Pontifícia Universidade Católica
do Rio de Janeiro

Felipe Oliveira Carvalho

Descoberta Contínua de Serviços em IoT

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
Abril de 2017

Pontifícia Universidade Católica
do Rio de Janeiro

Felipe Oliveira Carvalho

Descoberta Contínua de Serviços em IoT

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 assinada.

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

Prof. Sergio Colcher
Departamento de Informática – PUC-Rio

Prof. Francisco José da Silva e Silva
Universidade Federal do Maranhão – UFMA

Prof. Márcio da Silveira Carvalho
Coordenador Setorial do Centro Técnico Científico – PUC-Rio

Rio de Janeiro, 26 de Abril de 2017

Felipe Oliveira Carvalho

O autor graduou-se em Sistemas de Informação pela Universidade Federal de Sergipe (São Cristóvão, Sergipe) no ano de 2014. Ingressou no Mestrado em Informática na PUC-Rio no ano de 2015. Durante a etapa acadêmica de mestrado, participou de projetos no Laboratory for Advanced Collaboration (LAC), coordenado pelo professor Markus Endler.

Ficha Catalográfica

Carvalho, Felipe Oliveira

Descoberta Contínua de Serviços em IoT / Felipe Oliveira Carvalho; orientador: Markus Endler. – 2017.

v., 65 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Sistemas Distribuídos;. 3. Internet das Coisas;. 4. Middleware;. 5. Descoberta de Serviços;. 6. Processamento de Eventos Complexos.. I. Endler, Markus. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Primeiramente agradeço e louvo a Deus por ter proporcionado a oportunidade de chegar até aqui neste caminho de mestrado.

Agradeço a minha querida esposa Biancha, por ter suportado comigo os dois anos de jornada no Rio de Janeiro para que este mestrado se tornasse realidade. Obrigado Bia! Graças à sua companhia esta tarefa tornou-se menos árdua.

Agradeço imensamente aos meus pais Rusiley e Luis Antonio por todo o apoio e carinho, por terem acreditado nesse projeto de mestrado no Rio de Janeiro. Sem o apoio deles, essa jornada teria sido muito mais difícil.

Agradeço aos meus avós, tios e primos de Sergipe. A família é tão grande que se tivesse que citar todo mundo, não caberia aqui. Obrigado! Sei o quanto todos torceram e apoiaram desde sempre. Agradeço em especial às minhas duas irmãs, Luisa e Lorena, por todos os momentos compartilhados em toda uma vida e mesmo agora à distância. Agradeço também a todos os agregados da família (que não são poucos).

Agradeço a todos da família de Bia, minha segunda família, em especial aos meus sogros Antonio e Vania, e meus cunhados Andson e Wandson, por toda a torcida, apoio e orações.

Também gostaria de agradecer ao meu orientador, Professor Markus Endler, por todo o apoio, incentivo, disponibilidade, paciência e valiosas lições. Tem sido um enorme privilégio trabalhar com ele durante estes dois últimos anos.

Agradeço também aos professores Sergio Colcher e Franscisco Silva, por terem engrandecido a construção deste trabalho de pesquisa com valiosas visões.

Agradeço aos professores Sergio Lifschitz, Noemi Rodriguez, e Hermann, por todos os ensinamentos durante matérias e discussões acadêmicas. Agradeço também a todo o pessoal da secretaria do departamento de informática da PUC-Rio, pelo apoio e suporte.

Gostaria também de agradecer aos colegas que sofreram junto comigo em PAA, Daniel, Fernando, Paulo e Pedro. PAA tem a capacidade de gerar amigos pra vida toda.

Agradeço a todos os amigos do LAC, em especial ao Marcos Roriz, ao Luis Talavera e ao Bruno Olivieri por todo o apoio acadêmico e valiosas dicas. Também agradeço ao pessoal do Telemídia, por todo apoio, receptividade e ajuda com discussões sobre esta dissertação.

Agradeço também a todos os meus amigos de Sergipe e do Rio, que direta ou indiretamente proporcionaram momentos de descontração e amizade nesses últimos dois anos.

Por último, mas não menos importante, gostaria de agradecer à CAPES pelo apoio financeiro e à PUC-Rio pela bolsa de isenção de mensalidades do mestrado.

Resumo

Carvalho, Felipe Oliveira; Endler, Markus. **Descoberta Contínua de Serviços em IoT**. Rio de Janeiro, 2017. 65p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A popularização da Internet das Coisas (IoT, *Internet of Things*) provocou uma crescente oportunidade para a criação de aplicações em diversas áreas, através da combinação do uso de sensores e/ou atuadores. Em ambientes de IoT, o papel de elementos chamados de *gateways* consiste em fornecer uma camada de comunicação intermediária entre os dispositivos de IoT e serviços de nuvem. Um fator crucial para a construção de aplicações em larga escala é que os dispositivos de IoT possam ser utilizados de maneira transparente, num paradigma orientado a serviços, onde detalhes de comunicação e configuração destes objetos são tratados pelos *gateways*. No modelo de serviços, as aplicações devem descobrir as interfaces de alto-nível dos dispositivos e não precisam lidar com detalhes subjacentes, que são tratados pelos *gateways*. Em cenários de grande dinamismo e mobilidade (com conexões e desconexões de dispositivos acontecendo a todo momento), a descoberta e configuração de objetos deve ocorrer de forma contínua. Os protocolos de descoberta de serviços tradicional, como o *Universal Plug and Play* (UPnP) ou o *Service Location Protocol* (SLP), não foram desenvolvidos levando em consideração o alto dinamismo de ambientes IoT. Nesse sentido, introduzimos o processamento de eventos complexos (CEP), que é uma tecnologia para processamento em tempo real de fluxos de eventos heterogêneos, que permite a utilização de consultas em linguagem CQL (*Continuous Query Language*) para a busca de eventos de interesse. Em um modelo onde os eventos relacionados à descoberta de sensores são enviados para um fluxo CEP, consultas expressivas são escritas para que uma aplicação descubra continuamente serviços de interesse. Este trabalho apresenta a extensão do MHub/CDDL para o suporte à descoberta contínua de serviços em IoT, utilizando CEP. O MHub/CDDL (Mobile Hub / Context Data Distribution Layer) é um middleware para descoberta de serviços e gerenciamento de qualidade de contexto em IoT, desenvolvido numa parceria entre o Laboratory for Advanced Collaboration (LAC) da PUC-Rio e o Laboratório de Sistemas Distribuídos Inteligentes (LSDi) da Universidade Federal do Maranhão (UFMA). A implementação deste trabalho é feita para a plataforma Android (Java) e um estudo de caso no domínio de estacionamentos inteligentes é conduzido e implementado, elucidando o uso do mecanismo de descoberta contínuo.

Palavras-chave

Sistemas Distribuídos; Internet das Coisas; Middleware; Descoberta de Serviços; Processamento de Eventos Complexos.

Abstract

Carvalho, Felipe Oliveira; Endler, Markus (Advisor). **Continuous Service Discovery in IoT**. Rio de Janeiro, 2017. 65p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The popularization of the Internet of Things sparked a growing opportunity for the creation of applications in various areas, by combining the use of sensors and/or actuators. In IoT environments, the role of elements called gateways is to provide an intermediate communication layer between IoT devices and cloud services. A crucial factor for the construction of large-scale applications is to allow the use of IoT devices in a transparent manner, in a service-oriented paradigm, where details of communication and configuration are handled by the gateways. In service model, applications must discover the high-level interfaces of the devices and do not have to deal with underlying details that are handled by gateways. In scenarios of high dynamism and mobility (with connections and disconnections of devices occuring all the time), this discovery and configuration must occur continuously. Traditional service discovery protocols, such as Universal Plug and Play (UPnP) or Service Location Protocol (SLP), have not been developed taking into consideration the high dinamicity of IoT environments. In this sense, we introduce complex event processing (CEP), which is a technology for real-time processing of heterogeneous event flows, which allows the use of CQL (*Continuous Query Language* for the search of events of interest. In a model where events related to sensor discovery are sent to a CEP flow, expressive queries are written for an application to continuously discover services of interest. This work presents the extension of MHub / CDDL to support continuous service discovery in IoT, using CEP. The MHub / CDDL (Mobile Hub / Context Data Distribution Layer) is a middleware for service discovery and quality context management in IoT, developed in a partnership between the Laboratory for Advanced Collaboration (LAC) from PUC-Rio and the Laboratório de Sistemas Distribuídos Inteligentes (LSDi) from Universidade Federal do Maranhão (UFMA). The implementation of this work is done in Android (Java) platform and a case study in the domain of smart parking is conducted and implemented, elucidating the use of the continuous discovery mechanism.

Keywords

Distributed Systems; Internet of Things; Middleware; Service Discovery; Complex Event Processing.

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

A Internet of Things (IoT, Internet das Coisas) consiste num paradigma que visa estender a conectividade Internet a uma diversa gama de dispositivos, frequentemente chamados de objetos inteligentes. Grandes empresas de tecnologia, como a Amazon e a AT&T, têm investido no mercado emergente de IoT[1], colocando a área em evidência. Estas chamadas "coisas" inteligentes passam então a ser integradas numa infraestrutura de rede, tendo seus respectivos identificadores, atributos físicos e interfaces [2]. Estima-se que por volta de 2020, mais de 20 bilhões de objetos estejam conectados à rede [3]. A demanda de aplicações em IoT vai desde áreas como monitoração de meio ambiente, transportes e saúde, até a produção industrial, redes sociais e marketing. Exemplos de objetos inteligentes incluem: (i) Sensores simples de diversos tipos como o GPS, o acelerômetro, o giroscópio, sensor de luminosidade, sensor de gás carbônico, sensor de fumaça, dentre outros; (ii) Atuadores como alto-falantes, telas, fechaduras eletrônicas, dentre outros; (iii) Objetos complexos, construídos com uma composição de sensores e atuadores, como geladeiras, ar-condicionados, dentre outros.

Os objetos inteligentes de IoT são tipicamente equipados com um mecanismo de comunicação via rádio (e.g. Bluetooth, Wi-Fi, NFC, etc), um microcontrolador e sensores e/ou atuadores. Eles são limitados em temos de capacidade de processamento, memória e energia, por serem tipicamente pequenos, portáteis e equipados com uma fonte limitada de energia. Essas limitações impõem uma busca por uma melhor eficiência energética através de um melhor aproveitamento do poder de processamento e dos mecanismos de comunicação [4]. Outro desafio que conflita com a interconectividade proposta pela área de IoT reside na pouca padronização de protocolos de comunicação e nas tecnologias de *hardware* e *software* em constante desenvolvimento. Deste modo, pesquisas em IoT têm contribuído para o surgimento de diversos protocolos e tecnologias para dispositivos limitados, como por exemplo o 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*), o CoAP (*Constrained Application Protocol*), o RFID (*Radio-Frequency Identification*), o

[1]Matéria da Computer World - `http://www.computerworlduk.com/galleries/data/12-most-powerful-internet-of-things-companies-3521713/`

BLE (*Bluetooth Low Energy*), dentre outros.

Em IoT, o papel de elementos chamados de *gateways* consiste em fornecer um meio de comunicação intermediário entre os objetos inteligentes e serviços de nuvem. Visto que uma diversa gama de objetos inteligentes não possui mecanismos para se comunicar diretamente dentro de uma rede TCP/IP, os *gateways* de IoT estendem a conectividade destes objetos através de um *hardware* que se conecta a eles utilizando redes WPAN (*Wireless Personal Area Network*) e desta forma encaminha os dados desses objetos para serviços de nuvem ou outras aplicações. Um fator crucial para a construção de aplicações IoT em larga escala é que os objetos inteligentes possam ser utilizados de maneira transparente, num paradigma orientado a serviços, onde detalhes de comunicação e configuração destes objetos são tratados pelos *gateways*. Através do conceito de serviços, os recursos de objetos inteligentes podem ser expostos por meio de interfaces de alto nível, que escondem detalhes do *hardware* e aspectos de controle [1]. Nesse contexto, descoberta de serviços tem um papel significativo, pois permite que seja definido algum critério de busca para a obtenção de recursos em IoT. Através deste mecanismo de descoberta, uma aplicação obtém acesso ao recurso desejado utilizando um protocolo pré-acordado.

Os mecanismos de descoberta de serviço tradicionais como o UPnP (*Universal Plug and Play*), JINI (Atualmente chamado de Apache River) e SLP (*Service Location Protocol*) abordam a descoberta em âmbito de rede local e não foram projetados com foco em ambientes de alto dinamismo e distribuídos como os de IoT. Esse alto dinamismo se refere a cenários onde a descoberta atua em um ambiente de mobilidade, isto é, nem o *gateway* responsável pela descoberta e/ou nem os objetos inteligentes são fixos (estacionários). Esse cenário de mobilidade irrestrita é conhecido como IoMT (*Internet of Mobile Things*, Internet das Coisas Móveis) [5]. Nesses cenários, um mecanismo de descoberta oportunístico (contínuo e assíncrono) deve ser desenvolvido para lidar com conexões e desconexões de objetos no *gateway*.

O conceito de Cidades Inteligentes tem por objetivo trazer a pauta de um melhor uso de recursos públicos numa cidade, aumentando a qualidade dos serviços oferecidos para os cidadãos ao mesmo tempo em que reduz custos operacionais da administração pública. Este objetivo pode ser alcançado através de uma IoT urbana. Isto inclui: Infraestruturas de comunicação, serviços de armazenamento e processamento de dados de sensores, serviços de controle de atuadores, dentre outros [6]. Dentro desse conceito, podemos ilustrar duas aplicações hipotéticas que se beneficiariam de um mecanismo de descoberta contínuo:

– **Estacionamentos Inteligentes**: No ambiente de uma cidade inteligente,

apresentado na Figura 1.1, um motorista deseja procurar por vagas em estacionamentos à medida em que se aproxima deles. Essa é uma informação com grau de validade curto, pois por exemplo, não interessa para o motorista saber da disponibilidade de vagas ao redor do seu destino quando ele ainda está em casa (ou longe do destino). Uma possível aplicação para o motorista poderia manter um registro de interesse em vagas através de uma consulta que fica armazenada. Ao se aproximar de seu destino, o motorista recebe em sua aplicação as informações de vagas que estão próximas e desocupadas. Esse cenário pode incluir atributos desejados pelo motorista, como tempo estimado de disponibilidade de vagas, ou características específicas como vagas especiais para deficientes físicos. Nesse cenário ilustrado acima, um *middleware* com capacidade de descoberta contínua apoiaria a construção dessa aplicação. Sensores que detectam a ocupação das vagas seriam automaticamente descobertos pelo *gateway* associado a este *middleware* à medida em que o motorista se move, e teriam suas funcionalidades disponibilizadas como serviços. A aplicação não teria que lidar com os aspectos de configuração e comunicação subjacentes aos sensores e se preocuparia apenas em utilizar os serviços descobertos.

– **City Tagging**[2]: Um projeto denominado *City Tagging* (etiquetas em cidades), elaborado pela IEEE (*Institute of Electrical and Electronics Engineers*) como cenário de IoT para cidades inteligentes do futuro, também se beneficiaria de um mecanismo de descoberta contínuo oportunístico. Nesse cenário hipotético para IoT do futuro, eles projetam o uso de *tags* (etiquetas) virtuais em locais de uma cidade para permitir consulta através da internet. As *tags* poderiam fornecer informações importantes sobre localidades, beneficiando aplicações no domínio de turismo, por exemplo. Um exemplo de aplicação onde um usuário andando pela cidade com um *smartphone* registra o interesse em determinados gostos pessoais enquanto visita a cidade, poderia fazer uso do mecanismo de descoberta contínuo. O usuário poderia receber notificações de interesse, como sugestões de lugares para visitar ou locais para comer, sempre que se aproximasse de uma *tag*. A aplicação seria poupada de lidar com a descoberta, comunicação e configuração das *tags* e se preocuparia apenas em implementar as funcionalidades desejadas.

As diversas componentes de uma aplicação de IoT necessitam de um mecanismo de comunicação. Em IoT, a comunicação envolve geralmente troca

[2]Projeto *City Tagging* - `http://iot.ieee.org/iot-scenarios.html?prp=6`

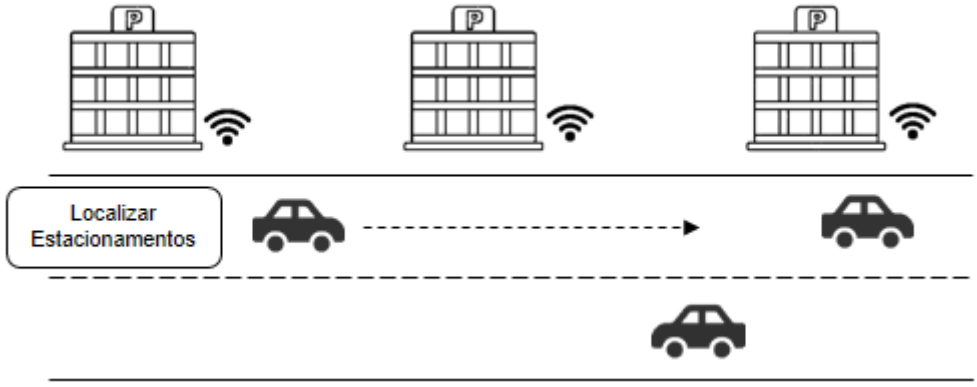


Figura 1.1: Cenário de uma cidade com estacionamentos inteligentes.

de mensagens pequenas porém frequentes, constituindo um fluxo de dados contínuo [7]. Nesse contexto, o CEP (*Complex Event Processing*) atua como uma tecnologia para processamento em tempo real de fluxos de eventos e permite o uso de uma linguagem de consulta denominada CQL (*Continuous Query Language*). O processamento CEP considera fluxos de eventos heterogêneos, vindos de fontes distintas, e procura nos eventos por padrões de ordem causal, lógica ou temporal. Ademais, CEP é capaz de derivar eventos complexos a partir dos fluxos, ou seja, de inferir uma relação entre eventos simples para a composição de eventos mais alta complexidade.

Além disso, sistemas distribuídos como os de IoT demandam por um modelo de comunicação mais flexível, que reflete a natureza dinâmica e desacoplada de aplicações. O paradigma de comunicação síncrona ponto-a-ponto leva à criação de aplicações rígidas e estáticas, que dificultam a construção de sistemas distribuídos robustos [8]. Um paradigma que apoia a construção de mecanismos distribuídos assíncronos é o de *Publish-Subscribe*. Nesse paradigma, existe o conceito de um barramento de serviços onde entidades denominadas de publicadores anunciam seus serviços e consumidores se subscrevem para receber serviços de interesse. Quando um publicador publica uma informação, os consumidores subscritos recebem-na através do barramento.

Outro aspecto relevante para IoT é que as aplicações possam buscar por serviços de acordo com critérios de qualidade desejados. Em computação ubíqua, a qualidade da informação (QoI), do serviço de distribuição (QoS) e do dispositivo que a provê (QoD) podem ser caracterizadas por um conjunto de parâmetros ou métricas conhecidos na literatura como Qualidade de Contexto (QoC). A noção de QoC inclui a qualidade da informação (ex: temperatura), do serviço de distribuição dos dados (ex: prazo e confiabilidade da entrega), e a característica dos sensores (ex: acurácia e intervalo de medição) [9, 10, 11].

Corroborando com a problemática do provisionamento de QoC e com o

processo de descoberta de serviços em IoT que leva em consideração os requisitos de QoC envolvidos, o Laboratório de Sistemas Distribuídos Inteligentes (LSDi) da Universidade Federal do Maranhão (UFMA), em parceria com o Laboratory for Advanced Collaboration (LAC) da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), vem desenvolvendo um middleware para descoberta e gerenciamento de contexto em IoT, denominado M-Hub/CDDL (*Mobile Hub e Context Data Distribution Layer*). Nesse middleware, o M-Hub [5] é responsável pela descoberta física, conexão e comunicação com objetos inteligentes utilizando alguma tecnologia WPAN (*Wireless Personal Area Network*), como o *Bluetooth Low Energy* (BLE), por exemplo. O CDDL é responsável pelo fornecimento do serviço de descoberta de serviços, publicação, e assinatura dos dados de contexto, de forma que sejam satisfeitas as necessidades de QoC da aplicação cliente. O M-Hub e o CDDL serão descrito em mais detalhes na Seção 2.4 do Capítulo 2.

1.1 Definição do Problema

IoT é visionada como um paradigma capaz de mudar radicalmente o modo como interagimos em sociedade e com o ambiente ao nosso redor, seja em casa, ou em outras localidades de nossa vida cotidiana. Ao mesmo tempo em que IoT proporciona um enorme potencial para aplicações que facilitariam diversas tarefas do dia-a-dia das pessoas, sejam elas coletivas ou individuais, diversos desafios ainda são uma barreira para o uso global e em larga escala dos diversos objetos capazes de sensoriar ou atuar no ambiente [3]. Um destes desafios é o problema de encontrar, configurar e utilizar objetos inteligentes de maneira que sejam disponibilizados como serviços transparentes às aplicações. Além disso, em cenários de mobilidade irrestrita, onde os *gateways* responsáveis pela descoberta de objetos inteligentes e os próprios objetos, podem ambos serem móveis, devem existir mecanismos que continuamente descubram e tornem disponíveis os serviços de objetos inteligentes para aplicações. Estes mecanismos devem tratar da comunicação com objetos que se conectam e desconectam a todo momento do *gateway*, devido ao raio de alcance da tecnologia WPAN utilizada na comunicação.

A descoberta contínua deve levar em consideração não somente os dados dos sensores e atuadores, mas também metadados de qualidade de contexto desejados por aplicações cliente. Para uma descoberta efetiva de serviços de objetos inteligentes com atributos e metadados diversos, uma linguagem estruturada aos moldes do SQL (*Structured Query Language*) proporciona um mecanismo para a criação de consultas expressivas que permitem selecionar e

efetivamente utilizar os serviços.

Tendo em vista os desafios presentes no paradigma de IoT, além dos aspectos desejáveis para um mecanismo de descoberta de serviços contínuo que leva em consideração as características de QoC dos serviços [12, 1], este trabalho propõe uma extensão ao M-Hub/CDDL para o suporte à descoberta contínua de serviços. Este aspecto consiste no registro do interesse de uma aplicação cliente em um determinado conjunto de serviços incluindo possíveis critérios de QoC, para que seja notificada de forma assíncrona assim que novos serviços estejam disponíveis.

Um mecanismo para descoberta de serviços contínuo auxilia a construção de aplicações oportunísticas em IoT, como o exemplificado na aplicação para estacionamentos inteligentes descrita na Introdução desta dissertação. A complexidade de descoberta, configuração e exposição das interfaces de acesso aos serviços dos objetos inteligentes deve ficar a cargo de um *middleware* conjugado a um *gateway* que tratarão destes aspectos. Os desenvolvedores de aplicações podem utilizar este "*middleware-gateway*" para selecionar e compor os serviços desejados.

Definir um mecanismo para descoberta de serviços contínuo em IoT proporciona alguns desafios de integração de tecnologias, uma vez que não existem padrões largamente estabelecidos [1, 12, 13]. Visto que o paradigma *Publish-Subscribe* [8] permite a criação de aplicações distribuídas desacopladas e assíncronas, e que o Processamento de Eventos Complexos é uma tecnologia que permite a criação de consultas contínuas sobre um fluxo de eventos com processamento em quase tempo-real, esta dissertação investiga as seguintes questões:

- É plausível criar um mecanismo para descoberta contínua e oportunística de serviços em ambientes de IoT com mobilidade irrestrita?
- É possível aliar os conceitos de um *middleware* de comunicação *Publish-Subscribe* com um mecanismo para a escrita de *queries* (consultas) que permitam uma descoberta contínua de serviços em IoT?

1.2 Objetivos

O principal objetivo desta dissertação é responder aos questionamentos da Seção 1.1, para os quais estendemos o suporte do *middleware* M-Hub/CDDL para permitir a descoberta contínua de serviços em IoT. Propomos aqui uma abordagem construída em plataforma Android, que utiliza MQTT (Message Queuing Telemetry Transport) como protocolo de comunicação *Publish-Subscribe* e CEP para a elaboração de consultas contínuas. As consultas aceitam

parâmetros de qualidade de contexto, e permitem a descoberta contínua e oportunística de serviços em ambientes IoT com mobilidade. O M-Hub/CDDL proporciona uma API para a construção de aplicações e esta foi estendida neste trabalho. Ela permite a implementação de *Listeners* de eventos que proporcionam um canal de subscrição de onde as notificações de serviços detectados pelas consultas são enviadas para a aplicação. Desta forma, as consultas por serviços passam a ser continuamente respondidas, estendendo a visão do padrão *Request-Response* para um ambiente de *Publish-Subscribe*.

1.3 Contribuições

As principais contribuições desta dissertação são:

- A definição de uma arquitetura assíncrona, contínua e oportunística para descoberta de serviços em IoT.
- Uma API para descoberta contínua de serviços em IoT, que leva em consideração características de QoC expressas através de consultas CEP.

1.4 Metodologia

Um estudo inicial sobre descoberta de serviços em IoT foi feito para elencar trabalhos relacionados e o estado da arte na abordagem. O estudo ajudou a compreender os desafios e limitações existentes em ambientes IoT com mobilidade, e para os quais um mecanismo de descoberta de serviços contínuo apoiaria a construção de aplicações com estas características. Dada a natureza desta dissertação, a avaliação conduzida mostra um estudo de caso de uma aplicação hipotética que suporte o problema de estacionamentos inteligentes utilizando a API estendida do M-Hub/CDDL.

1.5 Organização da Dissertação

A sequência desta dissertação está organizada da seguinte forma:

- **Capítulo 2 - Conceitos Fundamentais**: Neste capítulo são apresentados conceitos e tecnologias que foram utilizados na construção deste trabalho.
- **Capítulo 3 - Extensão do M-Hub/CDDL**: Neste capítulo a solução abordada neste trabalho é apresentada e discutida.
- **Capítulo 4 - API para Descoberta Contínua de Serviços**: São apresentados os principais métodos da API desenvolvida.

- **Capítulo 5 - Avaliação**: Aqui é discutida a implementação de um estudo de caso sobre estacionamentos inteligentes utilizando a API proposta nesta dissertação.
- **Capítulo 6 - Trabalhos Relacionados**: Neste capítulo, alguns trabalhos relacionados à descoberta de serviços em IoT são apresentados e discutidos.
- **Capítulo 7 - Conclusões e Trabalhos Futuros**: O trabalho é sumarizado e pesquisas futuras são sugeridas.

2
Conceitos Fundamentais

Este capítulo apresenta os conceitos fundamentais e tecnologias utilizadas nesta dissertação. Na Seção 2.1 será apresentado o conceito de descoberta de serviços e como ela é aplicada num contexto não-IoT. Na Seção 2.2 será apresentado o paradigma *Publish-Subscribe* (Publicação-Subscrição) bem como a tecnologia MQTT, que foi utilizada neste trabalho. Em seguida a Seção 2.3 apresenta o Processamento de Eventos Complexos (CEP), que foi o método utilizado neste trabalho para elaborar consultas para o mecanismo de descoberta de serviços contínuo. Por último, na Seção 2.4 será apresentado o *middleware* M-Hub/CDDL, que foi estendido para o suporte à descoberta contínua de serviços neste trabalho.

2.1
Descoberta de Serviços

Tradicionalmente, um paradigma de serviços se refere a uma abordagem onde os recursos oferecidos por um dispositivo (e.g. uma impressora, um *scanner*) podem ser utilizados de forma transparente. Isto significa que suas interfaces de comunicação são expostas de maneira que os detalhes de configuração ficam escondidos da aplicação [14]. Um exemplo clássico é o de uma impressora instalada em uma rede. Ao se conectar na rede, ela anuncia seus atributos (e.g. nome, formatos suportados, suporte a cores) através de uma URL (e.g. service:printer://printer). Uma aplicação (e.g. editor de textos, *browser*, etc.) que deseja utilizar o serviço de impressão dessa impressora, executará uma chamada através do protocolo de descoberta de serviços para obter acesso ao recurso e efetuar a impressão desejada.

Um mecanismo de descoberta de serviços permite que dispositivos sejam descobertos, configurados e se comuniquem corretamente com aplicações. Deste modo, os protocolos de descoberta de serviços são uma maneira de minimizar o *overhead* (sobrecarga) de administração e aumentar a usabilidade [13]. Protocolos para descoberta provêm mecanismos para a descoberta dinâmica dos serviços disponíveis em uma rede além de fornecer meios para: (i) pesquisar serviços; (ii) selecionar o serviços com as características desejadas; (iii) utilizar o serviço [14]. Do ponto de vista de uma aplicação, a descoberta de serviços

simplifica a tarefa de encontrar e utilizar serviços. Do ponto de vista da administração da rede, a tarefa de introduzir novos dispositivos fica facilitada, pois haverá uma autoconfiguração para que eles possam anunciar seus serviços. Quando há uma desconexão do dispositivo, no momento em que este se reconecta o processo de autoconfiguração e anúncio continuará automático.

Existem dois tipos básicos de arquitetura de descoberta de serviços: centralizada e descentralizada [15]. Na centralizada, a exemplo do *Service Location Protocol*[1] (SLP) e do Apache River[2] (anteriormente conhecido como Jini), existe um elemento denominado de serviço de *Lookup* (Pesquisa). Ele é basicamente um diretório onde os serviços se registram e podem ser consultados por outros elementos da rede. Nas arquiteturas descentralizadas, a exemplo do *Universal Plug and Play*[3] (UPnP) que utiliza o protocolo SSDP (*Simple Service Discovery Protocol*), um dispositivo que provê serviços faz anúncio de seus atributos através de um endereço de IP *multicast* da rede. Um cliente que procura por um serviço específico faz uma busca no endereço de *multicast* pré-determinado e os provedores de serviço respondem usando mensagens de *unicast*.

O conceito descrito acima se refere à descoberta e utilização de serviços no âmbito de dispositivos físicos. Entretanto, serviços não se limitam a este domínio apenas. No contexto de arquiteturas de software, existe o paradigma de *Service-oriented Architecture* (SOA, Arquitetura orientada a Serviços) [16]. O principal conceito aplicado por SOA é que as funcionalidades implementadas em aplicações são fornecidas como serviços, favorecendo à criação de componentes de software desacoplados e mais flexíveis. Em particular, uma abordagem extensivamente utilizada na construção de aplicações hoje em dia é a de *Web Services* (Serviços Web). Esta abordagem define um *framework* (arcabouço) extensível para a interação entre aplicações, utilizando a Internet e protocolos web (e.g. HTTP, XML) [17]. Para ilustrar o uso de *Web Services*, vamos tomar o exemplo de um sistema de comércio eletrônico. Para que o cliente efetue os pagamentos, existe um *Web Service* que se comunica com o sistema da operadora de cartão de crédito a fim de validar e efetuar a transação. Tudo isso é feito apenas se conhecendo as interfaces de comunicação, e todo o processamento interno efetuado pela operadora do cartão é transparente para o sistema do site de comércio eletrônico.

Analogamente ao que acontece com serviços em dispositivos físicos, existe um mecanismo padrão para descrever, descobrir e utilizar os *Web Services*. Uma especificação de *Web Service* conhecida é o SOAP (*Simple Object Access*

[1] RFC 2165 - `https://tools.ietf.org/html/rfc2165`
[2] Apache River - `https://river.apache.org/`
[3] ISO/IEC 29341-1:2011 - `https://www.iso.org/standard/57195.html`

Tabela 2.1: Requisitos em descoberta de serviços para IoT (Adaptado de Gomes *et al.* [1])

Requisito	Descrição
Esquema de identificação flexível	Transparência no esquema de identificação utilizado para identificar os recursos (e.g. URI, endereço IP, etc.).
Queries (Consultas) multi-atributo	Capacidade de encontrar recursos utilizando consultas em um ou mais atributos (e.g. Nome do serviço, Nome do publicador do serviço, Localização, etc.).
Queries (Consultas) com intervalos de valores	Capacidade de encontrar recursos utilizando consultas com intervalos de valores em um ou mais atributos (e.g. Parâmetros de qualidade, etc.).
Lidar com múltiplos publicadores	Várias entidades podem produzir e publicar informações de um dado recurso.
APIs de gerenciamento	Entidades autorizadas devem poder adicionar, atualizar, ou remover informações associadas a um recurso.
Descrição semântica de recursos e serviços	Uso de ontologias para para modelar semanticamente os recursos expostos.
Modelar critérios de Qualidade de Contexto (QoC)	Permitir a modelagem de indicadores de QoC associados a um dado contexto de um recurso (e.g. Acurácia, confiabilidade, etc.)
Interações síncronas e assíncronas	Capacidade de responder consultas síncronas (do tipo *Request-Response*) ou assíncronas (do tipo *Publish-Subscribe*)

Protocol), que estabelece um protocolo para comunicação. No SOAP, a descrição dos serviços é feita utilizando WSDL (*Web Services Description Language*) e a descoberta fica a cargo do UDDI (Universal Description, Discovery, and Integration). O UDDI atua como um serviço de *Registry* (diretório de registro), similar ao que ocorre no protocolo SLP. Ele define a forma de se anunciar serviços e uma API de consulta e atualização do *Registry*. Em particular o UDDI codifica 3 tipos de "catálogos": (i) Páginas Brancas, que definem um nome e detalhes de contato do serviço; (ii) Páginas Amarelas, que incluem as categorias dos serviços; (iii) Páginas verdes, que definem os detalhes técnicos sobre os serviços, incluindo suas interfaces de acesso.

Em IoT, não existem padronizações estabelecidas para o uso de serviços e mecanismos de descoberta, muito embora diversos trabalhos proponham

novas abordagens com diferentes tecnologias [4, 18, 19, 12, 20, 21]. O termo "*Sensing as a service*"é utilizado em alguns trabalhos para descrever um tipo de plataforma IoT onde os dados de sensores são utilizados como serviços [22, 23].

Os trabalhos em descoberta de serviços para IoT lidam com diferentes desafios. Paganelli *et al.* [12] e Gomes *et al.* [1] definem um conjunto de requisitos para descoberta de serviços em cenário de IoT, que são apresentados na Tabela 2.1.

Em particular, o *middleware* M-Hub/CDDL, no qual este trabalho se baseia, implementa os requisitos apresentados na Tabela 2.1, exceto pelo uso de uma ontologia para a descrição semântica dos serviços. Este trabalho visa estender o M-Hub/CDDL para permitir a descoberta contínua (assíncrona) de serviços. A descoberta contínua permite que uma aplicação publique uma consulta no *middleware* e receba notificações em caso de serviços que se registrem e atendam aos critérios definidos. Este fator é particularmente importante em cenários onde uma aplicação pode se beneficiar de serviços que encontra oportunisticamente. Em outras palavras, quando a aplicação encontra serviços ao se mover e ficar no alcance de sensores e atuadores implantados, como no exemplo da descoberta de estacionamentos, da Figura 1.1 apresentada na Introdução.

2.2 O Paradigma Publish-Subscribe

Para reduzir a complexidade na criação de aplicações onde diversas entidades necessitam se comunicar, uma infraestrutura de *middleware* é necessária para prover um esquema de comunicação adequado [8]. Um paradigma de comunicação que apoia a construção de sistemas distribuídos assíncronos é o de *Publish-Subscribe* (Publicação-Subscrição). Ele é capaz de prover uma forma de comunicação desacoplada, onde produtores de informações enviam mensagem sem a necessidade de saber quem são os consumidores. Isso aumenta a escalabilidade em sistemas de grande porte (com milhares de nós), pois os elos entre produtores e consumidores não são estaticamente definidos.

Em *Publish-Subscribe*, existe o conceito de um barramento de serviços onde entidades denominadas de publicadores (produtores) publicam seus eventos e subscritores (consumidores) se subscrevem para receber eventos de interesse. Quando um evento ocorre no barramento e é do tipo ao qual um subscritor registrou seu interesse, ele o recebe através de uma notificação. O barramento é o canal de comunicação que media a interação entre produtores e consumidores. Ele é implementado por um *middleware*, responsável por lidar com o processo de troca mensagens, além de quesitos relacionados à rede, como tolerância a

falhas, garantia de entrega, dentre outros. Na Figura 2.1 é possível observar a interação entre os elementos de um sistema de *Publish-Subscribe*.

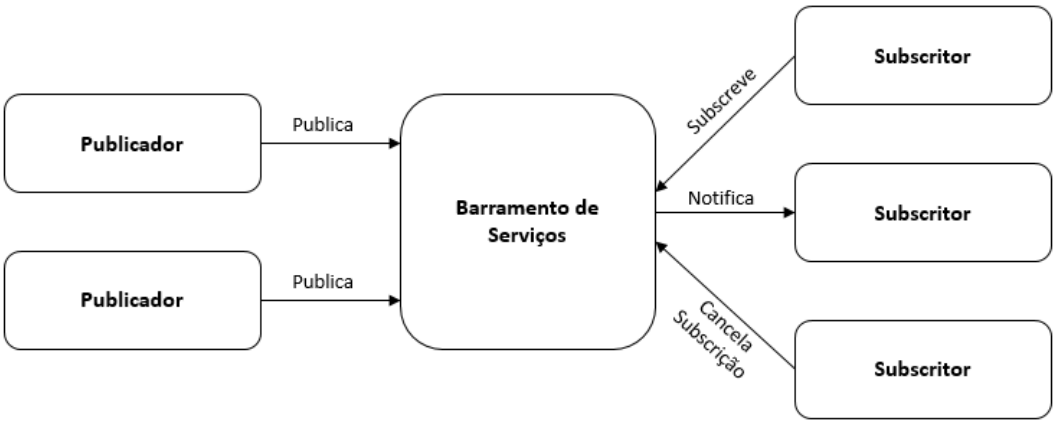


Figura 2.1: Um sistema *Publish-Subscribe* simples. Adaptado de Eugster el al. (2003) [8]

O desacoplamento que o barramento de serviços provê entre publicadores e subscritores é decomposto em três dimensões [8]:

- **Desacoplamento Espacial**: Publicadores e subscritores não têm conhecimento mútuo, nem guardam referências uns dos outros. O processo de comunicação entre eles é feito de forma indireta através do barramento.
- **Desacoplamento Temporal**: Publicadores e subscritores não necessitam participar da interação ao mesmo tempo. Um publicador pode enviar eventos enquanto não existem subscritores conectados. Reciprocamente, um subscritor pode ser notificado de um evento no momento em que o publicador original está desconectado.
- **Desacoplamento de Sincronização**: Publicadores e subscritores não bloqueiam sua execução durante o ciclo de vida da aplicação. A comunicação entre eles é assíncrona, ocorrendo através de *callbacks*.

Os sistemas de *Publish-Subscribe* podem implementar três tipos de esquemas de subscrição [8], a depender do modo como os subscritores especificam seu interesse em eventos: (i) Baseado em tópicos; (ii) Baseado em conteúdo; (iii) Baseado em tipo.

O primeiro esquema a surgir foi o baseado em tópicos. Este método classifica o conteúdo segundo palavras-chave. Publicadores e subscritores interagem através de tópicos individuais, num conceito que estende a noção de comunicação em grupo [24]. Essa similaridade se dá pelo fato de que quando um publicador publica um evento em um tópico T, este evento será transmitido aos subscritores do tópico T, como se eles pertencessem a um mesmo grupo de

comunicação. Cada tópico mapeia um canal de comunicação distinto e separado de outros tópicos dentro do barramento de serviço, reforçando o desacoplamento. Um tópico é escrito em uma notação de URL (*Uniform Resource Locator*) hierárquica, o que permite o uso de *wildcards* (curingas) para pareamento em diversos níveis dentro de uma árvore de hierarquia. Por exemplo, um tópico "*mhubcddl/publisher01/sensor_data*" mapeia todos os eventos de dados de sensores dentro de uma instância do M-Hub/CDDL que foram publicados pelo publicador *publisher01*. Uma subscrição no tópico "*mhubcddl/#/sensor_data*" recebe os eventos de dados de sensor, de todos os publicadores, por causa da adição da *wildcard* '#' na parte referente ao publicador da URL em questão.

Apesar de simplificar o desenvolvimento do *middleware* de comunicação, a abordagem baseada em tópicos é limitada em expressividade, pois não permite um mecanismo para a criação de consultas [25]. Além disso, os tópicos são estaticamente definidos na inicialização do sistema. Em contrapartida, o esquema de subscrição por conteúdo permite consultas em atributos internos dos eventos. Os subscritores se subscrevem em eventos utilizando filtros na forma "*chave-valor*" utilizando alguns operadores de comparação (=, $<$, $>$, $\leq$, $\geq$). Estes pares "*chave-valor*" podem ser combinados utilizando operadores lógicos (e.g. AND, OR, NOT, etc.) para a composição de filtros mais expressivos. Por exemplo, um subscritor poderia utilizar o filtro "*sensor_name == 'temperature' AND publisherID == 'publisher01'* para se subscrever em sensores de temperatura do *publisher01*. Além da expressividade adquirida, esse aspecto ajuda a economizar o tráfego de dados na rede devido ao maior refinamento do conteúdo antes de ser enviado. A grande desvantagem dessa abordagem está na complexidade de se desenvolver um esquema eficiente de *matching* (correspondência) entre publicadores e subscritores. O número de subscrições únicas é de maior ordem de magnitude do que os grupos que seriam formados num sistema baseado em tópicos.

O esquema de subscrição baseado em tipo permite uma maior aproximação entre o *middleware* de comunicação e a linguagem de programação utilizada [8]. Os eventos são mapeados em objetos da linguagem, sem necessidade de *typecast*. Nesse sentido, a subscrição baseada em tipos é uma extensão natural da subscrição baseada em conteúdo, pois reforça o encapsulamento dos eventos em objetos da linguagem. Por exemplo, dado um objeto do tipo *SensorData*, com atributos *sensor_name* e *publisherID*, o subscritor poderia instanciar uma consulta "*sensorData.getSensor_Name == 'temperature' && sensorData.getPublisherID == 'publisher01'*", para subscrever-se a dados de sensor de temperatura do *publisher01*. Apesar de ser um esquema com uma boa "amarração", pois há checagem de tipos, uma grande desvantagem desse

esquema reside no fato de não ser facilmente interoperável com plataformas escritas em linguagens diferentes. Nesse esquema os canais de comunicação são separados pelos tipos de eventos, o que induz complexidade na modelagem de tipos e subtipos (classes e subclasses, numa linguagem orientada a objetos).

Neste trabalho optamos pela simplicidade do esquema baseado em tópicos e adicionamos uma camada de consulta utilizando a expressividade do mecanismo CEP (*Complex Event Processing*), que será detalhado na Seção 2.3. A implementação escolhida foi o MQTT[4] (*Message Queue Telemetry Transport*), que atualmente é um protocolo padrão definido pela OASIS (*Organization for the Advancement of Structured Information Standards*) para comunicação em IoT. Ele é considerado um protocolo leve e simples, por ter um baixo *overhead* de comunicação. Por este fator é adequado aos ambientes limitados de IoT [26].

2.3 Complex Event Processing

Complex Event Processing (CEP, Processamento de Eventos Complexos) é uma tecnologia para processamento dinâmico de fluxos de dados de eventos em quase tempo-real. Foi proposta em meados dos anos 1990 por David Luckham [27]. Em contraste ao paradigma de um SGBD (Sistema de Gerenciamento de Banco de Dados), onde os dados são armazenados para uma consulta posterior, CEP armazena consultas contínuas e executa um fluxo de dados sobre elas. Em outras palavras, CEP permite a análise contínua de um fluxo de dados de eventos através de consultas armazenadas.

Eventos são a base do paradigma CEP. Um evento inserido numa máquina de processamento CEP é um modelo de algo que acontece no mundo real. Nesse contexto, os eventos são entidades criadas por elementos produtores que modelam ocorrências de interesse para aplicações. São exemplos de eventos: dados coletados de sensores, ações de uma companhia no mercado financeiro, dentre outros. Um evento pode mapear um objeto de uma classe em uma linguagem de programação orientada a objetos, incluindo seus atributos. Por exemplo, um evento pode mapear um objeto de uma classe chamada *SensorData*, que representa dados de sensores. Esse evento mapeia também seus atributos, tais como: nome do sensor, valor medido, *timestamp* da medição, latitude e longitude.

Um fluxo de eventos dentro de uma máquina CEP é uma sequência de eventos criados e enviados por elementos produtores [27]. Os eventos são processados de acordo com consultas definidas e o resultado é enviado para elementos consumidores. A Figura 2.2 apresenta a arquitetura básica de

[4]MQTT - `http://mqtt.org/`

um sistema CEP que define este fluxo. As consultas são definidas segundo uma linguagem denominada CQL (*Continuous Query Language*), que é uma linguagem declarativa semelhante a SQLs (*Sctructured Query Language*) de bancos de dados relacionais.

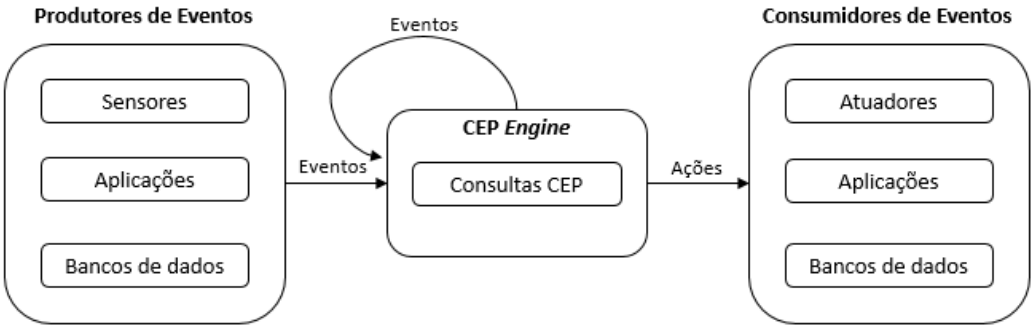


Figura 2.2: Arquitetura de um sistema CEP.

Os eventos em CEP são denominados complexos, pois o paradigma permite que eles possam produzir outros eventos a serem reinseridos na máquina CEP. Esse mecanismo permite a composição de uma hierarquia entre eventos, no qual eventos intermediários podem ser usados para definir outros eventos complexos de mais alto nível. Por exemplo, um evento denominado *Alarme de incêndio* pode ser construído a partir de eventos *Presença de Fumaça* e *Temperatura Alta*.

Cada consulta CEP é executada por um *Event Processing Agent* (EPA, Agente de Processamento de Eventos). Um EPA continuamente reage a eventos de entrada, produzindo eventos de saída para elementos consumidores, que podem ser outros EPAs ou *endpoints* em aplicações. Um conjunto de EPAs interconectados é chamado de *Event Processing Network* (EPN, Rede de Processamento de Eventos). Uma EPN possui uma topologia de grafo direcionado, o que facilita a distribuição de EPAs em máquinas diferentes para compor uma lógica de processamento distribuído.

Uma consulta CEP é escrita seguindo uma implementação de um formalismo denominado CQL. Por exemplo, a implementação *open-source* Esper[5], define uma linguagem denominada EPL (*Event Processing Language*). Para ilustrar a expressividade de da linguagem, uma consulta escrita em EPL é apresentada no Código 2.1. Essa consulta procura por eventos de sensor de temperatura dentro de um intervalo de latitude e longitude, contidas numa janela de tempo de 5 segundos.

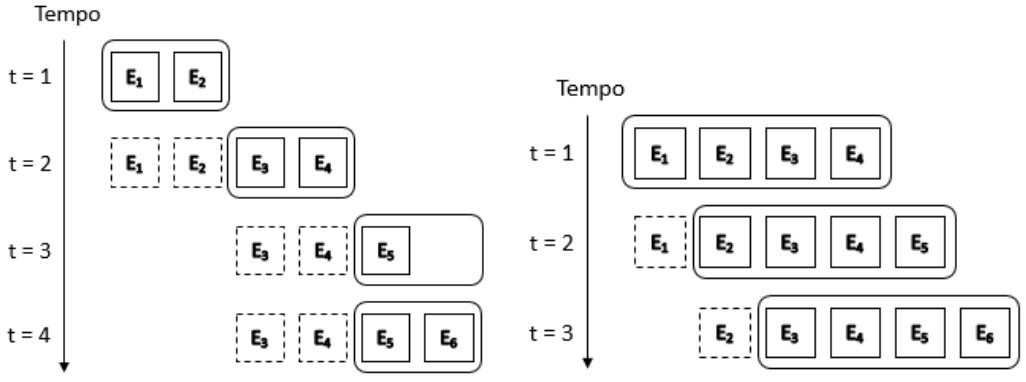
Código 2.1: Exemplo de consulta CEP escrita em EPL.

[5]Esper - `http://www.espertech.com/`

```
1 SELECT *
2 FROM SensorData.win:time(5 s)
3 WHERE sensorName LIKE '%temperature%' AND
4       latitude > -22.0 AND latitude < -21.0 AND
5       longitude > -43.0 AND longitude < -42.0
```

A linguagem CQL proporciona um conceito inerente ao domínio de processamento de eventos, que são as *Windows* (Janelas). As janelas CEP permitem definir um escopo de busca em um fluxo de eventos. Ele consiste num intervalo definido em cima do fluxo, que pode ser de tempo ou de quantidade de eventos. As janelas são particularmente úteis em consultas que detectam a negação (não presença de um evento) ou a agregação (e.g. média de valores, soma de valores), pois num fluxo infinito de eventos elas só poderiam ser respondidas depois que o fluxo encerrasse [28]. Com o uso de janelas, é possível detectar, por exemplo, se um determinado evento não ocorre dentro de um intervalo de tempo definido.

Existem dois tipos de janelas em CEP, que são classificadas de acordo como elas se movem em cima do fluxo de eventos. As janelas de processamento em lote (*batch windows*) têm tamanho fixo e aguardam até que n unidades de tempo ou n eventos ocorram para processar de uma vez só. Um exemplo desse tipo de janela é apresentado na Figura 2.3 (a), onde a janela espera pela ocorrência de 2 eventos para processar o resultado. Na Figura 2.3 (b) é apresentada uma janela deslizante, onde a cada 1 segundo os eventos que aconteceram entre o instante t e $t-1$ são processados.



(a) Janela em Lote. (b) Janela Deslizante.

Figura 2.3: Exemplos de Janelas em Lote e Deslizantes.

Existem diversas implementações de CEP disponíveis, tais como Esper [29], Microsoft StreamInsight [30], Apache Flink [31]. Neste trabalho escolhemos o Esper por ser *open-source* e distribuída sob licença GNU GPL. Os eventos em

Esper seguem um padrão de orientação a objetos com tipos dinâmicos de dados, o que permite facilmente mapear um evento em um objeto de uma linguagem como Java, por exemplo. Além disso Esper tem uma implementação Android disponível, denominada Asper [32], que permite o uso de CEP em *smartphones*.

CEP oferece todas as características desejáveis no que se refere à criação de consultas para descoberta contínua de serviços. A linguagem EPL possui grande expressividade para a elaboração das consultas necessárias nesse contexto. Além disso, a capacidade do CEP de processar eventos em quase tempo-real contribui para o mecanismo de descoberta oportunístico que desejamos neste trabalho.

2.4 O Middleware M-Hub/CDDL

O M-Hub/CDDL (*Mobile Hub / Context Data Distribution Layer*) é um *middleware* e *gateway* para IoT. A Figura 2.4 apresenta um cenário de uso do M-Hub/CDDL. Nesse cenário o Mobile Hub atua como o mecanismo de descoberta física de sensores e atuadores através de alguma tecnologia WPAN (*Wireless Personal Area Network*) como *Bluetooth* ou *ZigBee*, além de promover a interação com estes objetos. Nesse contexto o M-Hub implementa as funções de um *gateway* de IoT, visto que permite intermediar a comunicação entre objetos inteligentes e outros nós computacionais ou aplicações locais. Para isso ele pode utilizar redes de longo alcance como Wi-Fi ou redes celulares.

O CDDL é o *middleware* responsável por promover um mecanismo de comunicação *Publish-Subscribe* baseado em tópicos. Ele media a comunicação entre aplicações consumidoras e serviços de objetos inteligentes que foram descobertos pelo M-Hub. O CDDL estende o M-Hub permitindo a publicação de dados de contexto juntamente com parâmetros de QoC (*Quality of Context*) calculados. Também fornece recursos que permitem que aplicações consumidoras descubram serviços de objetos inteligentes que satisfaçam critérios de QoC. Por fim, o CDDL também permite o monitoramento de parâmetros de QoC associados aos dados de serviços que estão sendo consumidos.

O M-Hub/CDDL disponibiliza uma API para a construção de aplicações de IoT que necessitam de descoberta de serviços através de consultas contendo parâmetros de QoC desejáveis. Desta forma, as aplicações consumidoras são capazes de descobrir os sensores e atuadores disponíveis, consumir dados e enviar comandos a eles.

A Figura 2.5 apresenta os componentes do middleware M-Hub/CDDL. Nessa arquitetura, o S2PA (Short-Range Sensor, Presence and Actuation Service, Serviço de atuação e presença em sensores de curto alcance) é o componente do M-Hub [5] responsável pela descoberta, conexão e comunicação

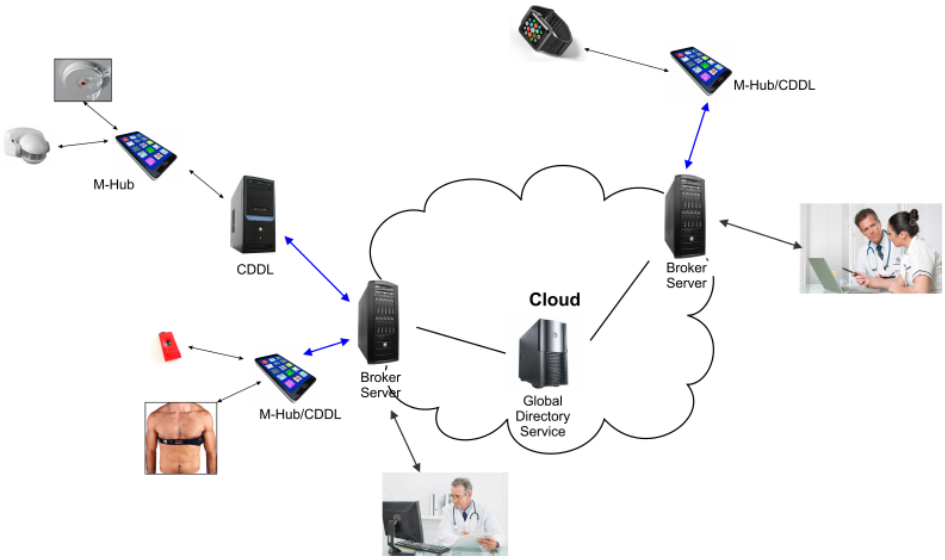


Figura 2.4: Visão geral da estrutura do M-Hub/CDDL.

com sensores e atuadores. A fim de lidar com a diversidade de tecnologias WPAN utilizadas por diferentes sensores e atuadores, o S2PA define uma API (Application Programming Interface) genérica chamada Technology, que fornece uma abstração uniforme para lidar com a comunicação com os objetos, independente da tecnologia de comunicação utilizada para este fim. Essa API requer a implementação de um conjunto de métodos que abrange as operações necessárias suportadas por cada objeto (ex: ativação/desativação da interface de rádio, busca de dispositivos, descoberta de serviços, estabelecimento de links de comunicação, etc).

Os dados recebidos pelo S2PA são encaminhados ao componente QoC Evaluator, responsável pelo cálculo, sempre que possível, da QoC de cada amostra de dados gerada pelos sensores. Esta QoC pode ser usada por aplicações consumidoras para a escolha de fontes de dados de contexto que atendam aos requisitos especificados.

O CDDL utiliza dois tipos de *broker* para mediar a comunicação entre produtores e consumidores: um *micro-broker*, que executa no dispositivo (local), e um *server-broker*, que executa na nuvem CDDL. O *micro-broker* é responsável pela distribuição de dados de contexto para aplicações locais, enquanto que o *server-broker* é responsável por distribuir os dados para aplicações remotas.

O *Local Directory Service* e o *Global Directory Service* são os componentes responsáveis pela manutenção das bases de dados de provedores de serviços disponíveis. O *Local Directory Service* mantém um registro de todos os provedores de serviços disponíveis em seu domínio, ou seja, que são acessíveis através do alcance de redes WPAN. Para manter esta base de dados, o *QoC*

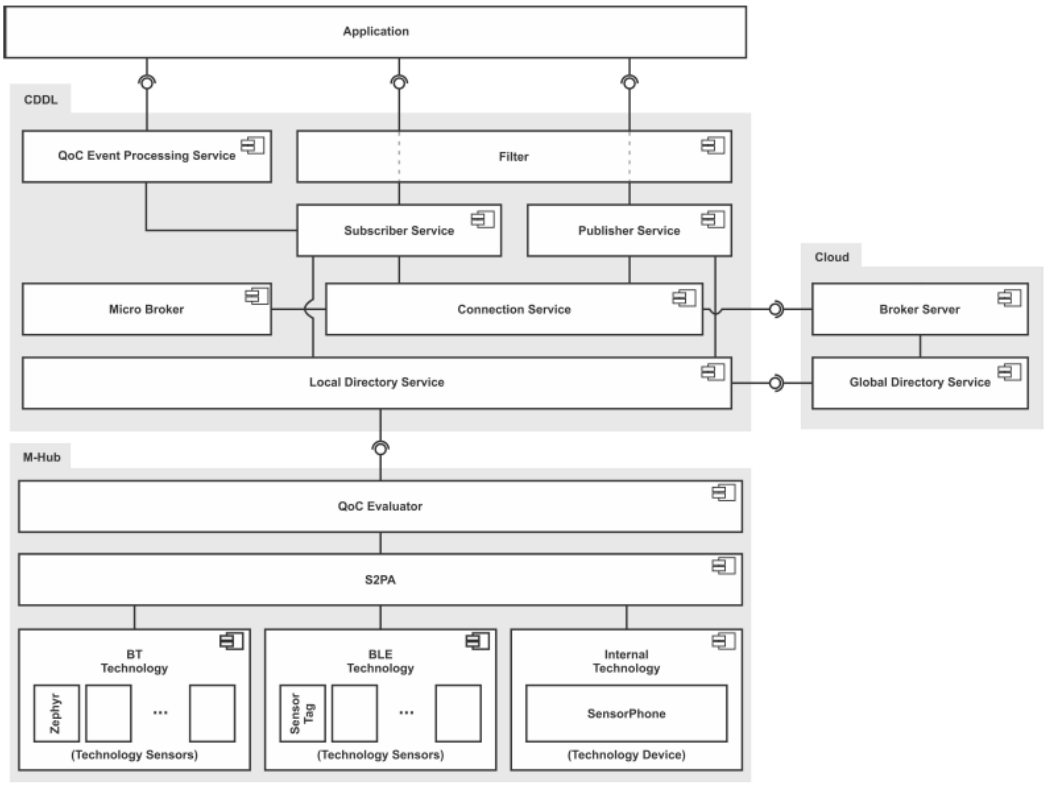


Figura 2.5: Arquitetura do middleware M-Hub/CDDL.

Evaluator encaminha periodicamente informações sobre os sensores disponíveis e suas qualidades para o diretório. O *Local Directory Service* envia periodicamente seus registros para o *Global Directory Service* em execução na nuvem, que mantém uma base de dados de todos os provedores de serviços disponíveis através de M-Hub/CDDLs registrados.

O *Publisher Service* é o componente responsável pela publicação de dados tanto a nível local quanto na nuvem CDDL. Através dele podem ser publicados dados de sensores (recebidos do *S2PA*) e notificações de descoberta de serviços (recebidas do *Local Directory Service*). Uma aplicação também pode usar o *Publisher Service* para publicar seus próprios dados de contexto, quando necessário. O *Subscriber Service* é o componente usado pela aplicação para assinar tópicos de interesse e assim receber os dados publicados. Alguns tópicos utilizados pelo *middleware* já são previamente conhecidos pelas aplicações, tais como os utilizados para a descoberta de provedores de serviços.

O *Publisher Service* e o *Subscriber Service* se conectam a um ou mais *brokers* utilizando o *Connection Service*. Ele usa um cliente MQTT, na implementação Paho[6], para solicitar aos *brokers* a execução de operações relacionadas à rede (e.g. conexão, desconexão, sessão, assinatura de tópicos, envio e recebimento de dados, confirmação de leitura, retransmissão).

[6]MQTT Paho - `http://www.eclipse.org/paho/`

O componente *Filter* é utilizado pela aplicação para especificar regras que determinam quais dados de contexto e/ou anúncios de serviços podem de fato ser publicados pelo *Publisher Service*, bem como para definir quais dados e/ou notificações de descoberta de serviços recebidos pelo *Subscriber Service* devem ser realmente encaminhados para a aplicação consumidora. No *Publisher Service*, o filtro pode, por exemplo, impedir a publicação da localização de determinadas coordenadas em que um usuário de *smartphone* não deseja ser rastreado. Além disso, pode ajudar a diminuir o volume de dados a ser transmitido. No *Subscriber Service*, um filtro pode ajudar reduzir o volume de dados a ser repassado para a camada de aplicação. Por exemplo, se a aplicação consome dados de localização, mas deseja ser notificada apenas quando as coordenadas correspondem a um lugar específico, então ela pode especificar um filtro que não repassará dados que indiquem localizações que não interessam. O mecanismo de filtragem é especificado por meio de regras CEP escritas utilizando-se EPL (*Event Processing Language*). O *Filter* também tem um papel importante no provisionamento de QoC, especialmente para o consumidor, que pode filtrar os dados considerando seus metadados de qualidade. Como exemplo, pode ser definida uma regra que filtra um fluxo de dados de localização, exigindo uma acurácia inferior a 5 metros (alta precisão).

O componente *QoC Event Processing Service* permite a especificação de regras CEP que ativam notificações de eventos quando são detectadas variações significativas na QoC fornecida pelos serviços. A especificação das regras também é feita em EPL. Quando eventos de variação de QoC são detectados, a aplicação consumidora é notificada e pode reagir de acordo. Há várias razões para a detecção de eventos relacionados às variações de QoC. Por exemplo, se uma aplicação escolheu usar um GPS devido à sua alta precisão e depois de um tempo essa precisão é reduzida, a aplicação precisa ser notificada para selecionar outro provedor de serviços, uma vez que os dados fornecidos podem não atender aos requisitos de QoC necessários.

O M-Hub/CDDL é um *middleware* construído com foco em provisionamento e monitoramento de critérios de QoC providos por serviços de sensores. Para tal fim, ele implementa uma arquitetura baseada em *Publish-Subscribe* que provê uma API para descoberta de serviços. Critérios de QoC permitem uma maior robustez na busca por serviços que atendam a necessidades específicas de aplicações. Por exemplo, suponhamos que existe uma aplicação que consome dados de GPS de um usuário e ele possui dois dispositivos com GPS. A aplicação precisa que o GPS tenha uma acurácia dentro de um raio de 10 metros e indica isso através de sua consulta para descoberta de serviços. Um dos papeis do M-Hub/CDDL é o de selecionar o GPS que atende esse critério,

podendo ser inclusive ambos os GPS do usuário.

Este trabalho estende o M-Hub/CDDL para o suporte a descoberta contínua de serviços, através da junção entre o paradigma de *Publish-Subscribe* e a expressividade de consultas escritas em linguagem EPL da implementação Esper CEP. Com a descoberta contínua de serviços, o M-Hub/CDDL passa a suportar que aplicações com descoberta oportunística sejam implementadas através da API disponibilizada. Ambientes de mobilidade irrestrita, onde os dispositivos que descobrem e os dispositivos que são descobertos podem ser móveis, são os que mais se beneficiam deste tipo de descoberta. A descoberta é contínua pois apenas uma requisição de descoberta é feita através de uma consulta e ela é registrada no *middleware*. Enquanto os dispositivos se movem, a descoberta por provedores de serviços que atendem à consulta continua acontecendo.

3
Extensão do M-Hub/CDDL

Neste capítulo, apresentamos a extensão ao M-Hub/CDDL que foi desenvolvida para suportar a descoberta contínua de serviços. Essa abordagem permite a escrita de uma consulta em linguagem EPL (*Event Processing Language*) (linguagem da implementação Esper[1] do CEP, que foi utilizada neste trabalho), que é publicada no M-Hub/CDDL. Particularmente neste trabalho, foi utilizada uma versão do Esper para Android denominada Asper [32]. Através desse mecanismo o M-Hub/CDDL cria a subscrição necessária para que os *Listeners* de uma aplicação recebam notificações dos eventos de interesse, ou seja, os serviços que atendem à consulta que foi feita.

A ideia base desse mecanismo é combinar a detecção em fluxos de eventos proporcionada pelo CEP, e um mecanismo de comunicação *Publish-Subscribe* com base em tópicos. Nessa arquitetura, quando objetos inteligentes são descobertos fisicamente, os metadados associados aos seus serviços são inseridos na máquina de regras CEP, constituindo um fluxo de eventos monitorável. Quando uma aplicação deseja fazer uma consulta por serviços de forma contínua, ela se subscreve em um tópico onde receberá as respostas. Em seguida ela envia sua consulta para ser inserida na máquina de regras CEP. Toda vez que um serviço atende a consulta CEP registrada, o M-Hub/CDDL publica as interfaces dos serviços encontrados no tópico assinado pela aplicação. Isso permite uma descoberta contínua e oportunística de serviços.

A grande utilidade deste mecanismo é que CEP proporciona uma linguagem com grande expressividade para elaborar consultas que podem envolver um ou mais atributos de qualidade de contexto (QoC), além de outros metadados relacionados aos serviços de objetos inteligentes (sensores). Além disso, CEP constitui uma tecnologia para processamento de fluxos de eventos com foco em processamento em quase tempo-real, o que contribui para um mecanismo de descoberta contínuo e oportunístico. Sempre que um novo serviço é descoberto fisicamente, o evento relacionado à descoberta é inserido no fluxo de processamento do CEP. As consultas de descoberta que ficam registradas no CEP recebem um *callback* em caso de correspondência com os eventos que foram inseridos e notificam seus consumidores.

[1]Esper - `http://www.espertech.com/`

A parte da comunicação implementada neste mecanismo fica a cargo do *middleware* de *Publish-Subscribe*, que neste caso é o MQTT. Como foi definido na Seção 2.2 do Capítulo 2, ele foi escolhido para este contexto por ser um dos padrões disponíveis para comunicação em ambientes IoT, definido pela OASIS. Utilizando este mecanismo de comunicação, as aplicações subscritoras ficam desacopladas dos publicadores de eventos. Deste modo as aplicações recebem os eventos de notificação de uma consulta de descoberta contínua, escrita em CEP, que foi previamente registrada no M-Hub/CDDL.

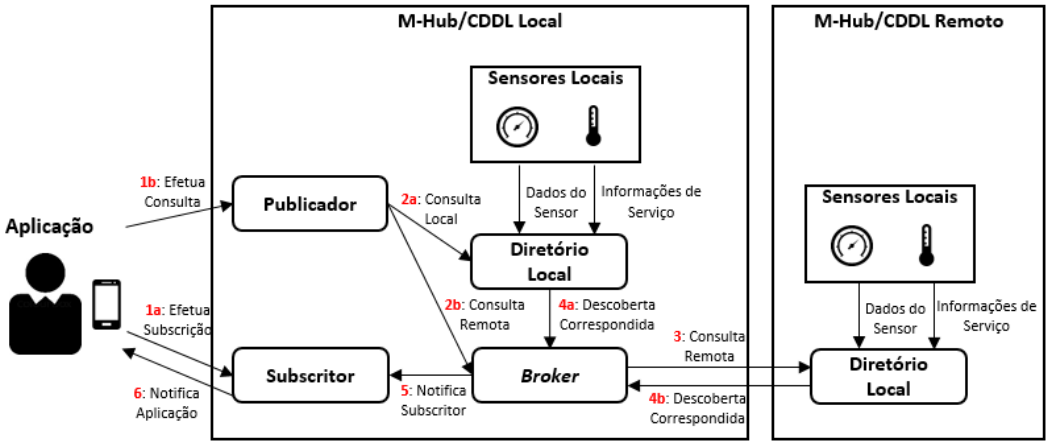


Figura 3.1: Fluxo de execução do mecanismo de descoberta contínuo em nível local e em nível global com publicação remota da consulta.

Ao contrário da abordagem onde uma consulta é do tipo *Request-Response*, ou seja, responde uma única vez, o mecanismo implementado neste trabalho permite que sempre que houverem novos serviços descobertos que atendem à consulta publicada, a aplicação recebe uma notificação através de um *Listener*. Ao recebe-la a aplicação poderá se subscrever para de fato utilizar o serviço.

No M-Hub/CDDL, o processo de descoberta de serviços é separado do processo de utilização dos serviços. Isso acontece porque dentro da máquina de processamento CEP, as informações relacionadas ao serviço são separadas dos dados que constituem os valores de sensores subjacentes. Essa separação também ocorre no processo de comunicação via *Publish-Subscribe*, onde existem tópicos para tráfego de dados de serviços, e outros tópicos para dados de sensores. Esse aspecto é demonstrado na Figura 3.1.

O M-Hub/CDDL permite que uma consulta de descoberta seja local e/ou global. No modo local, apenas os dispositivos descobertos no raio de alcance da tecnologia WPAN (e.g. BLE, ZigBee) do dispositivo hospedeiro são considerados no processo. No modo global a aplicação pode submeter a sua consulta para outros dispositivos que tenham o M-Hub/CDDL executando, sendo eles *smartphones* ou uma nuvem. No caso de outros *smartphones* eles devem ter

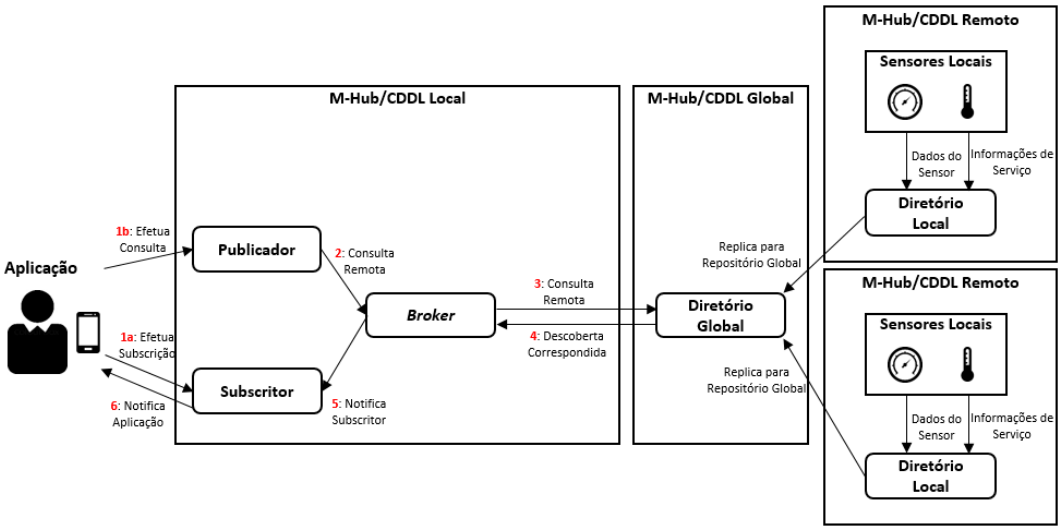


Figura 3.2: Fluxo de execução do mecanismo de descoberta contínuo utilizando um repositório global.

aceitado o modo de descoberta global em suas configurações, permitindo assim registrar consultas remotas. Neste caso, a consulta é distribuída para outros participantes e registrada em seus respectivos diretórios locais. No caso de uma nuvem, ela implementa um repositório global, que acumula serviços descobertos em diversos dispositivos registrados. O modo de descoberta utilizando nuvem é apresentado na Figura 3.2. Este modo de descoberta utilizando uma nuvem não foi implementado neste trabalho.

Na Figura 3.1 é demonstrado o fluxo de execução onde a consulta é publicada localmente e também em outro dispositivo remoto. Ela segue os seguintes passos:

- 1a: A aplicação efetua uma subscrição para recebimento de notificações de descoberta de serviços.
- 1b: A aplicação publica uma consulta para descoberta contínua de serviços incluindo os critérios desejados.
- 2a: A consulta é instanciada no diretório local.
- 2b: A consulta é publicada no *broker*, a fim de ser enviada para outras instâncias de M-Hub/CDDL.
- 3: A consulta é enviada pela rede para um M-Hub/CDDL remoto e é instanciada nos diretórios locais desses dispositivos.
- 4a: Quando um serviço local atende aos critérios especificados pela consulta, o diretório local emite uma notificação ao *broker*.
- 4b: Quando um serviço remoto atende aos critérios da consulta, o diretório local do M-Hub/CDDL remoto notifica o *broker* do M-Hub/CDDL que requisitou a consulta.

– 5: O *broker* notifica o subscritor da aplicação.

– 6: O subscritor notifica a aplicação.

Analogamente, a Figura 3.2 apresenta o fluxo de execução onde existe um repositório global de serviços. A descoberta segue os seguintes passos:

– 1a: A aplicação efetua uma subscrição para recebimento de notificações de descoberta de serviços.

– 1b: A aplicação publica uma consulta para descoberta contínua de serviços incluindo os critérios desejados.

– 2: A consulta é publicada no *broker*, a fim de ser enviada para o repositório global.

– 3: A consulta é enviada pela rede para o M-Hub/CDDL que está executando o repositório global.

– 4: Quando um serviço remoto atende aos critérios da consulta, o repositório global notifica o *broker* do M-Hub/CDDL que requisitou a consulta.

– 5: O *broker* notifica o subscritor da aplicação.

– 6: O subscritor notifica a aplicação.

A Seção 3.1, a seguir, apresenta a implementação e a infraestrutura do mecanismo de descoberta contínua.

3.1 Infraestrutura para Descoberta Contínua

O processo de descoberta físico (no raio de alcance da tecnologia WPAN do *smartphone*) é apresentado através do diagrama de sequência na Figura 3.3. Quando o módulo **S2PA** do M-Hub/CDDL encontra fisicamente um novo objeto inteligente, ele descobre seus serviços e envia os metadados para o componente **QoCEvaluator**. No QoCEvaluator os parâmetros de QoC são avaliados, enriquecendo os dados com metadados de contexto. Essa informação é então inserida no fluxo de eventos de informação de serviço (**ServiceInformationMessage**) que é tratada no componente **LocalDirectory**. O **LocalDirectory** é responsável por implementar o motor de regras CEP.

A regra escrita em EPL que captura esse evento é mostrada no Código 3.1. Essa regra captura os eventos de novos serviços dentro de uma janela de tempo deslizante definida pela variável "TIME_WINDOW" do M-Hub/CDDL, cujo valor padrão é 5 segundos. Essa janela de tempo permite calcular a média de alguns parâmetros de QoC (e.g. acurácia) para os serviços descobertos.

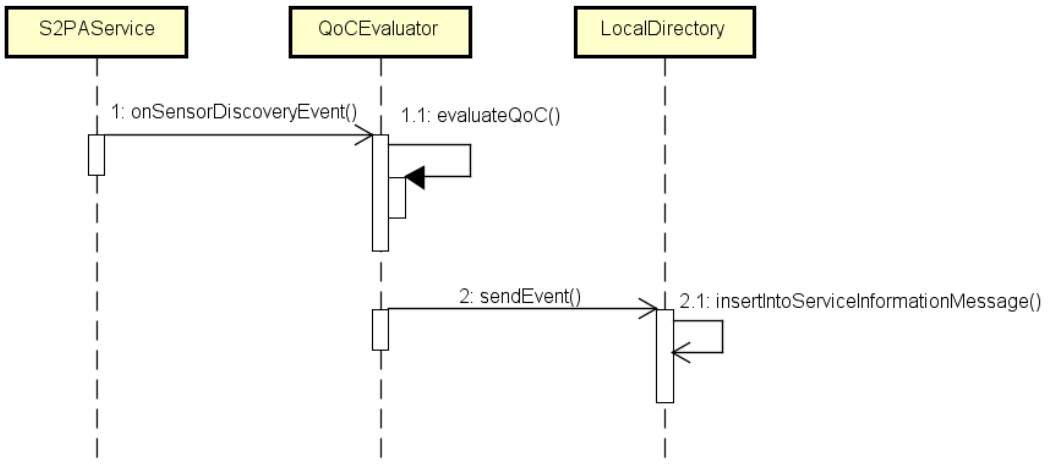


Figura 3.3: Diagrama de sequência da descoberta no S2PA.

Código 3.1: Regra CEP para inserção de um novo serviço no fluxo de eventos.

```
1 INSERT INTO ServiceInformationMessage(publisherID, serviceName,
    accuracy, measurementTime, availableAttributes,
    sourceLocationLatitude, sourceLocationLongitude,
    sourceLocationAltitude, measurementInterval,
    numericalResolution, age)
2   SELECT publisherID, serviceName, AVG(accuracy), AVG(
      measurementTime), AVG(availableAttributes), AVG(
      sourceLocationLatitude), AVG(sourceLocationLongitude), AVG(
      sourceLocationAltitude), AVG(measurementInterval), AVG(
      numericalResolution), AVG(age)
3   FROM SensorDataMessage.win:time(TIME_WINDOW)
4   GROUP BY publisherID, serviceName
```

O processo de descoberta contínua de serviços a nível local é apresentado no diagrama de sequência da Figura 3.4. Quando uma aplicação inicia ela deve registrar um *Listener* do tipo **ISubscriberListener** por meio do componente **Subscriber**. Este *Listener* possibilita o recebimento de mensagens de descoberta e também de dados de serviços subscritos. Ele é parte da API do M-Hub/CDDL, que será discutida com mais detalhes no Capítulo 4. Em seguida, o Subscriber cria um tópico de subscrição no **ConnectionService**, de onde mensagens de consulta podem ser respondidas para a aplicação. O ConnectionService é o componente que implementa o *broker* MQTT, responsável por estabelecer a comunicação no paradigma *Publish-Subscribe*.

Quando a aplicação submete uma consulta para descoberta contínua de serviços, uma regra CEP deve ser escrita. Um exemplo de regra simples é exibido no Código 3.2. Essa consulta é submetida ao componente **Publisher**

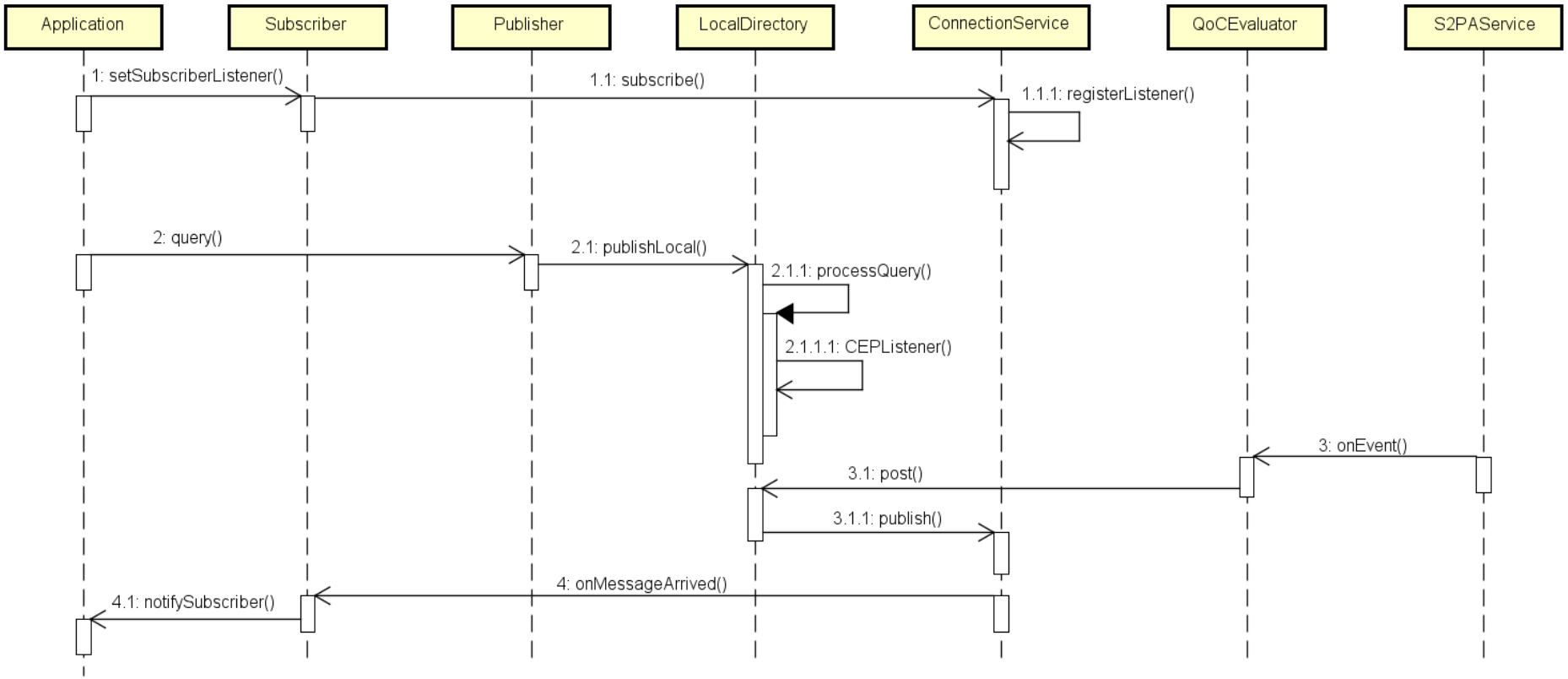


Figura 3.4: Diagrama de sequência da descoberta contínua de serviços a nível local no M-Hub/CDDL.

do M-Hub/CDDL, que submete-a ao **LocalDirectory**. O LocalDirectory é o componente responsável por registrar a regra no motor de regras CEP. Esse processo instancia um **UpdateListener** do CEP para esta regra, que é disparado sempre que hajam novos serviços que satisfaçam-na. Deste modo, quando o S2PA descobre um novo serviço e o insere no fluxo de eventos, se este serviço satisfizer à regra que a aplicação submeteu, a interface de acesso ao serviço será publicada no tópico de resposta a consultas, o qual a aplicação se subscreveu. O ConnectionService enviará uma notificação (**QueryResponseMessage**) através do ISubscriberListener da aplicação, contendo uma lista de objetos ServiceInformationMessage. Esses objetos contém a interface para a utilização do serviços.

Código 3.2: Exemplo de regra CEP para descoberta de um serviço de temperatura no M-Hub/CDDL.

```
1 SELECT * FROM ServiceInformationMessage
2 WHERE serviceName LIKE '%temperature%'
```

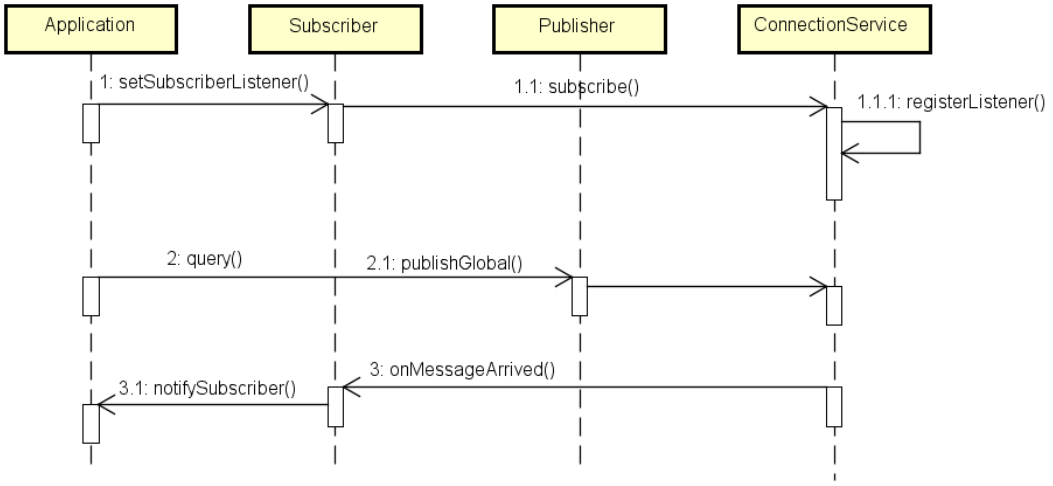



Figura 3.5: Diagrama de sequência da descoberta contínua de serviços a nível global no M-Hub/CDDL.

No modo de descoberta contínua de serviços a nível global utilizando *smartphones*, apresentado na Figura 3.5, o processo ocorre com a publicação direta da consulta no ConnectionService. Um tópico para recepção de respostas globais é registrado neste componente e um ISubscriberListener é instanciado na aplicação. Para que isso seja possível, outros *smartphones* com o M-Hub/CDDL instalado devem permitir a recepção de consultas globais. Esse processo é mostrado na Figura 3.6 e ocorre por meio do componente Subscriber se subscrevendo em um tópico para recepção de consultas globais. O ConnectionService registra um Listener para notificar o Subscriber. Toda vez que uma mensagem

de consulta remota é recebida, ela é enviada ao LocalDirectory desta instância de M-Hub/CDDL. O processo de registro da consulta ocorre da mesma forma descrita no processo de descoberta a nível local. Um UpdateListener do CEP é registrado para esta consulta e toda vez que hajam serviços que satisfaçam-na, uma QueryResponseMessage é enviada para publicação no ConnectionService. O M-Hub/CDDL remoto que publicou esta consulta, receberá esta resposta através de seu tópico de recepção de respostas globais e o ISubscriberListener da aplicação será notificado.

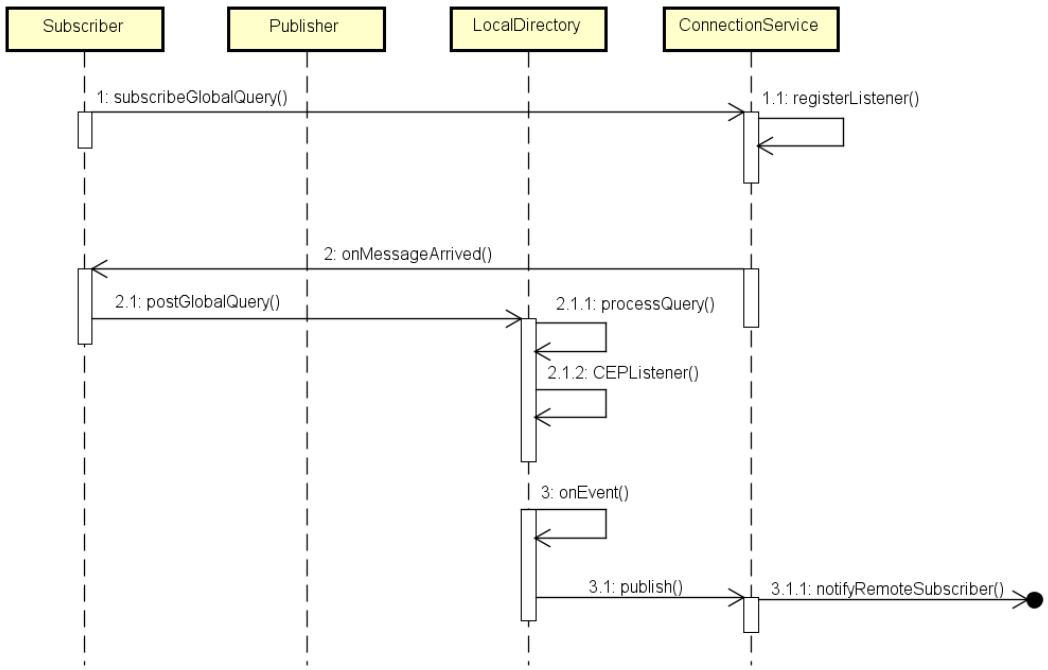


Figura 3.6: Diagrama de sequência do registro de permissão para recepção de regras globais no M-Hub/CDDL.

3.2 Discussão

Este capítulo descreveu o mecanismo de descoberta de serviços contínuo com critérios de QoC implementado neste trabalho. Ele constitui um aspecto importante para cenários de mobilidade irrestrita em IoMT, onde os *gateways* (o M-Hub/CDDL neste caso) e/ou os objetos inteligentes (e.g. sensores e atuadores) podem ser móveis. Estes cenários de grande dinamismo se beneficiam de um mecanismo de descoberta contínuo de serviços, que oportunisticamente encontra serviços de interesse em ambientes autorizados.

Com um mecanismo de descoberta de serviços simples (não contínuo), uma aplicação não poderia detectar a entrada de novos serviços de forma direta. Ela deveria implementar outros mecanismos para lidar com a descoberta

em mobilidade. Além disso, uma consulta contínua registrada no *middleware* permite uma maior expressividade na seleção de serviços que envolvem critérios de QoC desejados. Isso acontece porque a todo momento os novos serviços descobertos são inseridos no componente de processamento de eventos CEP e a aplicação recebe apenas os serviços que atendem à sua consulta publicada.

Ademais, o paradigma contínuo deixa a responsabilidade da descoberta a cargo do *middleware*, pois é baseado em notificações *bottom-up*. No paradigma de descoberta simples *top-down*, semelhante a uma consulta em um SGBD, a aplicação é responsável por dizer quando a consulta deve ser feita. Um sistema baseado em notificações permite o desacoplamento entre produtores e consumidores de eventos, além de permitir um melhor tempo de reação à ocorrência de eventos [33].

Do ponto de vista de um desenvolvedor de aplicações, uma API de um *middleware* que trate os problemas de descoberta de serviços, comunicação com sensores e atuadores, e a comunicação entre produtores e consumidores de eventos, facilita a criação de aplicações para IoT. Este trabalho propõe uma API que implementa os mecanismos descritos, que será mostrada e discutida com mais detalhes no Capítulo 4.

4
API para Descoberta Contínua de Serviços

Este Capítulo discutirá a API que permite a criação de aplicações utilizando o mecanismo de descoberta contínuo de serviços. Ela foi desenvolvida em linguagem Java na plataforma Android. O requisito mínimo necessário é utilizar a Android API 23.

O diagrama de atividades apresentado na Figura 4.1 apresenta a sequência básica de atividades necessárias que uma aplicação deve seguir para utilizar corretamente a API. O processo básico consiste em adquirir uma instância do serviço de conexão, registrar subscritor e publicador, publicar a consulta e aguardar a resposta dessas consultas. Cada resposta de consulta corresponde a um serviço que foi descoberto e pode ser utilizado pela aplicação.

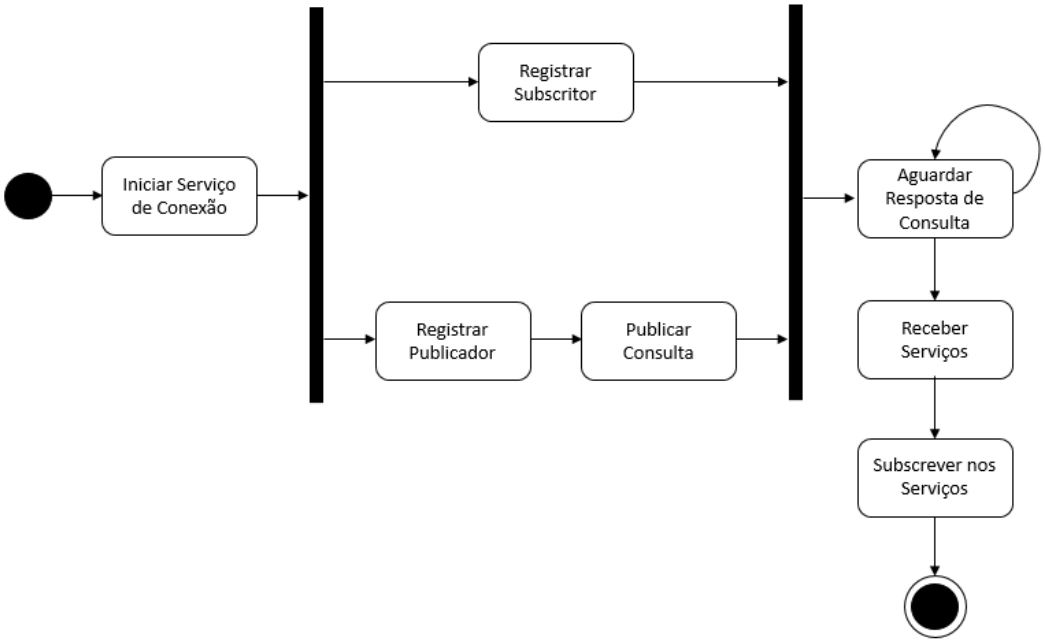


Figura 4.1: Diagrama de atividades da API de descoberta contínua.

A API demonstrada aqui será a parte necessária para o mecanismo de descoberta contínua de serviços. O M-Hub/CDDL possui diversas outras funcionalidades que não serão discutidas aqui, por não ser do escopo deste trabalho. As Seções a seguir descreverão o processo de configuração da API, uso do Listener de recepção de respostas à uma consulta, e a assinatura dos

serviços. Os exemplos que serão citados nas seções a seguir estão escritos em linguagem Java.

4.1 Configuração Inicial

Primeiramente a aplicação deve adquirir uma instância do serviço de conexão (**ConnectionService**), que implementa um *broker* MQTT, responsável por mediar a comunicação entre publicadores e consumidores. Este serviço permite a criação de um *micro-broker* para comunicação local e de um *server broker* para comunicação global. O Código 4.1 apresenta os principais métodos para instanciação de do ConnectionService.

Código 4.1: API para instanciação do serviço de conexão do M-Hub/CDDL

```
1 ConnectionService connectionService = ConnectionService.
      getInstance();
2 connectionService.setClientID(String "ID do cliente");
3 connectionService.setHost(String "host");
4 connectionService.setPort(int portNumber)
5 connectionService.startLocalBroker();
6 connectionService.connect();
```

Os métodos do ConnectionService permitem:

- Obter uma instância *singleton* do ConnectionService.
- Definir um identificador único para o cliente, que será utilizado posteriormente para identificar um cliente no *broker* de conexão e permitir a criação do tópico de recepção de mensagens de descoberta.
- Definir um *host* para uma conexão à um *broker* MQTT.
- Definir uma porta para uma conexão à um *broker* MQTT.
- Iniciar um *broker* local do MQTT, utilizando a implementação *Moquette*[1] para Android.
- Iniciar a conexão.

Com o mecanismo de comunicação provido pelos *brokers*, a aplicação deve obter instâncias do **Publisher** e do **Subscriber**, que serão utilizados para publicar consultas e receber respostas dessas consultas. A instância é obtida através da classe **Provider**. Com a instância de Subscriber, a aplicação deverá definir um *Listener* do tipo **ISubscriberListener** para receber as respostas de consultas de descoberta de serviços que publicará adiante. Esses métodos são apresentados no Código 4.2.

[1]Moquette Android - `https://github.com/technocreatives/moquette`

Código 4.2: API para obtenção do Publisher e do Subscriber.

```
1 Publisher publisher = Provider.newPublisher();
2 Subscriber subscriber = Provider.newSubscriber();
3 ISubscriberListener subscriberListener = new ISubscriberListener()
      ;
4 subscriber.setSubscriberListener(subscriberListener);
```

4.2 Implementação do ISubscriberListener

A aplicação deverá implementar o método obrigatório do ISubscriberListener para receber os resultados das consultas e também de dados de serviços assinados. Esse método é o **onMessageArrived (Message message)** que contém a resposta encapsulada numa instância da classe **Message**. Quando a Message é uma instância da classe **QueryResponseMessage**, ela contém as interfaces necessárias para a assinatura de serviços. Quando a Message é instância de **SensorDataMessage**, ela contém dados de sensores assinados. Um exemplo de fluxo no ISubscriberListener é apresentado no Código 4.3. Basicamente, uma QueryResponseMessage contém um método que devolve uma lista de serviços encontrados que satisfazem uma consulta publicada (**ServiceInformationMessage**). A aplicação pode então utilizar o Subscriber para subscrever-se aos serviços encontrados.

Código 4.3: Implementação do ISubscriberListener

```
1 ISubscriberListener subscriberListener = new ISubscriberListener()
      {
2    @Override
3    public void onMessageArrived(Message message) {
4       if(message instanceof QueryResponseMessage){
5          QueryResponseMessage queryRM = (QueryResponseMessage)
               message;
6          for (ServiceInformationMessage sim :
7          queryRM.getServiceInformationMessageList()) {
8             subscriber.
9             subscribeSensorDataTopicByPublisherIdAndServiceName(
10            sim.getPublisherID(),
11            sim.getServiceName());
12         }
13      }
14      if(message instanceof SensorDataMessage){
```

```
15          //Enviar dados para a aplicacao
16      }
17  }
```

4.3 Publicação de Consultas

Com o ISubscriberListener implementado, a aplicação pode por fim publicar sua consulta. Esse processo é demonstrado no Código 4.4. O método para consulta é o **query(QueryDestiny, QueryType, query)**. Dentro deste método o **QueryDestiny** indica o destino da consulta (que pode ser *GLOBAL, LOCAL* ou *LOCAL_AND_GLOBAL*). **QueryType** indica o tipo da consulta (*SIMPLE* ou *CONTINUOUS*), para ativar o modo de descoberta simples ou contínuo, respectivamente. O último parâmetro é uma *String* contendo a consulta, escrita como uma parte da cláusula "WHERE" de uma *query* EPL.

A consulta pode incluir a busca nos seguintes atributos:

- **serviceName**: Indica o nome do serviço publicado;
- **publisherID**: Indica o identificador do publicador do serviço;
- **accuracy**: É a acurácia da medição fornecida pelo serviço;
- **sourceLocationlatitude**: Coordenada de latitude do serviço;
- **sourceLocationlongitude**: Coordenada de longitude do serviço;
- **measurementTime**: Indica o tempo da medição (*timestamp*);
- **measurementInterval**: Indica o tempo entre duas medições consecutivas;
- **age**: Indica a idade da amostra (diferença entre o tempo atual e o tempo de medição);
- **numericalResolution**: Indica a granularidade dos dados em relação ao número de casas decimais fornecidos na medição.
- **availableAttributes**: Indica a quantidade de valores dentro do vetor de valores de resposta. Este atributo permite avaliar a completude da medição;

Código 4.4: Publicação e cancelamento de consultas.

```
1 //Consulta
2 String query = "serviceName LIKE '%temperature%'";
3 //Publica Consulta
```

```
4 long returnCode = publisher.query(QueryDestiny.LOCAL, QueryType.
      CONTINUOUS, query);
5 //Cancela Consulta
6 publisher.cancelQuery(returnCode);
7 }
```

A seguir são exibidos alguns exemplos de consultas:

- String query = "serviceName LIKE '%temperature%'";
- String query = "publisherID = 'felipe@gmail.com'";
- String query = "serviceName LIKE '%temperature%' AND publisherID = 'felipe@gmail.com'";
- String query = "serviceName LIKE '%temperature%' AND accuracy > 90 AND sourceLocationLatitude > -22.979000 AND sourceLocationLatitude < -22.978000 AND sourceLocationLongitude > -43.232000 AND sourceLocationLongitude < -43.231000";

Toda consulta publicada através do Publisher retorna um valor do tipo **long**. Este valor identifica o registro da consulta no M-Hub/CDDL. Este valor pode ser utilizado para a chamada ao método **cancelQuery(long returnCode)**, como visto no Código Código 4.4. Este método permite a exclusão da consulta no M-Hub/CDDL, para que a descoberta contínua de serviços seja cancelada.

4.4 Utilização dos Serviços Descobertos

Como pôde ser observado no Código 4.3, o ISubscriberListener devolve uma lista de ServiceInformationMessage para a aplicação. Ela pode utilizar os métodos definidos no Código 4.5 para se subscrever a serviços ou cancelar a subscrição. Esse mecanismo faz com que a aplicação internamente se subcreva no tópico MQTT ao qual as mensagens produzidas por aquele serviço específico estão sendo publicadas. Toda vez que um novo evento produzido pelo serviço é publicado, as aplicações que se subscreveram receberão os dados.

Código 4.5: Métodos da API do Subscriber para subscrição e cancelamento de subscrição a serviços.

```
1 //Subscricao por ID de publicador...
2 subscribeSensorDataTopicByPublisherId(String publisherID);
3
4 //Subscricao por instancia de ServiceInformationMessage...
```

```
5 subscribeSensorDataTopicByServiceInformationMessage(
      ServiceInformationMessage sim);
6
7 //Subscricao por ID de publicador e nome do servico...
8 subscribeSensorDataTopicByPublisherIdAndServiceName(String
      publisher_id, String serviceName);
9
10 //Cancelamento de subscricao por ID de publicador e nome do
      servico...
11 unsubscribeSensorDataTopicByPublisherIdAndServiceName(String
      publisher_id, String serviceName);
```

5
Avaliação

Diversas oportunidades para o mercado de IoT têm surgido com o avanço de pesquisas em Cidades Inteligentes. Um dos conceitos de Cidades Inteligentes é o de proporcionar melhores infraestruturas de tecnologia e comunicação nas cidades, visando a melhoria de vida das pessoas no contexto urbano do dia-a-dia [34]. Por essa definição, IoT proporciona uma diversa gama de sensores e atuadores que combinados podem prover melhoria nos serviços para os habitantes de uma cidade. Dentro de cidades inteligentes, um dos problemas é o de estacionar carros, o que leva à necessidade de soluções em estacionamentos inteligentes (*smart parking*).

A Seção 5.1 apresenta um estudo de caso no âmbito de estacionamentos inteligentes, visto que diversas empresas têm investido na elaboração de sensores para monitorar vagas em rua, estacionamentos públicos, e em estacionamentos privados. A Seção 5.2 apresenta uma aplicação hipotética, que foi construída utilizando o M-Hub/CDDL com sensores simulados, para permitir que um usuário descubra de forma contínua e oportunística, vagas em estacionamentos à medida em que se aproxima de seu destino. Nesse cenário, pressupõe-se que a aplicação do usuário está hospedada num *smartphone* com Android e ele inicia a procura por uma vaga quando está próximo ao seu destino. Ao pressionar o botão para a procura, a aplicação envia a consulta por serviços de vagas para o M-Hub/CDDL, que a registra no diretório de serviços local. A consulta registrada fica então enviando notificações para a aplicação quando novos serviços são descobertos.

5.1
Estacionamentos Inteligentes

A procura por vagas em estacionamentos é um dos grandes problemas de grandes cidades (com milhões de habitantes) [35]. Motoristas a procura de vagas acabam contribuindo para o aumento no congestionamento, devido à baixa velocidade em que trafegam, o que gera um efeito em cadeia, resultando na redução da velocidade média de tráfego. Isso traz consequências ambientais (poluição) e econômicas (aumento no consumo de combustível) para os motoristas e para a cidade como um todo.

Reconhecendo os problemas e impactos na vida cotidiana de uma grande cidade, causados por problemas de trânsito, diversas empresas têm investido em soluções para tornar mais inteligente o problema de estacionar carros. Mostrar a disponibilidade de vagas e guiar motoristas até elas através de aplicativos é um dos aspectos que pode ajudar a reduzir os congestionamentos.

Figura 5.1: Instalação de um sensor Siemens para monitorar vagas de rua em Berlim - Alemanha. Fonte: `www.siemens.com`

A Siemens, por exemplo, tem investido em uma infraestrutura para monitorar vagas disponíveis numa cidade através da instalação de um conjunto de sensores em postes de iluminação [36]. Os sensores atuam como radares e possuem algoritmos para identificar os espaços livres. Além disso, os sensores disponíveis ajudam a monitorar condições de tráfego, detectar erros de estacionamento que podem causar riscos e o fluxo de pedestres pelas vias. Não são revelados os aspectos tecnológicos destes sensores, por se tratar de um produto de mercado.

Um exemplo prático de implementação foi feito pelo Departamento de Transportes da cidade de Los Angeles (LADOT), num sistema denominado *LA Express Park™* [37]. Este projeto, inaugurado em 21 de Maio de 2012, consiste numa rede se sensores espalhados em uma área de $11.65Km^2$ no centro de Los Angeles. O principal objetivo é o de ajudar no controle de congestionamentos. Ele funde a tecnologia de sensores instalados no asfalto da rua para detectar a presença de carros, com um sistema de preços sob demanda para pagamento do estacionamento (parquímetros). O programa, que tem investimentos federais e municipais, ainda possui um aplicativo para guiar motoristas até as vagas em tempo-real, além de permitir o pagamento diretamente no aplicativo. Além dos sensores, o sistema de pagamento com preços dinâmicos ajuda a controlar o

Figura 5.2: Um sensor de vaga de estacionamento instalado no asfalto no centro de Los Angeles. Fonte: `www.laexpresspark.org`

preço utilizando taxas baseadas em oferta e demanda de vagas, contribuindo no aspecto geral do controle de congestionamento.

Figura 5.3: Sensor para monitorar vagas de estacionameto, produzido pela Nordic Semicondutor e utilizado no aplicativo ParkFi em Denver. Fonte: `www.nordicsemi.com`

Outro projeto prático, denominado ParkFi [38], constitui uma aplicação para *smartphones* na cidade de Denver - EUA, que utiliza sensores com conectividade *Bluetooth Low Energy* para mapear e monitorar vagas na cidade. O aplicativo interage com o sistema e guia os motoristas até uma vaga livre. Tudo que o motorista precisa fazer é apertar um botão no aplicativo, que o sistema o guiará até a vaga. O sensor detecta a presença do carro e envia uma notificação para a central de comando, a fim de manter a disponibilidade na oferta de vagas.

Diversas outras empresas, como a Cisco [39], a Comarch [40] e a Libelium [41], também oferecem soluções com uso de sensores para o domínio de Estacionamento Inteligente. Assim como as soluções apresentadas, elas também se baseiam em algum mecanismo para sensoriar as vagas e propagar para aplicações consumidoras. Algumas, por serem produtos vendidos no mercado, também possuem soluções para pagamentos sob demanda, além de outras facilidades.

Nos sensores para vagas de estacionamento que não possuem uma

infraestrutura cabeada, uma tecnologia WPAN é utilizada para se comunicar com *gateways*. Um mecanismo de descoberta como o implementado pelo M-Hub/CDDL poderia ser utilizado numa aplicação envolvendo estes sensores. A Seção 5.2 descreve uma aplicação hipotética num cenário onde existem sensores, que são capazes de detectar a presença de um automóvel, espalhados em vagas de estacionamentos pela cidade.

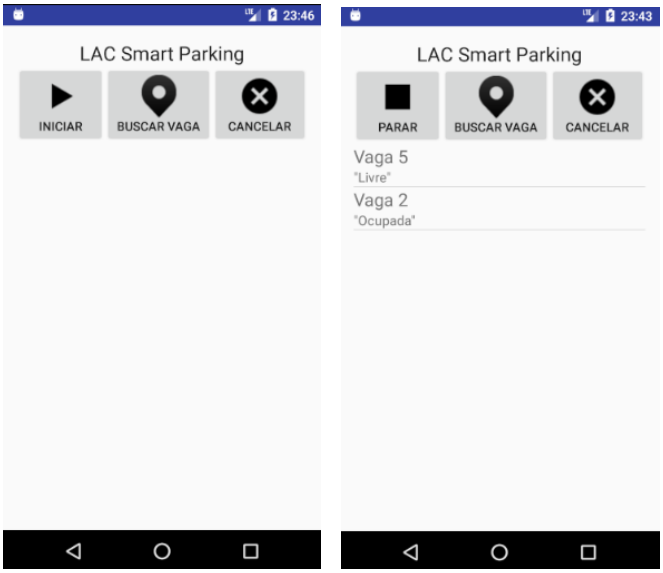
5.2 Implementação de uma Aplicação para Descoberta de Vagas em Estacionamentos com o M-Hub/CDDL

Nesta Seção descreveremos uma aplicação hipotética denominada *LAC Smart Parking* que utiliza o mecanismo de descoberta contínuo do M-Hub/CDDL. Ela permite a descoberta de sensores de vagas em estacionamentos espalhados por uma cidade enquanto um motorista transita próximo a elas.

Para a funcionalidade de descoberta de vagas livres desta aplicação, ela publica uma consulta por serviços de sensores de vagas. Ela também publica um filtro de subscrição informando que deseja receber apenas as vagas que estão com estado "Livre". Quando as vagas estão com estado "Livre" são descobertas, o subscritor as repassa para a aplicação. O filtro de subscrição descarta as que estão com estado de "Ocupada". A aplicação passa a monitorar as vagas com estado "Livre" e à medida em que algumas delas vão ficando ocupadas enquanto o motorista trafega, a aplicação executa o método para cancelar a subscrição. Por fim, quando o motorista estaciona e encerra a aplicação, a consulta é cancelada. Na Figura 5.4 algumas telas desta aplicação hipotética são exibidas, onde se pode observar vagas disponíveis e vagas que ficaram ocupadas.

Para esta aplicação, supomos que os sensores se comunicam localmente através de alguma tecnologia WPAN com uma instância do M-Hub/CDDL instalada num *smartphone*, e que a aplicação fará a consulta de descoberta contínua de serviços em nível local. Nesse cenário, os sensores sabem identificar quando uma vaga está ocupada ou livre e transmitem essa informação através de seus eventos publicados ao M-Hub/CDDL. Eles anunciam seus serviços utilizando os seguintes atributos:

- **Nome do serviço**: *Parking Spot Service*.
- **Valor de serviço**: 1 para vaga Livre ou 2 para vaga Ocupada.
- ***Timestamp***: Horário em que o sensor enviou os dados.
- **Latitude**: Coordenada de latitude da vaga.
- **Longitude**: Coordenada de longitude da vaga.



(a) Tela de Início (b) Vagas descobertas

Figura 5.4: Telas do LAC Smart Parking.

Para a modelagem desta aplicação deve ser utilizada uma consulta contínua com o parâmetro "*WHERE serviceName LIKE '%Parking Spot Service%'*". Quando um novo sensor que atende a esta consulta é encontrado, a aplicação executa o método de subscrição. Um filtro de subscrição do M-Hub/CDDL definido por "*WHERE serviceValue = 1*" observa se o estado da vaga é "Livre" e então encaminha a notificação para a aplicação. A aplicação agora passa a monitorar os eventos de notificação enviados pelos sensores de vagas. Quando o estado obtido é de vaga "Ocupada" o filtro não repassa a mensagem para a aplicação. Se em algum momento a vaga que antes era "Livre" passa a estar "Ocupada", a aplicação executa o método para cancelar a subscrição e para de monitorar a vaga.

Quando a aplicação recebe os eventos de vagas livres que estão subscritas, ela recebe as coordenadas de latitude e longitude, permitindo guiar o motorista até a vaga. À medida em que o motorista vai se movendo pela via, novas vagas na proximidade são descobertas, proporcionando mais opção de escolha. Por fim, quando o motorista estaciona, ele cancela a consulta e encerra a aplicação.

Possíveis extensões dessa aplicação poderiam incluir sensores com um pouco mais de funcionalidades e informações sobre as vagas. Por exemplo, os sensores poderiam informar o tamanho da vagas, a fim de permitir a ocupação de diferentes tamanhos de automóveis. Poderiam também incluir *flags* informando se são vagas preferenciais para idosos ou cadeirantes. Todas essas informações seriam passíveis de seleção através de um filtro do M-Hub/CDDL.

Com relação a critérios de QoC para a descoberta de vagas, a consulta

poderia utilizar o critério de verificar o tempo da medição (*timestamp*), para desprezar possíveis sensores de vagas de estacionamento que estejam com problema, ou seja, não atualizando corretamente os dados.

Outros controles lógicos e funcionalidades precisam ser criados para esta aplicação ter uso em ambientes reais. Um exemplo de funcionalidade seria o sistema para guiar os motoristas até as vagas. Outro exemplo seria a integração com sistemas de parquímetro para o pagamento automático do tempo de uso das vagas. Este exemplo descrito serviu apenas para elucidar o mecanismo de descoberta contínuo, utilizado através da API do M-Hub/CDDL.

Esse exemplo de aplicação, demonstra um fluxo básico do mecanismo de descoberta contínuo de serviços dentro do M-Hub/CDDL. Não é necessário um grande esforço de programação para a criação de aplicações simples como esta, visto que a API do M-Hub/CDDL fornece métodos para comunicação, escrita de uma consulta de descoberta contínua, e subscrição a eventos de serviços de sensores.

Construir esta mesma aplicação utilizando o mecanismo de descoberta convencional do M-Hub/CDDL exigiria que a consulta fosse publicada várias vezes durante o ciclo de vida de execução da aplicação. Se um novo sensor de vaga de estacionamento fosse descoberto pelo *Local Directory Service*, a aplicação não seria informada de imediato, pois teria que efetuar uma nova consulta para descobrir que existe um novo serviço no contexto do M-Hub/CDDL. Este cenário elucida a importância de um mecanismo de descoberta contínua em ambientes com mobilidade irrestrita, onde um ou mais elementos podem ser móveis. Estes ambientes possuem um grau de dinamismo muito maior e uma consulta estática prejudicaria a característica oportunística de aplicações neste domínio.

6
Trabalhos Relacionados

O aumento na disponibilidade de sensores e atuadores motivou diversos trabalhos acadêmicos a propor mecanismos de descoberta de serviços em IoT, com enfoque em permitir o uso em larga escala destes objetos. O paradigma de serviços é bastante aceito e difundido em IoT, visto que sua principal proposta é lidar com heterogeneidade de sensores e atuadores, além de permitir a criação de aplicações mais robustas, sem necessidade de preocupação com detalhes da comunicação entre objetos e *gateways*. Muitos deles defendem que estes mecanismos são e se tornarão ainda mais úteis em cenários de Cidades Inteligentes do futuro.

O mecanismo proposto neste trabalho utiliza o termo "*Descoberta Contínua de Serviços*" para referenciar um mecanismo assíncrono de descoberta baseado em sistemas de *Publish-Subscribe* que permite o uso de *queries* (consultas) para seleção de serviços com critérios de QoC desejados. Este termo representa uma ideia inerente ao mundo de processamento de fluxos de eventos contínuos, como no CEP, e é estendido para o mundo de *Publish-Subscribe* através do uso de *brokers* que lidam com aspectos de comunicação entre produtores e consumidores de eventos. Não são encontrados na literatura trabalhos que utilizem o termo "*Descoberta Contínua de Serviços*" para se referenciar a este tipo de paradigma, entretanto, existem trabalhos similares que implementam algum tipo de conceito semelhante.

Guinard et al. (2010) [42] apresentam um trabalho denominado *Real-World Service Discovery and Provisioning Process* (RSDPP, Provisionamento e Descoberta de Serviços no Mundo Real) para que uma aplicação dinamicamente descubra, consulte, selecione e utilize serviços de dispositivos do "mundo real". Os autores defendem uma abordagem baseada em *Web Services* para integrar os serviços dos dispositivos dentro de grandes sistemas de informação empresariais. Este mecanismo permite a consulta de descoberta de dispositivos utilizando palavras-chave, critérios de qualidade de serviço e outras propriedades de dispositivos. As consultas podem ser enriquecidas utilizando um mecanismo de chamado de *query augmentation* (aumento de consulta) que busca por palavras-chave similares em bases de ontologias na Internet. O mecanismo de descoberta é considerado ativo, pois a aplicação registra consultas e filtros sob demanda, ou

seja, quando necessita de novos dados. Quando novos dispositivos se conectam eles são registrados para futuras consultas. Não é mencionado sobre o uso de tecnologias WPAN para a descoberta de dispositivos. Os autores assumem que os dispositivos podem implementar padrões *Web Service* diretamente e não precisam de *gateways* para se comunicar.

van Sinderen et al. (2006) [43] definem uma infraestrutura denominada AWARENESS, que suporta o desenvolvimento de aplicações para ambientes móveis cientes de contexto e com seleção de critérios de QoC. É fornecida uma API para apoiar o desenvolvimento de aplicações. O trabalho permite o uso de qualquer tecnologia de rede para a comunicação com sensores, embora não mencione nenhuma que foi utilizada em testes. A consulta por serviços é suportada através de linguagem SQL ou *RDF Data Query Language*. É permitido um esquema de descoberta, o qual é denominado de passiva, onde a consulta fica publicada no *middleware* e notificações são enviadas para a aplicação quando existem melhores provedores de serviço (com melhores critérios de qualidade). Não é mencionado o esquema utilizado neste mecanismo de descoberta passivo. O comportamento das aplicações é definido nesse trabalho como um conjunto de regras de *Event Condition Action* (ECA) que definem ações na ocorrência de condições em cima de eventos que de interesse. Por exemplo, a mudança de qualidade de um serviço de sensor pode disparar uma regra para a troca por outro serviço sensor.

Gomes et al. (2016) [1] definem um *middleware* denominado QoDisco, que suporta a descoberta de serviços em IoT, utilizando critérios de QoC definidos por uma aplicação. O *middleware* tem suporte a consultas utilizando a linguagem *SPARQL Protocol and RDF Query Language*, onde os dados de sensores são modelados através de ontologias. Ele suporta os modos de descoberta síncrona e assíncrona. O modo síncrono é baseado no paradigma *Request-Response*, onde a consulta responde uma única vez. O modo assíncrono é implementado através de um esquema *Publish-Subscribe*, que permite o registro de uma consulta para quando novos serviços que atendam-na se conectem ao *middleware*, a aplicação seja notificada. O trabalho não informa os mecanismos utilizados para comunicação com sensores, abstraindo esse conceito para um repositório simulado de serviços.

BRIDGE [44] é um projeto suportado pela União Europeia (UE) e coordenado pela GS1[1] (empresa que detém a padronização de códigos de barras no mundo), que lida com a implementação de RFID (*Radio Frequency Identification*) na Europa. A visão do projeto é proporcionar descoberta de serviços em EPCIS (*Electronic Product Code Information Services*), que são

[1]GS1 - `http://www.gs1.org/`

descrições de um conjunto de objetos com etiquetas eletrônicas identificadas por EPCs (*Eletronic Produc Code*). Um código de EPC identifica unicamente um objeto físico através de RFID. Um dos padrões sugeridos pelo projeto para a descoberta de serviços utiliza o modelo *Publish-Subscribe*, onde clientes se subscrevem no sistema por certos códigos EPC. Quando um EPCIS transmite informações sobre um conjunto de EPCs que ele possui registro, os clientes subscritos recebem notificações com os EPC de interesse. Para obter os dados detalhados do EPC, o cliente consulta diretamente o EPCIS que mantém o registro. Esse processo constitui um mecanismo de descoberta contínuo, embora não utilize uma linguagem para consulta, pois não existem outros critérios de busca neste caso.

Diversos trabalhos propõem uma estratégia de criar uma *engine* de busca para serviços de dispositivos de IoT [45, 46, 47, 48]. Estes trabalhos se baseiam na ideia de criar repositórios de serviços de onde buscas podem ser efetuadas. O enfoque é, em geral, criar mecanismos eficientes para indexar os objetos e permitir uma busca eficiente. Entretanto, estas *engines* não lidam com o problema de descoberta contínua e oportunística, pois uma consulta feita acontece nos moldes de SGBDs, onde há apenas uma resposta ao final do processamento. Quando novos objetos são adicionados nos repositórios, uma nova busca deve ser feita para descobri-los.

6.1 Discussão

Os trabalhos apresentados neste Capítulo apresentam algum grau de semelhança com o trabalho desenvolvido nesta dissertação. Dentre todos, o AWARENESS [43] é o que possui um grau de semelhança maior, pois permite o uso de consultas para descoberta no modo contínuo (passivo, no caso do AWARENESS). A descoberta passiva descrita no AWARENESS se assemelha ao modo contínuo que foi implementado neste trabalho, já que utiliza um paradigma *Publish-Subscribe* para tal e funciona com notificações de entrada de novos dispositivos que atendem aos critérios de consulta. A Tabela 6.1 apresenta um resumo comparativo entre os trabalhos discutidos neste capítulo.

Embora existam trabalhos que utilizam CEP para processamento de dados de sensores em IoT [49, 50], nenhum deles utiliza-o para elaborar consultas referentes à descoberta de serviços. Neste trabalho, defendemos o uso de CEP no processo de descoberta, dada a expressividade da linguagem na busca por eventos com critérios desejados. CEP realiza um processamento dos eventos em memória, e não em disco. Isso faz com que a vazão de saída de eventos processados seja próxima ao tempo-real [49]. Devido a essa característica do

Tabela 6.1: Comparação entre os trabalhos relacionados.

Trabalho	Descoberta Contínua	Consultas	Comunicação	QoC
Guinard et. al. (2010)	X	Não especifica	Web Services	Somente Qos
van Sinderen et al. (2006)	✓	SPARQL / SQL	Pub-Sub	✓
Gomes et al. (2016)	✓	SPARQL	Pub-Sub	✓
BRIDGE (2007)	✓	Não especifica	Pub-Sub	Não especifica
M-Hub/CDDL (2017)	✓	CEP	Pub-Sub (MQTT)	✓

CEP, a utilizamos como tecnologia para que os objetos inteligentes fossem rapidamente detectados em cenários de mobilidade. Uma vez que um sensor é descoberto fisicamente e seus dados e metadados são inseridos no fluxo de eventos CEP, uma consulta registrada recebe um *callback* e em seguida a aplicação que a publicou é notificada localmente ou através de um *broker* MQTT.

O paradigma *Publish-Subscribe* é bastante utilizando como mecanismo de comunicação em IoT. Diversos trabalhos [42, 43, 1, 44, 51, 52] utilizam implementações desse mecanismo para mediar a comunicação entre produtores e consumidores de eventos. Embora muitas implementações de *Publish-Subscribe* não forneçam mecanismos para efetuar consultas expressivas através de uma linguagem estruturada, diversos trabalhos agregam esse conceito para selecionar serviços com características mais específicas. SPARQL é um exemplo de linguagem utilizada.

A busca por serviços em IoT pode envolver critérios de qualidade de contexto (QoC) desejáveis em domínios de certas aplicações. Utilizar um mecanismo que permite selecionar serviços segundo alguns parâmetros de QoC [1, 11] corrobora com a seleção de serviços mais alinhados às necessidades de aplicações. Por exemplo, uma aplicação que necessita de dados acurados de GPS pode efetuar uma busca por um GPS incluindo um valor de acurácia mínima desejada. Neste trabalho, o M-Hub/CDDL suporta que critérios de QoC sejam utilizados nas consultas para descoberta contínua de serviços.

7
Conclusão e Trabalhos Futuros

O enfoque dado a IoT na atualidade, principalmente no âmbito de cidades inteligentes, proporciona diversas oportunidades e desafios de pesquisa. Uma enorme quantidade de sensores e atuadores está presente em diversos ambientes de uma cidade ou na própria residência das pessoas. Através da composição destes sensores e atuadores, aplicações diversas podem ser criadas visando trazer melhorias na vida das pessoas e no funcionamento das cidades. Para que sensores e atuadores possam ser utilizados em larga escala, mecanismos para descoberta de serviços precisam ser criados. O paradigma de serviços permite esconder detalhes de implementação e comunicação que não interessam para aplicações de alto nível. Nesse contexto, um *middleware* para descoberta de serviços se torna necessário para prover este grau de abstração desejado.

Diante dos desafios expostos, este trabalho apresentou o *middleware* e *gateway* de objetos inteligentes M-Hub/CDDL, desenvolvido em plataforma Android. Foi desenvolvida uma extensão para permitir a descoberta contínua e oportunística de serviços em ambientes de mobilidade irrestrita. Esta descoberta proporciona um meio de registro contínuo para descobrir novos serviços de sensores e atuadores, utilizando uma consulta em linguagem EPL com possíveis critérios de qualidade de contexto expressos. É disponibilizada uma API que auxilia desenvolvedores de aplicações com o processo de descoberta e uso de sensores e atuadores. Essa API permite que aplicações sejam desenvolvidas sem se preocupar com detalhes subjacentes, como a descoberta física e conexão a sensores e atuadores. O M-Hub/CDDL trata dos aspectos necessários para que a API seja de alto nível.

A comunicação em IoT lida com uma grande quantidade de sensores e atuadores e aplicações cliente que os utilizam. Para que esta comunicação seja eficiente e escalável, o paradigma *Publish-Subscribe* é utilizado por proporcionar um meio de comunicação assíncrono e com desacoplamento entre produtores e consumidores de eventos.

Lidar com a mobilidade irrestrita de dispositivos é um desafio em diversas áreas de sistemas distribuídos, e uma delas é no processo de descoberta de serviços. Uma aplicação hipotética no âmbito de estacionamentos inteligentes foi demonstrada, a fim de exemplificar um cenário onde este mecanismo de

descoberta contínua pode ser aplicado. Construir essa aplicação não demanda grande esforço de codificação, pois a API suporta o processo de descoberta e uso dos sensores necessários.

O estudo de trabalhos relacionados, que possuem algum grau de semelhança com o trabalho desenvolvido nesta dissertação, mostrou algumas das dificuldades e possíveis soluções no âmbito da descoberta de serviços em IoT. Muito embora não tenhamos encontrado trabalhos que se referem a este mecanismo de descoberta como contínuo e oportunístico, alguns deles implementam uma descoberta aos moldes do *Publish-Subscribe* permitindo consultas com uma linguagem estrutura e critérios de QoC. Os três paradigmas que defendemos nesta dissertação (comunicação *Publish-Subscribe*, consultas contínuas elaboradas em CEP, busca em critérios de QoC), aparecem em trabalhos elencados, apesar de alguns não especificarem quais tecnologias utilizaram. A abordagem aplicada aqui, que combina comunicação MQTT e processamento CEP, é inovadora dentro desse contexto.

7.1 Trabalhos Futuros

Um dos aspectos desejáveis para um mecanismo de descoberta de serviços é a descrição semântica dos objetos descobertos. Uma ontologia pode definir termos comuns usados em IoT para modelar os serviços publicados. Isso facilita a definição dos termos a serem pesquisados, além de retirar um ponto de indução de falhas do *middleware*, ocasionado pela escrita incorreta de consultas. Este aspecto poderia estender e melhorar a utilização do M-Hub/CDDL

Outro aspecto a ser investigado é a escalabilidade desta extensão do M-Hub/CDDL num ambiente com diversas instâncias do *middleware* executando e diversos sensores. A escalabilidade é um parâmetro de suma importância em sistemas de larga escala para IoT, visto que estes ambientes são de grande dinamicidade e a quantidade de objetos disponíveis nestes ambientes tem perspectivas de aumentar num futuro próximo. Além disso, testes de desempenho podem ser feitos, visando avaliar a viabilidade de uso do *middleware*.

Outro aspecto interessante a ser investigado, e que corrobora com a viabilidade de uso, é o consumo de energia ocasionado pelo M-Hub/CDDL executando em *smartphones*, visto que a bateria destes dispositivos é limitada. Preocupações com problemas energéticos são recorrentes não somente em IoT, mas em diversos outros setores. Esse aspecto condiz com a visão de IoT do futuro, no âmbito de cidades inteligentes sustentáveis.

Referências bibliográficas

[1] P. Gomes, E. Cavalcante, T. Batista, C. Taconet, S. Chabridon, D. Conan, F. C. Delicato, and P. F. Pires, "A qoc-aware discovery service for the internet of things," in *Ubiquitous Computing and Ambient Intelligence: 10th International Conference, UCAmI 2016, San Bartolomé de Tirajana, Gran Canaria, Spain, November 29–December 2, 2016, Part II 10*. Springer, 2016, pp. 344–355.

[2] R. van Kranenburg and S. Dodson, *The Internet of Things: A Critique of Ambient Technology and the All-seeing Network of RFID*, ser. Network notebooks. Institute of Network Cultures, 2008. [Online]. Available: https://books.google.com.br/books?id=PilgkgEACAAJ

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[4] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, and L. Veltri, "A scalable and self-configuring architecture for service discovery in the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 508–521, 2014.

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

[6] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[7] F. DaCosta, *Rethinking the Internet of Things: a scalable approach to connecting everything*. Apress, 2013.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[9] T. Buchholz and M. Schiffers, "Quality of context: What it is and why we need it," in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, vol. 3. IEEE, 2012, pp. 1009–1012.

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

[11] A. Manzoor, H.-L. Truong, and S. Dustdar, "Quality of context: models and applications for context-aware systems in pervasive environments," *The Knowledge Engineering Review*, vol. 29, no. 02, pp. 154–170, 2014.

[12] F. Paganelli and D. Parlanti, "A dht-based discovery service for the internet of things," *Journal of Computer Networks and Communications*, vol. 2012, 2012.

[13] F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," *IEEE Pervasive computing*, vol. 4, no. 4, pp. 81–90, 2005.

[14] C. Bettstetter and C. Renner, "A comparison of service discovery protocols and implementation of the service location protocol," in *Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications*, 2000.

[15] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, "Toward distributed service discovery in pervasive computing environments," *IEEE Transactions on mobile computing*, vol. 5, no. 2, pp. 97–112, 2006.

[16] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *OASIS standard*, vol. 12, p. 18, 2006.

[17] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *IEEE Internet computing*, vol. 6, no. 2, pp. 86–93, 2002.

[18] T. A. Butt, I. Phillips, L. Guan, and G. Oikonomou, "Adaptive and context-aware service discovery for the internet of things," in *Internet of things, smart spaces, and next generation networking*. Springer, 2013, pp. 36–47.

[19] S. K. Datta, R. P. F. Da Costa, and C. Bonnet, "Resource discovery in internet of things: Current trends and future standardization aspects," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 542–547.

[20] I. Chatzigiannakis, H. Hasemann, M. Karnstedt, O. Kleine, A. Kroller, M. Leggieri, D. Pfisterer, K. Romer, and C. Truong, "True self-configuration for the iot," in *Internet of Things (IOT), 2012 3rd International Conference on the*. IEEE, 2012, pp. 9–15.

[21] M. Zhou and Y. Ma, "A web service discovery computational method for iot system," in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, vol. 3. IEEE, 2012, pp. 1009–1012.

[22] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[23] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Cloud of things for sensing as a service: sensing resource discovery and virtualization," in *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE, 2015, pp. 1–7.

[24] D. Powell, "Group communication," *Communications of the ACM*, vol. 39, no. 4, pp. 50–54, 1996.

[25] Y. Liu, B. Plale *et al.*, "Survey of publish subscribe event systems," *Computer Science Dept, Indian University*, vol. 16, 2003.

[26] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[27] D. Luckham, *The power of events*. Addison-Wesley Reading, 2002, vol. 204.

[28] M. Eckert and F. Bry, "Complex event processing (cep)," *Informatik-Spektrum*, vol. 32, no. 2, pp. 163–167, 2009.

[29] EsperTech, "Esper - Complex Event Processing," 2017. [Online]. Available: http://www.espertech.com/esper/

[30] Microsoft, "Microsoft StreamInsight," 2017. [Online]. Available: https://msdn.microsoft.com/pt-br/library/ee362541(v=sql.111).aspx

[31] Apache, "Apache flink," 2017. [Online]. Available: https://flink.apache.org/

[32] M. Eggum, "Smartphone assisted, complex event processing," 2014. [Online]. Available: https://www.duo.uio.no/bitstream/handle/10852/41663/Marcel-Eggum|Thesis.pdf

[33] B. Segall and D. Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching," *Proceedings of the I997 Australian UNLX Users Group (A UUG'I997)*, pp. 243–255, 1997.

[34] T. Nam and T. A. Pardo, "Conceptualizing smart city with dimensions of technology, people, and institutions," in *Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times*. ACM, 2011, pp. 282–291.

[35] E. Polycarpou, L. Lambrinos, and E. Protopapadakis, "Smart parking solutions for urban areas," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*. IEEE, 2013, pp. 1–6.

[36] Siemens, "Siemens integrated smart parking solution," 2017. [Online]. Available: http://www.mobility.siemens.com/mobility/global/en/urban-mobility/road-solutions/integrated-smart-parking-solution/pages/integrated-smart-parking-solution.aspx

[37] L. A. D. of Transportation, "La express park™," 2017. [Online]. Available: http://www.laexpresspark.org/

[38] P. Inc, "Parkfi," 2017. [Online]. Available: https://denver.parkifi.com/

[39] Cisco, "Cisco smart+connected parking," 2017. [Online]. Available: http://www.cisco.com/c/en/us/solutions/industries/smart-connected-communities/city-parking.html

[40] Comarch, "Comarch smart parking," 2017. [Online]. Available: https://smartcity.comarch.com/smart-city/smart-parking/

[41] Libelium, "Libelium smart parking," 2017. [Online]. Available: http://www.libelium.com/smart_parking/

[42] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, 2010.

[43] M. J. van Sinderen, A. T. van Halteren, M. Wegdam, H. B. Meeuwissen, and E. H. Eertink, "Supporting context-aware mobile applications: an infrastructure approach," *IEEE Communications Magazine*, vol. 44, no. 9, pp. 96–104, 2006.

[44] BRIDGE, "Bridge wp02 – high level design for discovery services," 2007.

[45] W. T. Lunardi, E. de Matos, R. Tiburski, L. A. Amaral, S. Marczak, and F. Hessel, "Context-based search engine for industrial iot: Discovery, search, selection, and usage of devices," in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 2015, pp. 1–8.

[46] B. Ostermaier, K. Römer, F. Mattern, M. Fahrmair, and W. Kellerer, "A real-time search engine for the web of things," in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[47] Z. Ding, Z. Chen, and Q. Yang, "Iot-svksearch: a real-time multimodal search engine mechanism for the internet of things," *International Journal of Communication Systems*, vol. 27, no. 6, pp. 871–897, 2014.

[48] A. J. Jara, P. Lopez, D. Fernandez, J. F. Castillo, M. A. Zamora, and A. F. Skarmeta, "Mobile digcovery: A global service discovery for the internet of things," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*. IEEE, 2013, pp. 1325–1330.

[49] N. Stojanovic, L. Stojanovic, Y. Xu, and B. Stajic, "Mobile cep in real-time big data processing: challenges and opportunities," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*. ACM, 2014, pp. 256–265.

[50] C. Y. Chen, J. H. Fu, P.-F. Wang, E. Jou, and M.-W. Feng, "Complex event processing for the internet of things and its applications," in *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1144–1149.

[51] Y. Sun, X. Qiao, B. Cheng, and J. Chen, "A low-delay, lightweight publish/-subscribe architecture for delay-sensitive iot services," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 179–186.

[52] M. Collina, G. E. Corazza, and A. Vanelli-Coralli, "Introducing the qest broker: Scaling the iot by bridging mqtt and rest," in *Personal indoor and mobile radio communications (pimrc), 2012 ieee 23rd international symposium on*. IEEE, 2012, pp. 36–41.