provendo uma API que permita a aplicação registrar interesse em cada tipo de evento individualmente;

c) permitir que estes eventos possam estar acessíveis tanto para a aplicação que os gera, quanto para aplicações remotas que registrem interesse em interações de outros smartphones com *smart objects*;

d) fornecer uma API assíncrona para o recebimento de cada notificação dos eventos desejados.

3.2 Implementação

3.2.1 Propagação de eventos do S2PA para o CDDL

Como descrito na subseção 2.2.1.1, o M-Hub encapsula todos os eventos em objetos do tipo `SensorData`, estes objetos devem então ser propagados para o CDDL. O S2PA comunica-se com o CDDL através do componente `QoCEvaluator`, como pode ser observado na Figura 2 (GOMES, 2017).

A comunicação entre os componentes é feita através da biblioteca EventBus[1]. O EventBus é uma biblioteca de eventos de código livre escrita em Java para a plataforma Android utilizando o padrão *publisher/subscriber* , fornecendo um mecanismo central de comunicação simplificando a interação entre componentes da aplicação.

O S2PA foi modificado então para publicar cada `SensorData` no EventBus, enquanto o `QoCEvaluator` registra interesse em receber objetos do tipo `SensorData` publicados no EventBus. O que efetivamente transfere todos os eventos gerados pelo S2PA ao CDDL.

3.2.2 Separação dos fluxos de eventos

Como todos os dados publicados pelo CDDL são do tipo `Message`, decidiu-se manter este padrão, de forma a manter a retrocompatibilidade e assim continuar suportando as aplicações antigas. Ao invés de utilizar a estratégia adotada pelo M-Hub (utilizar um atributo que identifica que tipo de evento o objeto está representando), adotou-se uma estratégia baseada em orientação a objetos. Foram criadas quatro novas classes que herdam os atributos de `Message`, são elas:

a) `ObjectFoundMessage`: Para os eventos de descoberta;

b) `ObjectConnectedMessage`: Para os eventos de conexão;

[1] <http://greenrobot.org/eventbus/>

c) `ObjectDisconnectedMessage`: Para os eventos de desconexão;

d) `SensorDataMessage`: Para eventos de leitura de dados.

Com esta abordagem é possível utilizar a API e métodos existentes, valendo-se do mecanismo de polimorfismo da orientação a objetos. Ao receber um `SensorData`, o `QoCEvaluator` faz o seguinte: identifica, utilizando o atributo `action`, que tipo de evento ele representa; instancia um objeto de um dos tipos descritos anteriormente; e o publica no *broker* MQTT como um objeto do tipo `Message`.

3.2.3 Definição da estrutura de tópicos

De forma a gerar fluxos diferentes para cada evento, e permitir consultas mais expressivas, o `QoCEvaluator` deve publicar cada mensagem gerada em um tópico específico, de acordo com o tipo de evento que este representa.

A estrutura de tópicos proposta para publicação de eventos de descoberta, conexão e desconexão segue o seguinte padrão:

$$\underbrace{\textbf{domain}}_{\text{identificador do CDDL}} \; / \; \underbrace{\textbf{clientId}}_{\text{identificador do cliente}} \; / \; \underbrace{\textbf{eventTopic}}_{\text{identificador do tipo de evento}}$$

Onde:

a) *domain*: É um prefixo que compõe o primeiro nível de todos os tópicos gerados pelo CDDL, isto permite que seja possível identificar as mensagens publicadas no contexto do middleware mesmo que um *broker* público seja utilizado (*e.g.*, realizando uma inscrição no tópico `domain/#`).

b) *clientId*: É um identificador da aplicação cliente que gerou tal mensagem. Este valor é informado pelo desenvolvedor ao iniciar o CDDL;

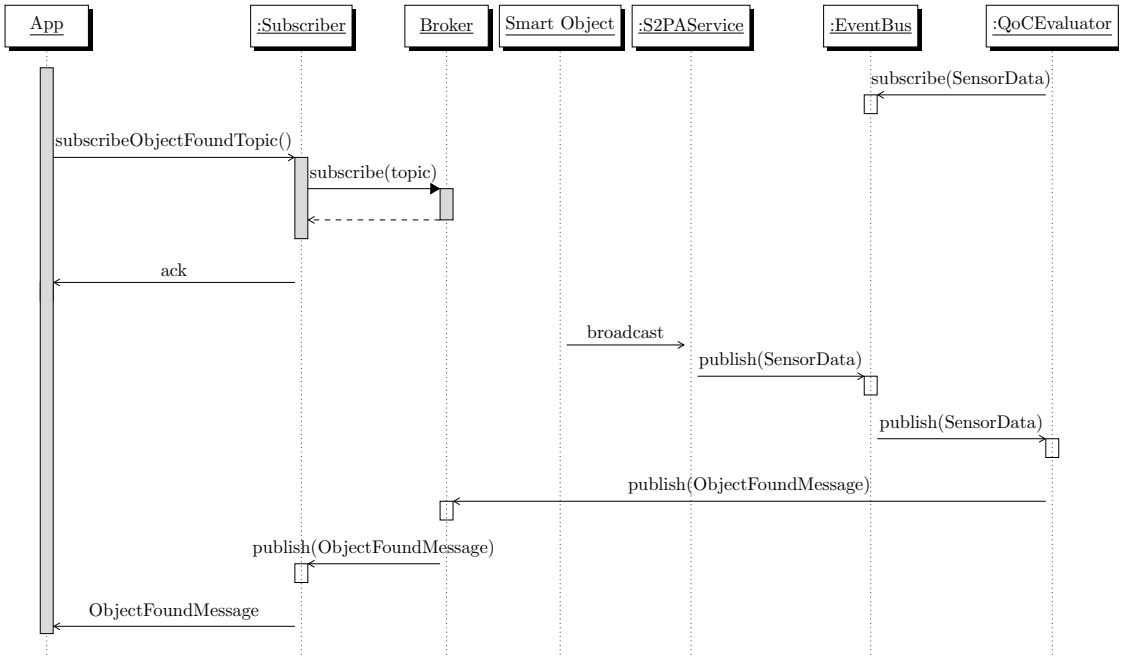
c) *eventTopic*: Identificador único para cada tipo de evento, ou seja, para os eventos de descoberta, conexão e desconexão:

- `object_found_topic`: Para os eventos de descoberta;
- `object_connected_topic`: Para os eventos de conexão;
- `object_disconnected_topic`: Para os eventos de desconexão.

A fim de se inscrever em eventos gerados por outras aplicações M-Hub/CDDL, faz-se uso dos caracteres coringa do MQTT descritos na subseção 2.1.1. Fazendo com que a aplicação se inscreva em tópicos com o seguinte padrão: `domain/+/eventTopic`.

A Figura 5 apresenta o diagrama de sequência do funcionamento da solução proposta. Ilustrando, neste caso, o processo de notificação de eventos de descoberta.

Figura 5 – Diagrama de sequência do mecanismo implementado



Fonte: Produzido pelo autor

3.2.4 Modelo de programação

Foram então adicionados métodos à interface `Subscriber` que permitem que a aplicação registre interesse em cada tipo de evento gerado através de interações entre *smart objects* e o próprio dispositivo móvel em que executa, e também métodos que permitem que a aplicação receba eventos gerados por outras aplicações. Estes métodos inscrevem a aplicação no tópico MQTT específico, e estão descritos adiante:

a) `subscribeObjectFoundTopic()`:
   - Inscreve a aplicação no tópico `domain/userId/object_found_topic`, fazendo com que ela receba os eventos de descoberta de *smart objects*.

b) `subscribeObjectConnectedTopic()`:
   - Inscreve a aplicação no tópico `domain/userId/object_connected_topic`, fazendo com que ela receba os eventos de conexão de *smart objects*.

c) `subscribeObjectDisconnectedTopic()`:
   - Inscreve a aplicação no tópico `domain/userId/object_disconnected_topic`, fazendo com que ela receba os eventos de desconexão de *smart objects*.

Aplicações que estão interessadas em certo tipo de serviço receberão uma instância de `Message` a cada evento gerado. Como ela é superclasse das anteriores, caso a aplicação deseje diferenciar qual tipo de evento o objeto `Message` que ela recebeu representa, é

possível verificar em tempo de execução se determinado objeto é instância das classes mais especializadas, como é mostrado no Código 3.1.

```
1  Subscriber = SubscriberFactory.createSubscriber();
2  subscriber.addConnection(connection); //connection with mqtt broker
3  subscriber.subscribeObjectConnectedTopic();
4  subscriber.subscribeObjectDisconnectedTopic();
5
6  subscriber.setSubscriberListener(new ISubscriberListener() {
7      @Override
8      public void onMessageArrived(Message message) {
9          if (message instanceof ObjectConnectedMessage) {
10             /* do something */
11         } else if (message instanceof ObjectDisconnectedMessage) {
12             /* do somethinge else */
13         }
14     }
15 });
```

Código 3.1 – Uso das classes mais especializadas em conjunto com a API existente

Nas linhas 1 e 2 uma instância de `Subscriber` é criada e uma conexão com um *broker* estabelecida. Nas linhas 3 e 4, a aplicação registra interesse em eventos de conexão e desconexão. Como foi realizado a inscrição em múltiplos tópicos com um único *subscriber*, todas as mensagens serão entregues em um mesmo *listener*, que é definido na linha 6. Nas linhas 9 e 11, a aplicação determina se o objeto que recebeu é instância de `ObjectConnectedMessage` ou `ObjectDisconnectedMessage`. Desta forma pode escolher o comportamento mais adequado para cada evento.

4 Avaliação quantitativa

O objetivo deste capítulo é descrever as avaliações utilizadas na solução proposta. A avaliação tem como objetivo determinar a performance e o percentual de perdas do mecanismo de descoberta, conexão e desconexão de *smart objects*.

A avaliação será realizada por meio de experimentos, onde a performance será avaliada através da contagem de tempo desde que o M-Hub efetivamente descobre um objeto até o momento em que a aplicação é informada sobre este evento. O percentual de perdas da solução consistirá na verificação de quantas operações de conexão, desconexão e descoberta geraram eventos correspondentes para a aplicação.

Em todos os experimentos, utilizou-se o *microbroker* interno do M-Hub/CDDL.

4.1 Experimento 1

4.1.1 Métricas

A fim de determinar a performance da solução em relação à entrega dos eventos de descoberta, a aplicação permanece constantemente anotando os *timestamps* de cada evento de descoberta emitido pelo S2PA, e os *timestamps* de notificações destes eventos na aplicação—ambos em milissegundos. O primeiro *timestamp* é subtraído do segundo de forma a calcular o tempo de propagação dos eventos de descoberta, que será de agora em diante referido como $\Delta t_{descoberta}$ e calculado conforme a Equação 4.1.

$$\Delta t_{descoberta} = t_{descoberta,app} - t_{descoberta,S2PA} \tag{4.1}$$

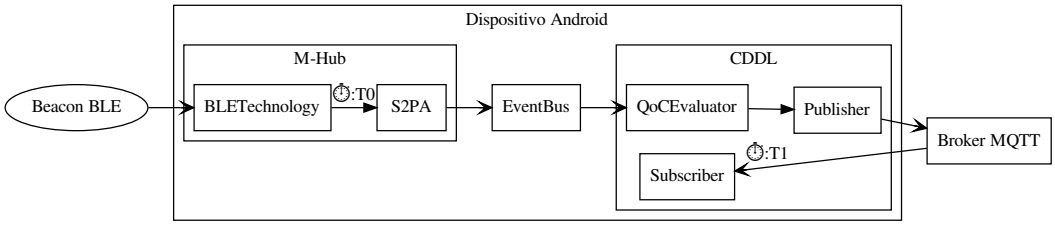
Onde $t_{descoberta,app}$ e $t_{descoberta,S2PA}$ se referem ao *timestamp* de uma notificação de descoberta na aplicação e ao *timestamp* do evento de descoberta gerado pelo S2PA, respectivamente.

Na Figura 6 é possível observar onde os *timestamps* utilizados na avaliação de performance são anotados. Ambos os *timestamps* são denotados na imagem por um cronômetro. Na imagem, $T0$ e $T1$ são equivalentes à $t_{descoberta,S2PA}$ e $t_{descoberta,app}$ da Equação 4.1.

Para avaliar o percentual de perdas das notificações de eventos, é realizada uma comparação entre a quantidade de eventos de descoberta de *smart objects* que foram gerados e quantas notificações foram entregues à aplicação, e será calculada através da Equação 4.2.

$$P_{perda} = \frac{EventosGerados - EventosNotificados}{EventosGerados} \cdot 100 \tag{4.2}$$

Figura 6 – Componentes onde há a captura de *timestamps*



Fonte: Produzido pelo autor

Onde $EventosGerados$ e $EventosNotificados$ se referem à quantidade de eventos gerados no S2PA e à quantidade de notificações desses eventos que chegaram à aplicação, respectivamente.

4.1.2 Cenário

Para medir os parâmetros anteriores, foi desenvolvido um cenário de uso, onde será possível determinar tais aspectos.

Este cenário de uso consiste em uma casa onde cada cômodo é equipado com um *beacon* BLE, calibrado de forma que o sinal emitido não possa ser detectado fora do cômodo e configurado para emitir 10 anúncios por segundo. Uma pessoa portando um smartphone é instruída a conduzir as atividades do dia a dia nesta residência. Este smartphone executa uma aplicação desenvolvida com o middleware M-Hub/CDDL que detecta os anúncios dos *beacons*, determinando em qual região da casa o indivíduo se encontra.

Ao entrar em um cômodo, o *beacon* será encontrado pelo M-Hub/CDDL, gerando um evento de descoberta no S2PA que deverá ser propagado para a aplicação.

4.1.3 Simulação da locomoção na residência

A locomoção de um indivíduo em uma residencia como descrito na subseção 4.1.2 foi simulado utilizando um *dataset*. Os dados utilizados para a simulação deste experimento foram obtidos do *dataset* disponibilizado por Byrne et al. (2018)[1]. Neste trabalho os autores instruíram que participantes conduzissem suas rotinas diárias em casa, enquanto sua localização na residência era monitorada.

O chão dos cômodos das casas foi marcado com etiquetas, onde cada etiqueta é uma imagem binária que codifica um número inteiro, e este, único para cada uma das etiquetas. Os participantes são equipados com uma câmera na região do torso que aponta

[1] O *dataset* pode ser obtido em (BYRNE; KOZLOWSKI, 2019)

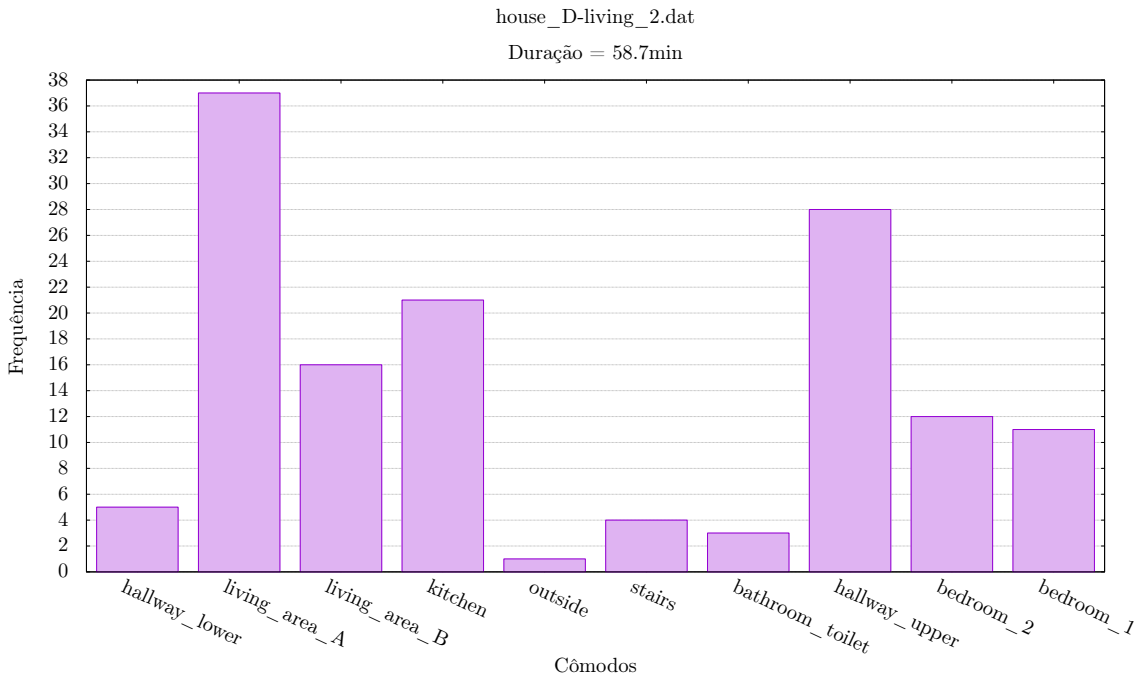
em direção ao chão, detectando e interpretando qual etiqueta está visível no momento, e deste modo, identificando em qual cômodo o participante se encontra.

No trabalho citado, o autor realizou o experimento em 4 residências distintas, e estas foram denotadas de "`Residence A`", "`Residence B`", "`Residence C`" e "`Residence D`". Para cada residência, o experimento foi conduzido múltiplas vezes, e cada um denominado de "`living_1`", "`living_2`", "`living_3`", ..., "`living_n`".

A fim de maximizar a quantidade de eventos de descoberta gerados na simulação, foi utilizado os dados do experimento "`living_2`" da casa "`Residence D`" pois este experimento possui um dos maiores tempos de monitoramento e quantidade de entrada em cômodos em todo o *dataset*. Os dados deste experimento consistem de 58 minutos de monitoramento de um indivíduo em uma casa de 10 cômodos. Os dados possuem uma resolução média de aproximadamente 11 medições por segundo.

Informações referentes à distribuição da quantidade de vezes em que o indivíduo visita um quarto estão sumarizadas na Figura 7.

Figura 7 – Frequência de entrada por cômodo



Fonte: Produzido pelo autor

O *dataset* original é composto principalmente por um arquivo de texto que contém em cada linha uma representação de um momento no decorrer do experimento. Entre os dados contidos por linha, destaca-se o *timestamp* da medição e a etiqueta detectada naquele instante.

Realizou-se um pré-processamento no *dataset* de forma a facilitar a utilização

dos dados para fim da simulação, gerando um arquivo contendo 3 valores separadas por espaço em que cada linha representa o instante em que o participante entrou e permaneceu continuamente em um cômodo. Os 3 valores são: um identificador numérico do cômodo, o nome do cômodo e o *timestamp* em que o indivíduo entrou naquele cômodo, como pode ser visto no Código 4.1.

```
#room    room-name          timestamp(seconds)
2        living_area_A      485.352
1        hallway_lower      506.273
6        stairs             513.380
8        hallway_upper      519.319
10       bedroom_1          526.159
8        hallway_upper      561.328
7        bathroom_toilet    565.832
8        hallway_upper      628.495
9        bedroom_2          632.065
```

Código 4.1 – Parte do *dataset* pré-processado

4.1.4 Simulação dos *beacons*

Como descrito anteriormente, o cenário consiste em um *beacon* em cada cômodo de uma casa. Utilizando os dados descritos na subseção 4.1.3 foi realizado uma simulação de uma pessoa portando um smartphone com uma aplicação M-Hub/CDDL que detecta a presença de *beacons* em cada cômodo onde o indivíduo adentra, estes *beacons* foram configurados de forma a emitir 10 anúncios por segundo. É esperado, então, que o aplicativo detecte um anúncio a cada 0.1 segundo em que o participante permanece em um cômodo.

Para este experimento, cada anúncio detectado corresponde a um evento de descoberta gerado pelo S2PA, este evento deve então ser propagado até a aplicação.

Utilizando o *dataset* já descrito, a cada momento em que o participante entra em determinado cômodo, calcula-se a quantidade de anúncios o *beacon* daquele cômodo transmitirá durante o tempo de permanência no local, o cálculo é apresentado a seguir.

$$anuncios = \frac{tempoPermanencia}{0.1} \tag{4.3}$$

Onde $tempoPermanencia$ denota o tempo que o indivíduo permaneceu no cômodo. Com isso foi possível calcular previamente os valores da Tabela 1. Esta tabela mostra os valores do termo $EventosGerados$ na Equação 4.2, estes valores serão posteriormente comparados com a quantidade de eventos notificados para calcular o percentual de perdas.

Tabela 1 – Quantidade de eventos que serão gerados no experimento 1

	$EventosGerados$
Eventos de descoberta	34814
Eventos de conexão	0
Eventos de desconexão	0

Fonte: Produzido pelo autor

4.1.5 Recursos computacionais

O experimento descrito foi executado em um smartphone Lenovo Moto Z Play, com processador de 8 núcleos e 2GHz, e 3GB de memória RAM.

4.1.6 Resultados

Tabela 2 – Quantidade de eventos notificados no experimento 1

	$EventosGerados$	$EventosNotificados$	P_{perdas} (%)
Eventos de descoberta	34814	34814	0
Eventos de conexão	0	0	—
Eventos de desconexão	0	0	—

Fonte: Produzido pelo autor

Tabela 3 – Performance da entrega de mensagens do experimento 1

	Média	Desvio padrão	Mínimo	Máximo	Intervalo de 95% de confiança
$\boldsymbol{\Delta t_{descoberta}}$ **(ms)**	20.1	3.7	7.0	78.9	20.1 ± 0.03

Fonte: Produzido pelo autor

4.1.7 Análise dos resultados

Mesmo com os *beacons* configurados para realizarem *broadcast* a cada 0.1 segundo, o que representa uma frequência muito alta para aplicações similares—como exemplo, em Kim et al. (2016) o autor utiliza 1 broadcast a cada dois segundos—a solução conseguiu notificar a aplicação em um tempo na casa de milissegundos. Este tempo pode ser considerado é desprezível para aplicações reais.

Do ponto de vista do percentual de perdas, a pesar de um grande número de eventos em um curto espaço de tempo, não houve perdas de notificações de eventos de descobertas dos *beacons*.

A solução apresenta características satisfatórias para o uso em aplicações onde as interações com *smart objects* não são baseadas em conexões, como é o caso de aplicações de localização *indoor*.

4.2 Experimento 2

O segundo experimento visa avaliar o desempenho e o percentual de perdas da solução proposta, entretanto levando em consideração também os eventos de conexão que não foram analisados no primeiro experimento.

4.2.1 Métricas

As avaliações de desempenho e percentual de perdas deste experimento são agora baseadas no tempo entre eventos de *descoberta* no S2PA e notificações de *conexão* na aplicação. A Equação 4.4 será utilizada para avaliar a performance da solução.

$$\Delta t_{conectado} = t_{conexao,app} - t_{descoberta,S2PA} \tag{4.4}$$

Onde $t_{descoberta,S2PA}$ é o *timestamp* de quando um evento de descoberta de um *smart object* é gerado no S2PA, $t_{conexao,app}$ é *timestamp* de quando o a notificação do evento de conexão do mesmo *smart object* é entregue a aplicação e $\Delta t_{conectado}$ o tempo que processo de descoberta até notificação de conexão levou para ser concluído.

Neste contexto, será medido o tempo entre o evento de *descoberta* do *smart object* no S2PA e a notificação do evento de *conexão* na camada de aplicação. Deseja-se saber em quanto tempo desde a descoberta do dispositivo os dados disponibilizados estarão disponíveis para o consumo pela aplicação.

Para medir o percentual de perdas será utilizado a Equação 4.2.

4.2.2 Cenário

O experimento consiste em um cenário de uso composto por uma casa equipada com um sistema multimídia na sala de estar que possibilita que smartphones se conectem a ele através de *bluetooth*, onde o conteúdo da televisão se adapta de acordo com os usuários que estão presentes na sala, e a interação com as aplicações da televisão digital acontece através do smartphone.

O sistema multimídia pode ter ciência de qual usuário está presente através dos smartphones conectados no momento, pois ele armazena a relação dos moradores da residência com seus respectivos smartphones.

Uma aplicação Android foi desenvolvida com o middleware M-Hub/CDDL que permanece ativamente procurando e se conectando ao sistema multimídia no ambiente.

Ao entrar na sala de estar o sistema multimídia é encontrado e em seguida a conexão é estabelecida. Quando o usuário sai da sala de estar a conexão é desfeita.

4.2.3 Simulação do sistema multimídia

Para este experimento foi feita uma simulação de um indivíduo se locomovendo em uma residência. Foram utilizados os dados do *dataset* descrito na subseção 4.1.3 para esta simulação.

Escolheu-se também o experimento "`living_2`" da casa "`Residence D`" por fornecer um dos maiores tempos de monitoramento do participante juntamente com a maior quantidade de visitas à sala de estar. Será utilizado apenas a parte do *dataset* referentes à entradas no cômodo denotado como "`living_area_A`" na Figura 7, pois este cômodo foi escolhido como a sala de estar onde o sistema multimídia se localiza.

Desta forma, a todo momento em que o participante entra na sala de estar acontece interações entre o aplicativo e o sistema multimídia, essas interações são compostas de um evento de descoberta, seguido por um evento de conexão. Ao sair do quarto um evento de desconexão é também gerado. Cada um desses eventos são gerados pelo S2PA e notificados posteriormente à aplicação.

Como pode ser notado na Figura 7, é possível determinar a quantidade de vezes em que o participante entra na sala de estar. Deste modo pode-se saber a quantidade total de interações com o sistema multimídia ocorrerão de antemão. A Tabela 4 sumariza os resultados esperados para o experimento.

Tabela 4 – Quantidade de eventos que serão gerados no experimento 2

	EventosGerados
Eventos de descoberta	37
Eventos de conexão	37
Eventos de desconexão	37

Fonte: Produzido pelo autor

Para a modelagem do tempo entre a descoberta e conexão do smartphone e o *smart object*, foram utilizados dados empíricos coletados pelo autor. Um *smart object* baseado na plataforma Arduino e com suporte à tecnologia BLE foi construído, e várias amostras de tempo entre a descoberta deste dispositivo e a conexão com ele foram tomadas.

4.2.4 Resultados

Tabela 5 – Quantidade de eventos notificados no experimento 2

	$EventosGerados$	$EventosNotificados$	P_{perdas} (%)
Eventos de descoberta	37	37	0
Eventos de conexão	37	37	0
Eventos de desconexão	37	37	0

Fonte: Produzido pelo autor

Tabela 6 – Performance da entrega de mensagens do experimento 2

	Média	Desvio padrão	Mínimo	Máximo	Intervalo de 95% de confiança
$\boldsymbol{\Delta t_{conectado}}$ **(ms)**	501.4	60.8	425.1	654.5	501.4 ± 19.6

Fonte: Produzido pelo autor

4.2.5 Análise dos resultados

Assim como no experimento 1, não foi possível observar perda de notificações neste experimento. O tempo entre a descoberta de um *smart object* no ambiente e sua conexão, e consequentemente a disponibilidade de consumir os dados oferecidos por este, se manteve em torno de 500 milissegundos.

Isto implica que uma aplicação que necessite de dados oferecidos por sensores no ambiente, começaria a receber estes dados em média 0.5 segundo após o primeiro encontro com este dispositivo.

Tomando como exemplo o cenário em que uma aplicação faz uso de dados de umidade do ar fornecidos por sensores em postes públicos que se comunicam através da tecnologia BLE, é possível fazer uma análise de o quão rápido o usuário pode passar nas redondezas do sensor sem perder a oportunidade de capturar o dado. Assumindo que o raio de cobertura do sinal *bluetooth* é de 5 metros, e que o usuário tem que atravessar o diâmetro completo para perder o sinal, ele deve percorrer 10 metros desde o momento que encontra o dispositivo até perder o sinal.

Dividindo a distância que ele deve percorrer pelo tempo médio que uma conexão é realizada, temos:

$$\frac{10\,\mathrm{m}}{0.5\,\mathrm{s}} = 20\,\mathrm{m/s} = 72\,\mathrm{km/h}$$

A implicação deste valor é que o usuário pode se locomover pelo entorno do sensor com até 72km/h, e ainda assim adquirir o dado que a aplicação necessita. Fazendo com que o usuário não necessite diminuir a velocidade de seu trajeto estando a pé, de bicicleta ou conduzindo veículo motorizado na maioria dos cenários urbanos.

5 Conclusões

A IoT tem motivado o desenvolvimento de uma gama de aplicações com os mais variados requisitos. Os requisitos de mobilidade e adaptação à ambientes onde os *smart objects* interagem de forma oportunísticas fazem da IoMT particularmente desafiadora. A habilidade de uma aplicação ser notificada com eficiência a respeito de interações com objetos inteligentes presentes no ambiente é crucial para o funcionamento de diversas aplicações.

Este trabalho traz a implementação de tal funcionalidade no middleware M-Hub/CDDL, componente de software construído em parceria dos laboratórios de pesquisa LSDi–UFMA e LAC–PUC-Rio.

A solução proposta permite que eventos de descoberta, conexão e desconexão de *smart objects* obtidas pelo M-Hub sejam distribuídas pelo CDDL. Implementada utilizando o protocolo MQTT, permitiu que as alterações realizadas no middleware seguissem os padrões já estabelecidos de programação e comunicação.

Os resultados experimentais mostram que o mecanismo implementado é eficiente e atende bem as necessidades de um grande número de aplicações de IoMT.

5.1 Trabalhos futuros

Como trabalhos futuros está previsto a realização de novas avaliações que possam responder sobre a performance da solução ao utilizar um *broker* em rede, e como as condições da rede a afeta. Também espera-se avaliar características como a escalabilidade do mecanismo de notificação, em particular, como ele se comporta na presença de um grande número de *smart objects* no ambiente.

Referências

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, IEEE, v. 17, n. 4, p. 2347–2376, 2015. Citado na página 15.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, Elsevier North-Holland, Inc., New York, NY, USA, v. 54, n. 15, p. 2787–2805, out. 2010. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2010.05.010>. Citado na página 12.

BANDYOPADHYAY, D.; SEN, J. Internet of things: Applications and challenges in technology and standardization. *Wireless Personal Communications*, Springer, v. 58, n. 1, p. 49–69, 2011. Citado na página 12.

BYRNE, D.; KOZLOWSKI, M. *Residential Wearable RSSI and Accelerometer Measurements with Detailed Annotations*. figshare, 2019. Disponível em: <https://figshare.com/articles/Residential_Wearable_RSSI_and_Accelerometer_Measurements_with_Detailed_Annotations/6051794/5>. Citado na página 31.

BYRNE, D. et al. Residential wearable rssi and accelerometer measurements with detailed location annotations. *Scientific data*, Nature Publishing Group, v. 5, p. 180168, 2018. Citado na página 31.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley Professional, 1994. ISBN 0201633612. Citado na página 20.

GOMES, B. d. T. P. *Uma Abordagem de Middleware com Suporte à Qualidade de Contexto voltada para Aplicações de Internet das Coisas*. Tese (Doutorado) — Universidade Federal do Maranhão, 2017. Programa de pós-graduação em engenharia de eletricidade/CCET. Disponível em: <https://tedebc.ufma.br/jspui/handle/tede/tede/2105>. Citado 3 vezes nas páginas 21, 22 e 26.

GOMES, B. d. T. P. et al. A middleware with comprehensive quality of context support for the internet of things applications. *Sensors*, MDPI AG, v. 17, n. 12, p. 2853, dec 2017. Disponível em: <https://doi.org/10.3390%2Fs17122853>. Citado 3 vezes nas páginas 13, 17 e 24.

GUBBI, J. et al. Internet of things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, Elsevier, v. 29, n. 7, p. 1645–1660, 2013. Citado na página 12.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. MQTT-S a publish/subscribe protocol for wireless sensor networks. In: IEEE. *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. [S.l.], 2008. p. 791–798. Citado na página 16.

KIM, J.-E. et al. Navigating visually impaired travelers in a large train station using smartphone and bluetooth low energy. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2016. (SAC '16),

p. 604–611. ISBN 978-1-4503-3739-7. Disponível em: <http://doi.acm.org/10.1145/2851613.2851716>. Citado na página 34.

LEE, S. et al. Correlation analysis of MQTT loss and delay according to qos level. In: *The International Conference on Information Networking 2013 (ICOIN)*. [S.l.: s.n.], 2013. p. 714–717. ISSN 1550-445X. Citado na página 15.

LIGHT, R. A. *Mosquitto man page*. [S.l.], 2019. Acessado em 19/06/2019. Disponível em: <https://mosquitto.org/man/mosquitto-8.html>. Citado na página 16.

MUNIZ, L. C. M. *Avaliação e monitoramento de QoC em sistemas cientes de contexto*. Dissertação (Mestrado) — Universidade Federal do Maranhão, 2017. Programa de pós-graduação em ciência da computação/CCET. Disponível em: <https://tedebc.ufma.br/jspui/handle/tede/tede/2065>. Citado na página 22.

NAHRSTEDT, K. et al. Internet of mobile things: Mobility-driven challenges, designs and implementations. In: IEEE. *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*. [S.l.], 2016. p. 25–36. Citado na página 12.

TALAVERA, L. E. et al. The mobile hub concept: Enabling applications for the internet of mobile things. In: IEEE. *Pervasive computing and communication workshops (PerCom workshops), 2015 IEEE international conference on*. [S.l.], 2015. p. 123–128. Citado 4 vezes nas páginas 12, 13, 18 e 19.

TANTITHARANUKUL, N. et al. MQTT-Topics management system for sharing of open data. In: IEEE. *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*. [S.l.], 2017. p. 62–65. Citado na página 16.