**UNIVERSIDADE FEDERAL DE PELOTAS**
**Centro de Desenvolvimento Tecnológico**
**Programa de Pós-Graduação em Computação**


Dissertação

# EXEHDA-HS: Uma Abordagem Baseada em Redes Peer-to-Peer para Descoberta de Recursos na IoT

**Huberto Filho Kaiser**

Pelotas, 2019

**Huberto Filho Kaiser**

# EXEHDA-HS: Uma Abordagem Baseada em Redes Peer-to-Peer para Descoberta de Recursos na IoT

Dissertação apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Adenauer Corrêa Yamin
Coorientadora: Profa. Dra. Ana Marilza Pernas Fleischmann

Pelotas, 2019

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

K13e Kaiser Filho, Huberto

EXEHDA-HS : uma abordagem baseada em redes peer-t--peer para descoberta de recursos na IoT / Huberto Kaiser Filho ; Adenauer Corrêa Yamin, orientador ; Ana Marilza Pernas Fleischmann, coorientadora. — Pelotas, 2019.

92 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2019.

1. Descoberta de recursos. 2. Peer-to-peer. 3. Internet das coisas. 4. Middleware exehda. I. Yamin, Adenauer Corrêa, orient. II. Fleischmann, Ana Marilza Pernas, coorient. III. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

**Huberto Filho Kaiser**

**EXEHDA-HS: Uma Abordagem Baseada em Redes Peer-to-Peer para Descoberta de Recursos na IoT**

Dissertação aprovada, como requisito parcial, para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

**Data da Defesa:** 25 de Julho de 2019

**Banca Examinadora:**

Prof. Dr. Adenauer Correa Yamin (orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Profa. Dra. Ana Marilza Pernas Fleischmann (coorientadora)
Doutora em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. João Ladislau Barbará Lopes
Doutor em Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Rafael Iankowski Soares
Doutor em Computação pela Pontifícia Universidade Católica do Rio Grande do Sul.

*Not everything that can be counted counts, and not everything that counts can be counted.*
— WILLIAM BRUCE CAMERON

# RESUMO

KAISER, Huberto Filho. **EXEHDA-HS: Uma Abordagem Baseada em Redes Peer-to-Peer para Descoberta de Recursos na IoT**. Orientador: Adenauer Corrêa Yamin. 2019. 92 f. Dissertação (Mestrado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2019.

A Internet das coisas (IoT) é caracterizada por uma grande quantidade de recursos heterogêneos conectados à Internet. Contudo, esta alta heterogeneidade e a crescente escalabilidade se mostram como desafios a ser enfrentados, principalmente no que diz respeito a gerência destes recursos.

Registra-se uma tendência de remover estas funcionalidades das aplicações, repassando as mesmas para *middlewares*, ficando estes responsáveis por prover um meio padronizado para o acesso aos dados e serviços fornecidos pelos objetos através de uma interface de alto nível, além de facilitar a construção de aplicações para a IoT.

O trabalho desenvolvido nesta dissertação, denominado EXEHDA-HS, está relacionado à concepção de mecanismos a serem integrados ao *middleware* EXEHDA destinados a descoberta e busca por recursos distribuídos no ambiente celular do *middleware*. Contribuindo com o suporte para o tratamento da elevada escalabilidade dos ambientes computacionais típicos da IoT no que diz respeito aos procedimentos de busca, inclusão e remoção de dispositivos, bem como minimizando os esforços de configuração por parte do usuário quando da necessidade de realização destes procedimentos.

Na avaliação do EXEHDA-HS foram empregados dois cenários de uso no domínio da agricultura, tendo como base o projeto plenUS em desenvolvimento na Embrapa Clima Temperado. Os resultados atingidos com os cenários de uso se mostraram promissores, estimulando a continuidade das pesquisas na área.

Palavras-chave: Descoberta de Recursos. Peer-to-Peer. Internet das Coisas. Middleware EXEHDA.

# ABSTRACT

KAISER, Huberto Filho. **EXEHDA-HS: A Peer-to-Peer Networking Approach for IoT Resource Discovery**. Advisor: Adenauer Corrêa Yamin. 2019. 92 f. Dissertation (Masters in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2019.

The Internet of Things (IoT) is characterized by a large number of heterogeneous resources connected to the Internet. However, this high heterogeneity and increasing scalability are presented as challenges to be faced, especially in the management of these resources.

There is a tendency to remove these functionalities from the applications, passing them to *middlewares*, and these are responsible for providing a standardized way to access the data and services provided by the objects through a high-level interface, besides facilitating the construction of applications for IoT.

The work developed in this dissertation, called EXEHDA-HS, is related to the design of mechanisms to be integrated to the *middleware* EXEHDA for the discovery and search for resources distributed in the cellular environment of *middleware*. Contributing with the support for the treatment of the high scalability of the typical computational environments of the IoT concerning the procedures of search, inclusion and removal of devices, as well as minimizing the efforts of configuration by the user when the need to perform these procedures.

In the evaluation of the EXEHDA-HS were used two scenarios of use in the field of agriculture, based on the plenus project under development in the Embrapa Temperate Climate. The results achieved with the use scenarios were promising, stimulating the continuity of research in the area.

Keywords: Resource Discovery. Peer-to-Peer. Internet of Things. Middleware EXEHDA.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | Application Program Interface |
| AVU | Ambiente Virtual do Usuário |
| BDA | Base de Dados das Aplicações |
| BLOB | Binary Large Object |
| CAN | Content Addressable Network |
| CEP | Complex Event Processing |
| CIB | Cell Information Base |
| CORE | Common Open Research Emulator |
| CRUD | Create, Read, Update and Delete |
| CoAP | Constrained Application Protocol |
| CoT | Cloud of Things |
| DGT | Distributed Geographic Table |
| DHCP | Dynamic Host Configuration Protocol |
| DHT | Distributed Hash Table |
| DLS | Distributed Location Service |
| DNS | Domain Name System |
| DS | Discovery System |
| DoS | Denial of Service |
| EPC | Electronic Product Code |
| EXEHDA-HS | EXEHDA: Hierarchical Search |
| EXEHDA | Execution Environment for Highly Distributed Applications |
| GPS | Global Positioning System |
| GSM | Global System for Mobile |
| HTTP | Hypertext Transfer Protocol |
| IPFS | InterPlanetary File System |
| IP | Internet Protocol |

| | |
|---|---|
| IoT | Internet of Things |
| LBS | Location Based Service |
| LUPS | Laboratory of Ubiquitous and Parallel Systems |
| mDNS | Multicast Domain Name System |
| MIT | Massachusetts Institute of Technology |
| MQTT | Message Queuing Telemetry Transport |
| NAT | Network Address Translation |
| ORM | (Object-Relational Mapping |
| OX | Objeto EXEHDA |
| P2P | Peer-to-Peer |
| PHT | Prefix Hash Tree |
| plenUS | Plentiful Ubiquitous Systems |
| QoC | Quality of Context |
| RDF | Resource Description Framework |
| RDO | Repository Domain Ontology |
| REST | Representational State Transfer |
| RID | Repositório de Informações Descobertas |
| RID | Repositório de Informações Descobertas |
| RMI | Remote Method Invocation |
| RPC | Remote Procedure Call |
| SD | Sistema de Discoberta |
| SFC | Space Filling Curves |
| SQL | Structured Query Language |
| SSDP | Simple Service Discovery Protocol |
| UDP | User Datagram Protocol |
| UPNP | Universal Plug and Play |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| XML | Extensible Markup Language |

# SUMÁRIO

# 1 INTRODUÇÃO

A Computação Ubíqua considera um ambiente computacional populado por dispositivos que interoperam de modo automatizado na oferta de serviços (OCEGUEDA-MIRAMONTES; SANCHEZ; AGUILAR, 2019). Estes devem prover acesso proveniente de qualquer lugar, a todo o tempo e por qualquer dispositivo. Nesta perspectiva, a computação e seus diversos sistemas interagem com o ser humano a todo o momento, não importando onde ele esteja, em casa, no trabalho ou na rua, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico, mutável e com forte interação entre homem e máquina (SEZER; DOGDU; OZBAYOGLU, 2018). Dentre as várias abordagens para implementação das premissas da Computação Ubíqua, a Internet das Coisas é a que vem ganhando destaque no cenário internacional.

A Internet das Coisas (do inglês *Internet of Things*: IoT) (CHOUDHARY; JAIN, 2017) é uma abordagem que visa a integração do mundo físico com o digital e sua intercomunicação. Desta forma, as coisas são recursos capazes de prover informações com valor agregado e funcionalidades úteis para o usuário final e/ou suas aplicações (BASSI et al., 2016).

O cenário atual contabiliza mais de seis bilhões de coisas conectadas à IoT, com uma previsão de crescimento para 100 bilhões até 2025 (BIS Research, 2017), cada coisa podendo disponibilizar mais de um serviço às aplicações dos usuários.

Considerando estes indicadores, neste cenário de elevada escalabilidade, um serviço autônomo de descoberta é fundamental para tornar estes dispositivos facilmente acessíveis. Deste modo, gerenciar uma grande quantidade de recursos heterogêneos e com disponibilidade dinâmica vem constituindo um dos grandes desafios da IoT (PAGANELLI; PARLANTI, 2012).

Visando o tratamento deste desafio, destaca-se o emprego de *middlewares*. Estes são softwares inseridos entre as aplicações e as infraestruturas computacionais, os quais disponibilizam, dentre outras características, um meio padronizado para o acesso aos recursos. Os *middlewares*, por meio de uma interface de alto nível, permitem a integração e interoperabilidade dos diferentes componentes IoT (LOPES, 2016). O LUPS (*Laboratory of Ubiquitous and Parallel Systems*), desenvolve um *middleware*

para atendimento das demandas de IoT denominado EXEHDA (*Execution Environment for Highly Distributed Applications*), o qual fará parte do escopo de pesquisa desta dissertação.

## 1.1 Objetivos

Considerando as motivações direcionadas à IoT, o objetivo central deste trabalho denominado EXEHDA-Hierarchical Search (EXEHDA-HS) é a concepção de uma arquitetura destinada ao atendimento das demandas inerentes à IoT, principalmente no que diz respeito a descoberta de recursos em um cenário de alta escalabilidade e heterogeneidade. O seu desenvolvimento tem como premissa a integração à arquitetura de software do *middleware* EXEHDA.

Tal contribuição para a área da descoberta de recursos pressupõem a consecução de alguns objetivos específicos, os quais além de nortear os esforços de natureza científica para o desenvolvimento do trabalho, irão contextualizar as opções tecnológicas disponíveis para concepção do EXEHDA-HS. Abaixo a relação de objetivos específicos prevista:

- Concepção de uma arquitetura com a respectiva definição de seus módulos e funcionalidades, destinadas à busca e descoberta de recursos;
- Definição de uma abordagem de interoperação descentralizada, considerando a instanciação da arquitetura proposta na organização multicelular do EXEHDA;
- Concepção de procedimentos, suas diferentes mensagens e metadados, para suporte às operações da arquitetura proposta.

De modo mais específico, o EXEHDA-HS está concebido para ser instanciado no Subsistema de Execução Distribuída do *middleware*, oferecendo uma camada de abstração para permitir que um determinado recurso ao estar disponível possa se integrar a outros descobertos dinamicamente. Ou seja, o EXEHDA-HS contribui com o *middleware* EXEHDA por meio de um subsistema para gerenciar a descoberta de recursos no ambiente celular. Este atua por meio de uma sobreposição P2P para descoberta intercelular e utilizando o protocolo mDNS para descoberta de recursos locais da célula.

## 1.2 Estrutura do Texto

O Capítulo 1 apresenta uma introdução ao escopo desta dissertação, abordando as motivações e os objetivos do EXEHDA-HS. O Capítulo 2 apresenta os conceitos utilizados para a fundamentação conceitual, descrevendo as principais definições associadas aos esforços de pesquisa. Na sequência, o Capítulo 3 apresenta trabalhos

relacionados e ao final os compara em relação as características de arquitetura e descoberta de recursos.

Uma visão geral da concepção do EXEHDA-HS é apresentada no Capítulo 4, apresentando as características da arquitetura proposta, suas premissas de concepção e tecnologias empregada. No Capítulo 5, dois cenários são apresentados com o objetivo de avaliar as funcionalidades do EXEHDA-HS e seus comportamentos no que diz respeito a descoberta de recursos no ambiente celular.

Por fim, no Capítulo 6 são apresentadas as considerações finais acerca do trabalho, revisando as principais contribuições da dissertação e apresentando os trabalhos futuros.

# 2 FUNDAMENTAÇÃO CONCEITUAL

Este capítulo sistematiza conceitos e definições associados aos esforços de pesquisa desta dissertação. A Seção 2.1 apresenta uma discussão sobre Internet das Coisas, abordando conceitos, definições e desafios de pesquisa no domínio da descoberta de recursos.

A Seção 2.2 é destinada a apresentar os conceitos relacionados ao *middleware* EXEHDA e seus subsistemas, apresentando sua arquitetura de software e perfis de operação. Por fim, a seção 2.3 apresenta uma discussão de propostas para descoberta de recursos em sistemas distribuídas, descrevendo diferentes arquiteturas, requisitos e abordagens de funcionamento.

## 2.1 Descoberta de Recursos na IoT: Conceitos e Desafios de Pesquisa

A consolidação da Ubicomp vem acontecendo devido a sinergia de vários fatores, onde destacam-se o crescente uso de dispositivos portáteis e a melhora na conectividade entre diferentes dispositivos. Uma das abordagens que vem ganhando destaque no cenário mundial para a materialização de suas premissas, tendo como principal meio de comunicação a Internet, é denominada Internet das Coisas.

O termo Internet das Coisas foi cunhado no laboratório de pesquisa Auto-ID Center do MIT (*Massachusetts Institute of Technology*) e foi mencionado pela primeira vez em 1998 por Kevin Ashton, em uma apresentação onde apresenta o potencial da Internet das coisas de mudar o mundo assim como a Internet mudou (ASHTON, 2009). Em 2001 o termo surge publicado, através do artigo (BROCK, 2001). No artigo o autor introduz este novo conceito relatando o desejo de criar uma infraestrutura inteligente que liga objetos, informações e pessoas através da rede de computadores, permitindo a coordenação de recursos físicos amplamente distribuídos.

A Internet das Coisas preconiza a ideia do “tudo conectado”, onde qualquer coisa, bem como o corpo humano, os objetos e os animais poderão ter sensores capazes de gerar informações e compartilhar estas com outros recursos automaticamente.

### 2.1.1 Conceitos Associados à Infraestrutura da IoT

Um fator comum em aplicações IoT é a inerente inteligência da infraestrutura computacional, que reflete na denominação dos seus domínios, como por exemplo *smart home, smart health care, smart agriculture* e outros. Sendo parte destas "*smart*" aplicações, os dispositivos podem coletar dados automaticamente, compartilhar informações entre si, e iniciar e executar serviços com o mínimo de intervenção humana.

Um dos principais desafios está em como interligar, reconhecer e armazenar o grande número de informações geradas por ambientes IoT, de forma a constituírem informações relevantes, ditas informações de contexto para as aplicações e seus usuários, bem como reagirem de forma inteligente a estas informações. Com base neste cenário, algumas características são almejadas para infraestruturas IoT (LEE; KIM, 2010):

- Automatização: componente chave que implica no suporte do processamento e inferência contextual de forma automática por parte da infraestrutura computacional;
- Inteligência: dispositivos que possuam inteligência devem estar capacitados a operar de forma adaptativa a diferentes situações. A Ciência de Contexto vem se mostrando um componente chave para viabilizar sistemas inteligentes na IoT;
- Dinamicidade: um dispositivo pode se mover de um lugar a outro, necessitando que sua infraestrutura esteja apta para reconhecer esta mudança e consequentemente realizar a adaptação necessária baseada no seu ambiente;
- Zero Configuração: para suportar a fácil integração de dispositivos, recursos *plug and play* devem estar disponíveis, otimizando a gerência dos dispositivos e possibilitando o crescimento descentralizado de sistemas IoT.

O grande facilitador para a integração destes diversos dispositivos independentemente de sua localização, por ser uma rede global, foi a Internet. Porém alguns fatores são tidos como importantes para a efetiva materialização da IoT, como os avanços tecnológicos na miniaturização de dispositivos embarcados e inclusão de novos e diferentes sensores, atuadores e *tags* inteligentes. No entanto, há ainda uma série de desafios a serem superados para alavancar a ampla disseminação da IoT.

Um dos principais desafios inerentes ao cenário computacional típico da IoT, está relacionado com a alta heterogeneidade decorrente da diversidade de tecnologias de *hardware* e *software* presentes neste ambiente (SILVA et al., 2019). Esta situação demanda que haja uma busca de soluções que permitam a interoperabilidade e integração destes diferentes componentes.

Uma alternativa de solução promissora que vem sendo amplamente utilizada, de forma a tratar os desafios da IoT, como o da heterogeneidade, está na utilização de plataformas de *middleware*. As plataformas de *middleware* são inseridas entre as aplicações e a infraestrutura (de comunicação, processamento e sensoriamento) subjacente, provendo um meio padronizado para o acesso aos dados e serviços fornecidos pelos recursos (objetos) por meio de uma interface de alto nível (BANDYOPADHYAY et al., 2011).

### 2.1.2 Desafios de Pesquisa em IoT

De acordo com a literatura (EJAZ; ANPALAGAN, 2019), (CHAQFEH; MOHAMED et al., 2012), alguns desafios inerentes ambientes IoT merecem destaque e constituem requisitos importantes para o propósito de um sistema para IoT. A seguir alguns destes desafios:

#### *Interoperabilidade*

Dentre os requisitos, o considerado primordial e que deve ser imperativamente endereçado por uma plataforma de *middleware* para a IoT, diz respeito a interoperabilidade entre as diversas plataformas e dispositivos heterogêneos. Tal importância se deve ao fato do número crescente de dispositivos sendo utilizados e integrados a diferentes plataformas de serviços. Tais *middlewares* devem então estar capacitados a lidar com a heterogeneidade que diz respeito aos hardwares e softwares, interagindo com diferentes representações de dados.

Em (PIRES et al., 2015) é destacado que a integração de dispositivos no contexto da IoT acontece em múltiplos níveis:

- Baixo: é necessário integrar, de maneira transparente, uma grande quantidade de dispositivos físicos heterogêneos de modo a ocultar detalhes com relação à rede, aos formatos de dados empregados, e até mesmo à semântica das informações;
- Intermediário: a fim de prover serviços de valor agregado aos usuários, é necessário integrar e disponibilizar dados providos por esses dispositivos, podendo incluir simples funções de processamento de dados ou mesmo aplicações Web mais complexas;
- Alto: um modelo padronizado de programação pode promover integração no que se refere à agregação e transformação de informações providas pelos dispositivos, de modo que desenvolvedores de aplicações não necessitam ter qualquer conhecimento acerca das especificidades dos dispositivos físicos e do ambiente de rede subjacente.

Com diferentes dispositivos interoperando, pode-se fazer uso de informações providas de diferentes meios e/ou aplicações, permitindo que sejam criadas aplicações com maior valor agregado para os usuários.

### *Interfaces de Alto Nível*

A adoção de uma plataforma de *middleware* também pode contribuir para facilitar a construção de aplicações para IoT. Desta forma, o desafio reside no fato de que, a fim de permitir a criação de aplicações que combinem recursos do mundo físico disponibilizados via Web, são necessários modelos de alto nível que abstraiam os serviços e recursos físicos subjacentes. Com isso, usuários e aplicações consumidores dos dados originados dos dispositivos conectados, podem possuir acesso aos dados de forma padronizada, por meio de interfaces de alto nível, não necessitando assim lidar com funcionalidades de baixo nível para a manipulação de tais objetos.

### *Escalabilidade*

Nos primeiros dias de 2015, o mundo registrou 25 bilhões de dispositivos conectados à Internet segundo uma das maiores feiras de tecnologias do mundo, a CES 2015 (*Consumer Electronics Show 2015*). Como consequência deste crescente número de dispositivos sendo conectados, as plataformas de *middleware* para IoT devem atender ao requisito de escalabilidade e possuir capacidade para suportar requisições provindas de inúmeros dispositivos, funcionando corretamente, mesmo em situações de uso intenso. Soluções de nuvem vêm sendo utilizadas para este fim, devido à sua facilidade de provisão e uso de recursos computacionais, que podem ser alocados e liberados sob demanda, proporcionando o surgimento da chamada "Nuvem das Coisas"(em inglês, *Cloud of Things*: coT) (SOLDATOS; SERRANO; HAUSWIRTH, 2012).

Por fim, destaca-se como frente de pesquisa a descoberta de recursos, a qual surge devido a decorrência dos ambientes de IoT terem sua infraestrutura de comunicação com uma topologia dinâmica e frequentemente desconhecida, visto que dispositivos podem ser integrados ao ambiente e utilizados de maneira oportunista e não previamente planejada (BANDYOPADHYAY et al., 2011).

Considerando esta dinamicidade da IoT, é importante que uma plataforma de *middleware* possibilite a descoberta de dispositivos presentes no ambiente em questão, realizada dinamicamente a fim de atender os requisitos das aplicações.

Também destaca-se a necessidade de prover mecanismos para o gerenciamento de dispositivos, que diz respeito à capacidade de fornecer informações de localização e estado do dispositivo permitindo, dentre outras funcionalidades, desconectar algum dispositivo roubado ou não reconhecido, atualizar *software* embarcado, modificar configurações de segurança, modificar remotamente configurações de *hardware*, localizar um dispositivo perdido, apagar dados sensíveis de dispositivos, e até mesmo

possibilitar a interação entre dispositivos.

## 2.2 Execution Environment for Highly Distributed Applications: EXEHDA

Esta seção é destinada a apresentar os conceitos relacionados ao EXEHDA e seus subsistemas. O EXEHDA é um *middleware* adaptativo ao contexto, baseado em serviços que visa criar e gerenciar um ambiente celular, bem como promover a execução de aplicações sobre esse ambiente (LOPES, 2016).

O *middleware* tem como premissa principal a definição de uma arquitetura que capacite aplicações a se comunicarem em um ambiente computacional, de uma forma ciente de contexto gerindo as informações, sendo seus aspectos funcionais e não-funcionais mutáveis a partir deste controle. As aplicações alvo do *middleware* EXEHDA são distribuídas, sensíveis ao contexto e abrangem a mobilidade lógica e física. Onde mobilidade lógica se refere ao deslocamento de software ao longo do ambiente computacional, e a mobilidade física ao deslocamento do usuário, portanto seus dispositivos.

### 2.2.1 Arquitetura de Software

A arquitetura de software do EXEHDA, apresentada na Figura 1, fornece uma solução integrada para construir e executar aplicações distribuídas em grande escala. Seus principais objetivos são: (i) gerenciar tanto aspectos não-funcionais como funcionais da aplicação, de modo independente; (ii) dar suporte à adaptação dinâmica de aplicações; (iii) disponibilizar mecanismos para obter e tratar informações de contexto; (iv) empregar informações de contexto na tomada de decisões; (v) decidir as ações adaptativas de forma colaborativa com a aplicação; e (vi) disponibilizar a semântica siga-me, permitindo ao usuário iniciar as aplicações e acessar dados a partir de qualquer lugar, e executar as aplicações continuamente mesmo em deslocamento (LOPES et al., 2014).

No EXEHDA, as condições de contexto são proativamente monitoradas e o suporte à execução deve permitir que tanto a aplicação como ele próprio utilizem essas informações na gerência de seus aspectos funcionais e não-funcionais. O mecanismo de Ciência de Contexto proposto para o EXEHDA emprega uma estratégia colaborativa entre aplicação e ambiente de execução, através da qual é facultado ao programador individualizar políticas para reger o comportamento de cada um dos componentes que constituem o software da aplicação.

A estrutura de software do EXEHDA contempla um núcleo mínimo e serviços carregados sob demanda, os quais estão organizados nos seguintes subsistemas: (i) Acesso Ubíquo, (ii) Comunicação , (iii) Execução Distribuída e (iv) Adaptação e Reco-

nhecimento do Contexto (vide Figura 1).

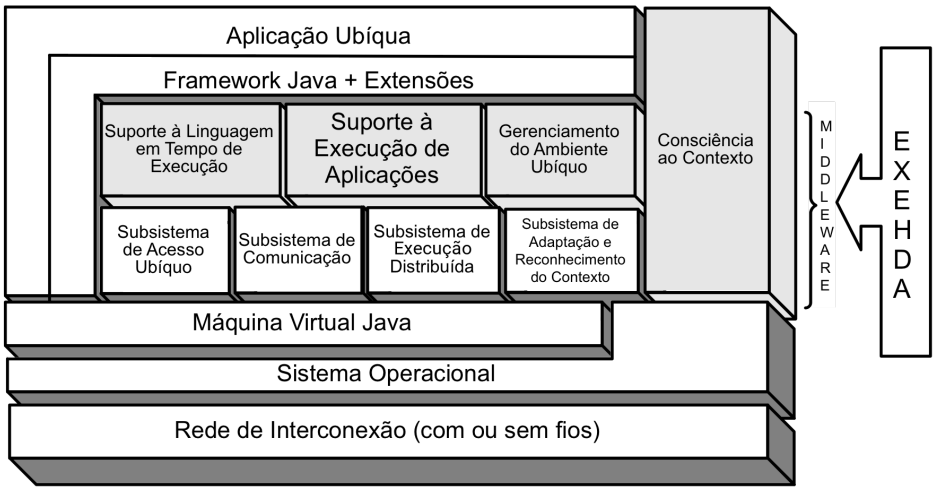


Figura 1 – Arquitetura de Software do EXEHDA. Fonte: (DAVET, 2016)

Este trabalho contribui especificamente com o Subsistema de Execução Distribuída, ampliando as funcionalidades dos serviços que tratam os aspectos de descoberta e localização de recursos.

### 2.2.2 Ambiente Celular do EXEHDA

O EXEHDA propõe um ambiente computacional constituído por células de execução, nas quais os dispositivos computacionais são distribuídos como pode ser observado na Figura 2. Cada célula é constituída dos seguintes componentes (DAVET, 2016):

- EXEHDAbase: elemento central da célula, sendo responsável por todos serviços básicos e constituindo referência para os demais elementos;
- EXEHDAnodo: corresponde aos dispositivos computacionais responsáveis pela execução das aplicações;
- EXEHDAnodo móvel: um subcaso do anterior, que corresponde aos dispositivos tipicamente móveis que podem se deslocar entre as células do EXEHDA, como *notebooks*, *tablets* ou *smartphones*;
- EXEHDAborda: responsável por fazer a interoperação entre os serviços do *middleware* e os diversos tipos de *gateways*;
- EXEHDAgateway: consiste no elemento responsável por setorizar pontos de coleta e/ou atuação distribuídos, disponíveis no meio físico, realizando a interação destes com os outros componentes do *middleware*.

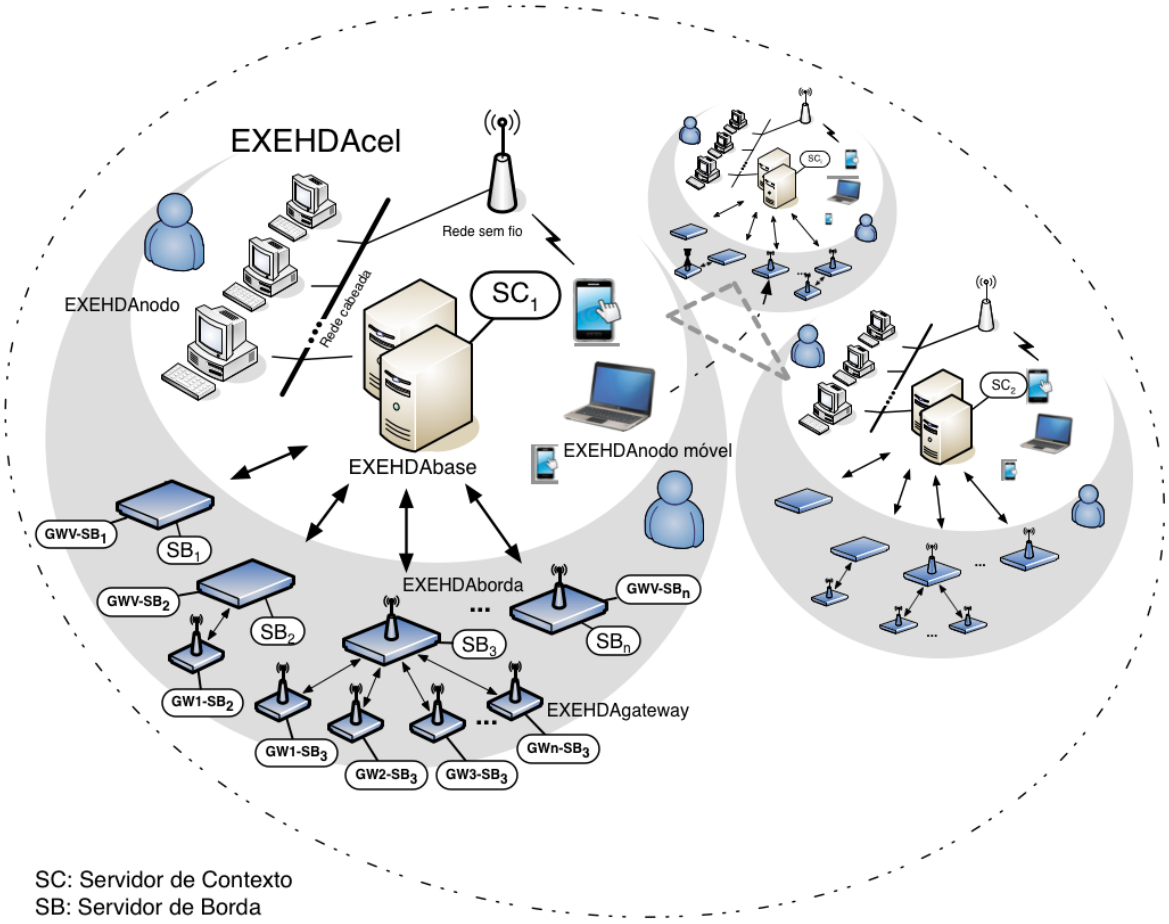


Figura 2 – Organização celular do ambiente gerenciado pelo EXEHDA. Fonte: (JOAO et al., 2018)

O Servidor de Borda é instanciado em um equipamento do tipo EXEHDAborda e se destina a gerenciar a interação com o meio físico através de *gateways*, bem como a execução de regras de contingência e armazenamento temporário das informações contextuais coletadas, em caso de falha de comunicação. O Servidor de Contexto, por sua vez, é alocado no EXEHDAbase e atua no armazenamento e no processamento das informações contextuais, integrando informações históricas e aquelas provenientes de diferentes Servidores de Borda distribuídos no ambiente celular.

Na proposta do EXEHDA a premissa é que os sensores e/ou atuadores sejam incorporados ao Servidor de Borda através de *gateways*. Os *gateways* são utilizados então, para tratar a heterogeneidade, tanto de hardware como de software, inerentes a dispositivos de sensoriamento e/ou atuação, realizando a conversão de protocolos e o gerenciamento dos dispositivos, além de fornecer a estes capacidade de comunicação via Internet.

Com o intuito de garantir a interoperabilidade com as tecnologias de mercado, e

também potencializar a distribuição das iniciativas de coleta e/ou atuação, foram previstos três tipos distintos de *gateways* (DAVET, 2016):

- Gateways Proprietários: tem funcionalidades e protocolos heterogêneos variando de acordo com seus fabricantes;
- Gateways Nativos: suas funcionalidades operam de maneira integrada à arquitetura do EXEHDA, sendo instanciados em equipamentos do tipo EXEHDAgateway;
- Gateway Virtual: localizado no mesmo hardware em que executa o Servidor de Borda, constituindo-se em uma virtualização do Gateway Nativo, dispensando quando possível a necessidade de um hardware específico para o *gateway*.

Os ambientes computacionais da atualidade, como os providos pela IoT, são bastante complexos, compreendendo sistemas legados, sistemas fechados provenientes de empresas e sistemas desenvolvidos por grupos de pesquisas interessados. São caracterizados por sua distribuição e heterogeneidade, características comuns em ambientes computacionais.

Com base nisso, percebe-se a dificuldade no desenvolvimento de sistemas com sincronização de dados e desenvolvimento de novas funcionalidades que supram necessidades requisitadas por serviços na perspectiva da IoT.

### 2.2.3 Subsistemas do EXEHDA

O middleware EXEHDA é formado por quatro subsistemas (LOPES, 2016): (i) Subsistema de Execução Distribuída; (ii) Subsistema de Comunicação; (iii) Subsistema de Acesso Ubíquo; e (iv) Subsistema de Reconhecimento de Contexto e Adaptação. Os subsistemas do EXEHDA são descritos da seguinte forma:

*Subsistema de Execução Distribuída*

O Subsistema de Execução Distribuída é responsável pelo suporte ao processamento distribuído no EXEHDA. No intuito de promover uma execução transparente, este subsistema interage com outros subsistemas do EXEHDA. Em específico, interage com o Subsistema de Reconhecimento de Contexto e Adaptação, de forma a prover comportamento distribuído e adaptativo às aplicações da IoT.

*Subsistema de Comunicação*

A natureza da mobilidade do hardware e, na maioria das vezes, também a do software, não garante a interação contínua entre os componentes da aplicação distribuída. As desconexões são comuns, não somente devido à existência de alguns links sem fio, mas sobretudo como uma estratégia para economia de energia nos dispositivos

móveis. O Subsistema de Comunicação do EXEHDA disponibiliza mecanismos que atendem estes aspectos da IoT.

*Subsistema de Acesso Ubíquo*

A premissa de acesso em qualquer lugar e todo o tempo, requer um suporte do *middleware* em aspectos relacionados a recuperação de códigos disponibilizados remotamente, de disponibilização de um ambiente virtual para suporte para acesso as aplicações e dados aos usuários e um suporte para gerência da sessão do usuário que dentre outros aspectos deverá prover recursos para *login*.

*Subsistema de Reconhecimento de Contexto e Adaptação*

Este subsistema inclui serviços que tratam desde a extração da informação bruta sobre as características dinâmicas e estáticas dos recursos que compõem o ambiente celular, passando pela identificação em alto nível dos elementos de contexto, até o disparo das ações de adaptação em reação a modificações no estado de tais elementos de contexto.

## 2.3 Descoberta de Recursos na Internet das Coisas

Segundo (CHOUDHARY; JAIN, 2017) recurso é qualquer dispositivo que pode prover sensoriamento, atuação, arquivos, sistema de arquivos, memória, capacidade de CPU, capacidade de comunicação (por exemplo, modem de rádio) ou até mesmo hardwares embarcados. Na Internet das Coisas, um recurso pode ser caracterizado por um dispositivo, o qual possui informações relevantes para o usuário. Por sua vez, a descoberta de recursos na IoT é um procedimento para detecção e verificação da disponibilidade de um recurso na infraestrutura computacional.

Em um procedimento de descoberta de recursos, usualmente um protocolo de descoberta tem como função prover meios de propagação de buscas por um recurso em outros dispositivos da rede. A diferença entre os protocolos, costuma residir na forma como os recursos são organizados e como é feito o encaminhamento de buscas, podendo haver impactos em aspectos de escalabilidade e desempenho (CHOUDHARY; JAIN, 2017).

Os serviços de descoberta de recursos podem ser organizados de três formas segundo o artigo de referência na área (MESHKOVA et al., 2008): (i) estruturados, onde há uma estrutura rígida de interconexão dos clientes além de armazenar um índice de recursos em cada cliente; (ii) não estruturados, onde cada cliente é conectado aleatoriamente a outros clientes sem armazenar informações a respeito dos seus recursos; e (iii) híbridos os quais tentam superar as desvantagens das outras duas abordagens e manter os benefícios de cada uma.

### 2.3.1 Requisitos para Descoberta de Recursos

Visando atender as demandas da Internet das Coisas, os serviços de descoberta de recursos devem atender requisitos funcionais e não funcionais. Buscar, publicar, modificar, remover e classificar recursos são funcionalidades essenciais para a operação de serviços de descoberta. Ao mesmo tempo que estes requisitos devem atender as demandas de descoberta, deve-se levar em consideração aspectos não funcionais como o tempo de resposta, a confiabilidade e segurança dos dados, a disponibilidade e a escalabilidade dos sistemas (KRITIKOS; PLEXOUSAKIS, 2016).

#### *2.3.1.1 Funcionais*

Os aspectos funcionais são aqueles necessários para a execução das ações de gerência da descoberta. Um serviço de descoberta deve agir no meio capacitando os clientes ou aplicações a utilizar qualquer recurso disponível na rede. Podemos citar como exemplo de requisitos funcionais:

- Buscar: no que diz respeito a localização de recursos, o método de Busca tem como objetivo encontrar os recursos independentemente da sua localização na rede ou geoespacial;
- Publicar: os estados e informações contextuais devem ser persistidos, armazenados em um banco de dados a fim de estar disponível para qualquer usuário ou aplicação. Também é necessário para haver dados históricos de cada recurso e seus estados;
- Modificar: existem dados específicos que devem poder ser manipulados pelos sistemas de gerência de dados. Este tipo de requisito possibilita que informações se mantenham sempre atualizadas;
- Remover: remover recursos é uma funcionalidade disponível para poucos casos, mas ainda assim é importante para manter a consistência das informações caso necessário;
- Classificar: quando um usuário ou aplicação necessita fazer uso de um recurso desconhecido, é feita uma listagem levando em consideração os atributos definidos. Com base nesses atributos é feita uma classificação, assim os recursos são classificados e apresentados para o usuário/cliente de acordo com as suas necessidades.

#### *2.3.1.2 Não Funcionais*

Aspectos não funcionais dizem respeito a condições necessárias para que os sistemas de descoberta de recursos atuem de acordo com as demandas da IoT (KARIMI; KHAYYAMBASHI, 2015). Os aspectos não funcionais necessários são:

- Tempo de resposta: os sistemas mais resilientes necessitam que o tráfego de requisições e informações aconteça de maneira ágil. Para isso é necessário que os sistemas respondam a requisições com um baixo tempo de resposta;

- Confiabilidade: deve-se haver uma preocupação constante em relação às várias falhas que podem ocorrer em um sistema, tais como perda de comunicação de algum dispositivo ou perda de dados. Em um sistema centralizado, uma falha de dados pode ser catastrófica e os sistemas devem estar preparados prevendo estes tipos de falhas;

- Segurança: a Internet das coisas trabalha não só com informações públicas, mas também com informações pessoais e privadas. Os sistemas devem garantir que estas informações sejam acessadas apenas por aqueles habilitados;

- Disponibilidade: a arquitetura deve ter como premissa garantir que quando um cliente ou aplicação solicite um recurso, o mesmo se encontre atualizado e disponível, do contrário as informações com respeito ao status devem ser persistidas. Neste caso, os sistemas podem trabalhar com a atualização de status e com a redundância de armazenamento;

- Escalabilidade: considerando o escopo da IoT, como um ambiente altamente escalável, a preocupação em não perder desempenho à medida que há um aumento do volume de dados é essencial e inerente as necessidades da IoT.

### 2.3.2 Arquiteturas P2P na Descoberta de Recursos

Cada serviço de descoberta de recursos pode ser executado em camadas de sobreposição estruturadas, não estruturadas ou híbridas (MESHKOVA et al., 2008). As arquiteturas com sobreposições estruturadas são ainda subdivididas em centralizadas (cliente-servidor) ou descentralizadas (cliente-cliente). As híbridas tentam combinar as vantagens oferecidas pelos diferentes tipos de arquitetura para melhorar o desempenho geral do sistema. Detalhes mais específicos serão discutidos a seguir.

#### *2.3.2.1 Arquiteturas Estruturadas*

Redes com arquiteturas P2P estruturadas (MESHKOVA et al., 2008) fornecem uma arquitetura de redes em múltiplas camadas auto organizáveis para aplicações P2P de larga escala. Estas redes podem implementar uma *Distributed Hash Table* (DHT) ou Tabela Hash Distribuída escalável e tolerante a falhas, na qual qualquer item pode estar localizado em uma determinada distância medida em número de saltos. Esta quantidade de saltos é definida através da tabela de roteamento de cada nó.

Descentralizadas

Em arquiteturas estruturadas descentralizadas, a distribuição da informação relativa aos serviços é rigidamente controlada pelo serviço de descoberta. Não existem servidores centrais e a informação é distribuída quase uniformemente entre todos os nós participantes de uma maneira pré-definida.

Nas arquiteturas descentralizadas, a exemplo do Universal Plug and Play (UPnP) que utiliza o protocolo Simple Service Discovery Protocol (SSDP), 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*.

Arquiteturas desse tipo geralmente usam DHTs (LUA et al., 2005) (RUSLAN et al., 2019). Com esse mecanismo, os nós precisam indexar ou armazenar dados que podem não estar interessados. Um algoritmo de hash é usado para identificar o nó que armazena o recurso pesquisado.

Propostas baseadas em DHT, como por exemplo as redes eDonkey baseadas em Kadmelia, denominadas Overnet e Kad, são exemplos de uso de arquiteturas estruturadas descentralizadas.

Centralizadas

As arquiteturas estruturadas centralizadas seguem uma abordagem cliente-servidor clássica, em geral, os servidores armazenam informações sobre os serviços que outros nós fornecem. Quando um usuário deseja encontrar um determinado serviço, ele envia solicitações ao servidor (geralmente chamado de diretório) que retorna uma resposta. SLP e Napster (MESHKOVA et al., 2008) são exemplos de uso de arquiteturas centralizadas estruturadas.

As arquiteturas estruturadas e centralizadas têm normalmente o tempo de pesquisa mais rápido, mas têm potencialmente um único ponto de falha e são vulneráveis a ataques de negação de serviço ou *Denial of Service* (DoS).

#### *2.3.2.2 Arquiteturas Não Estruturadas*

As arquiteturas P2P não estruturadas oferecem suporte a consultas parciais e complexas. O uso desta arquitetura torna a rede resistente a falhas de nós e ataques de DoS. Os sistemas se adaptam facilmente a junções de nós frequentes e se desmembram. Por outro lado, arquiteturas P2P não estruturadas não descobrem itens raros com tanta eficiência quanto os mais populares ou mais requisitados na rede.

O desempenho de arquiteturas P2P não estruturadas depende muito dos mecanismos de busca implementados. Normalmente, arquiteturas puramente não estruturadas, por exemplo, como a utilizada pelo Gnutella, não se adaptam bem a redes

muito grandes. É por isso que os *frameworks* P2P puramente não estruturados foram substituídos atualmente por sobreposições de P2P que utilizam uma arquitetura híbrida.

#### *2.3.2.3 Arquiteturas Híbridas*

Finalmente, as arquiteturas P2P híbridas tentam combinar as vantagens oferecidas pelos tipos de arquiteturas estruturadas e não estruturadas. Isto é, oferecem a possibilidade de fazer uso da descentralização das informações e ao mesmo tempo manter agrupado as informações que possuem maior correlação. Na maioria das vezes, eles usam a técnica de *clustering* para introduzir uma estrutura flexível à redes não estruturadas.

*Clustering* é a técnica mais popular para a criação de redes híbridas. O principal objetivo do *clustering* é manter as propriedades desejáveis de *overlays* desestruturados como arquitetura livre e, ao mesmo tempo, melhorar a escalabilidade do sistema (MESHKOVA et al., 2008). Os dois principais tipos de agrupamento são descritos abaixo:

- *Clustering* por Super Nodo: neste método de *clustering*, os chefes de *cluster* são eleitos dinamicamente entre os nós na rede. Então, quando o recurso é localizado por um chefe de *cluster* em outro super-nó, esse super-nó geralmente encaminha a consulta para o respectivo nó folha que hospeda o recurso necessário. Também é possível que o chefe do *cluster* não encaminhe a consulta para o nó folha, mas envia diretamente a resposta para o nó solicitante em nome do nó folha;

- Similaridade e agrupamento associativo: esta sobreposição é construída sobre as sobreposições originais com base em algumas regras que definem a similaridade entre os nós. Por exemplo, para sobreposições semânticas, a similaridade fica por conta do conteúdo armazenado. No agrupamento associativo, a semelhança é definida como o interesse de um nó na área específica. Normalmente, cada nó pode pertencer a vários *clusters* de similaridade, dependendo dos tipos de objetos nos quais ele armazena ou está interessado. A abordagem se baseia na suposição de alta correlação de dados. Se um nó procurar um serviço pertencente a um grupo lógico, é muito provável que ele também pergunte posteriormente por um serviço do mesmo grupo.

### 2.3.3 Busca de Recursos

Uma capacidade essencial dos sistemas de descoberta é buscar por recursos interessantes para a aplicação. Esta capacidade possibilita que cada aplicação busque por recursos disponíveis em células vizinhas. Esta busca pode acontecer de forma

sistemática onde os recursos são buscados de forma estruturada, ou seja, com uma forma definida, ou de forma assistemática na qual a busca acontece de modo randômico, sem uma estrutura previamente definida.

#### *2.3.3.1 Busca Sistemática*

Normalmente, os métodos de busca sistemática exploram uma árvore ou grafo, o qual é pesquisado de acordo com algumas regras predefinidas. Não há lugar para escolha probabilística ou aleatória em tais métodos. Muitas vezes, métodos sistemáticos conduzem uma busca completa ou quase completa do grafo estudado, dependendo das regras de busca definidas.

#### *2.3.3.2 Busca Assistemática*

Os métodos de busca assistemática dizem respeito a caminhos randômicos percorridos pelo buscador. Este método procura por vizinhos em passos sucessivos, começando com um número fixo de vizinhos em direções aleatórias, podendo ou não ter um *Time to Live* (TTL);

A busca também pode ser probabilística, neste caso existem duas abordagens: (i) para cada nó em que a consulta atingir, um número aleatório é gerado (0 ou 1). Se esse número exceder um limite, o nó será procurado, caso contrário, é ignorado; e (ii) as réplicas de consulta são encaminhadas apenas para uma porcentagem (p) dos vizinhos do nodo.

### 2.3.4 Tabela Hash Distribuída

Em arquiteturas estruturadas descentralizadas, a distribuição da informação relativa aos serviços é rigidamente controlada pelo serviço de descoberta. Não existem servidores e as informações são distribuídas entre todos os nós participantes de uma maneira predefinida.

Os sistemas de descoberta baseados em DHT permitem encontrar uma localização a partir do valor de uma chave (BALAKRISHNAN et al., 2003) (SAVANAH; WRIGHT, 2019). Os sistemas que implementam este princípio são totalmente descentralizados, escaláveis, conseguem balanceamento de carga, fornecem uma certa tolerância a falhas e a maioria deles tem correção e desempenho teoricamente prováveis.

Podemos dizer que as principais desvantagens de tais sistemas se encontram na sua incapacidade de suportar consultas complexas, isto é, suportam apenas a pesquisa de correspondência exata. Outro problema do qual a maioria dos sistemas P2P básicos sofrem é um baixo nível de segurança. Os sistemas são vulneráveis, por exemplo, a ataques de Trojan e Man-In-The-Middle (LUA et al., 2005).

As implementações de redes de sobreposição com base em DHT diferem em como

nós e conteúdos são associados e como o roteamento para cada nó responsável por um dado identificador é definido. Podemos citar como principais algoritmos de roteamentos, que fazem uso de DHT, o CAN, Chord, Pastry, Tapestry e Kademlia os quais criam sobreposições capazes de organizar e associar conteúdos e recursos dentro de uma rede de arquitetura estruturada. Estes serão descritos na sequência.

#### *2.3.4.1 Roteamento Content Addressable Network*

O roteamento feito pelo Content Addressable Network (CAN) (RATNASAMY et al., 2001) (SHEN; GUO, 2019) consiste em um espaço de coordenadas cartesiano multi-dimensional virtual, essa estrutura é completamente lógica. Cada nó é atribuído a uma área exclusiva no espaço de coordenadas. As coordenadas dessa área identificam o nó e são usadas no roteamento.

O CAN usa a estratégia de roteamento gulosa, em que uma mensagem é roteada para o vizinho do nó atual, que está mais próximo do local desejado. Portanto, para o roteamento efetivo, um nó precisa conhecer apenas as coordenadas de seus vizinhos e seus endereços IP correspondentes. Essa estratégia de roteamento requer um espaço de coordenadas contínuo sem espaços “vazios” e não atribuídos. Assim, quando um nó ingressa ou sai do sistema, o espaço precisa ser realocado dinamicamente, de modo que não haja espaços “vazios” e cada nó tenha uma determinada zona a ser controlada. Uma nova zona é geralmente obtida dividindo-se uma zona de algum nó aleatório em duas partes. A ausência de zonas não alocadas é garantida pela ampliação das zonas dos nós cujo vizinho acabou de sair da rede.

A alocação dos *peers*, par (chave, valor), também é feita usando o espaço de coordenadas. Os *peers* são mapeados uniformemente usando uma função hash e cada nó hospeda os *peers* que foram alocados para sua zona.

O algoritmo CAN é altamente escalável, tolerante a falhas e auto organizado. Seu roteamento de pesquisa requer $O(d \cdot N^{1/d})$ mensagens e o tamanho da tabela de roteamento não excede 2d entradas, onde d é o número de dimensões em CAN. O número de mensagens necessárias para reagir a um nó que se junta ou sai do sistema também é 2d.

#### *2.3.4.2 Roteamento Chord*

O roteamento apresentado por Chord (STOICA et al., 2001) (JOGUNOLA et al., 2017) é organizado em formato de um anel. Os pacotes são sempre enviados para uma direção (por exemplo, no sentido horário) até que eles cheguem ao nó de destino. Para que este sistema funcione, o Chord requer para cada nó apenas o conhecimento de seu sucessor. No entanto, para acelerar o processo de entrega, cada nó armazena os elementos $O(logN)$ em sua tabela de roteamento. Na tabela de roteamento, um nó armazena um ponteiro para a metade do anel do nó, uma entrada para o nó de posição

referente a um quarto do anel, um referente a um oitavo do anel e assim por diante. Portanto, cada nó conhece melhor sua vizinhança mais próxima que nós distantes.

Os *peers*, par (chave, valor), são alocados no primeiro nó, cujo ID é igual ou maior ao valor da chave. Chord usa uma *Hash* consistente para garantir a distribuição uniforme das teclas entre os nós. Para tornar o nó tolerante a falhas do sistema, além de manter uma lista de sucessores, também possui um ponteiro para seu predecessor. O Chord executa um protocolo de estabilização, que verifica periodicamente a consistência dos ponteiros sucessores/predecessores imediatos dos nós.

No Chord, o roteamento de pesquisa requer $O(logN)$ mensagens e a tabela de roteamento consiste na mesma quantidade de mensagens. A atualização dos sistemas no caso de um nó se unir ou sair resulta em $O(log^2N)$ mensagens.

#### *2.3.4.3 Roteamento Pastry*

O roteamento de Pastry (ROWSTRON; DRUSCHEL, 2001) (MALATRAS, 2015) é organizado de maneira híbrida. A procura por um par (chave, valor) pode ser feita em uma árvore ou, na proximidade de um nó de destino, em forma de anel. A abordagem de anel também é usada quando o roteamento de árvore falha.

Para manter essa geometria híbrida, um nó precisa hospedar três tipos de tabelas que auxiliam o processo de roteamento. O tamanho de uma tabela usada para o roteamento de árvore em Pastry é aproximadamente $(log_BN) \cdot (B-1)$ entradas. Ele contém metade dos IDs de nós numericamente maiores e metade dos IDs de nós menores. O algoritmo de Pastry não falhará até que metade dos nós da mesa foliar falhe simultaneamente. Outra tabela que cada nó mantém é a tabela de vizinhança. Ele é usado para roteamento de localidade, isto é, para rotear pacotes via nós fisicamente próximos. Neste caso, o tamanho da tabela hash utilizada pelo Pastry é de $2 \cdot 2^B$ entradas.

Pastry tem as seguintes características: (i) o roteamento é feito em mensagens $Olog_BN$; (ii) o número de entradas da tabela de roteamento por nó é $2 \cdot B \cdot log_BN$; e (iii) o número de mensagens adicionais necessárias para um nó ingressar ou sair da rede é $log_BN$.

#### *2.3.4.4 Roteamento Tapestry*

O roteamento de Tapestry (ZHAO et al., 2004) (MALATRAS, 2015) é baseado em árvore e usa correlação entre um ID de nó e uma chave para rotear uma mensagem. O prefixo no ID do nó na entrada da tabela de roteamento é comparado ao da chave. Se seus IDs compartilharem um prefixo comum pelo menos em mais de um dígito em comparação com o ID do nó atual, a mensagem será encaminhada para o nó considerado.

Para realizar uma pesquisa, cada nó precisa manter as entradas $log_BN$, onde B é

tipicamente igual a 4. Isso garante a entrega de um pacote em $O(log_B N)$ saltos. A consistência do sistema é restaurada no caso de uma junção/saída do nó em mensagens $log_B N$.

#### *2.3.4.5 Roteamento Kademlia*

Kademlia (MAYMOUNKOV; MAZIERES, 2002) (CHEN et al., 2016) é um roteamento de redes P2P descentralizadas organizada em torno da operação XOR. Ambos os endereços de *peer* (Node IDs) e as chaves recebem valores no espaço de 160 bits. Os *peers* (chave, valor) são armazenados nos nós, cujos IDs de Nó estão próximos das chaves em termos de métrica XOR $(Distncia = chave \bigoplus NodeID)$. Para localizar a chave, o algoritmo de roteamento do Kademlia usa a mesma métrica XOR para estimar a distância entre a chave e a ID do nó de um par. O nó envia a consulta para esses nós da sua tabela de roteamento que estão mais próximos da chave desejada. O processo de pesquisa é interrompido quando o par necessário (chave, valor) é recuperado. Além disso, o armazenamento em cache desse par (chave, valor) é feito no nó mais próximo da chave, que ainda não possui a réplica do par.

O armazenamento em cache faz sentido para o Kademlia, pois ele usa uma métrica unidirecional, que garante que todas as pesquisas para a mesma chave convergem para o mesmo caminho, independentemente do *peer* que emite a solicitação. O cache, portanto, permite aliviar pontos de acesso ao longo do caminho de pesquisa.

Na tabela de roteamento, cada *peer* armazena informações (ID do nó, porta UDP, endereço IP) sobre outros *peers* localizados na distância de $2^i$ a $2^{i+1}$ de si mesmo $(0 < i < 160)$. Para realizar o roteamento, o Kademlia mantém listas especiais, chamadas *k-buckets*, onde k é um número inteiro do sistema, por exemplo 20. Cada *k-buckets* armazena uma lista de nós que estão situados na faixa específica de distâncias do considerado nó. A distância é obtida pela operação XOR conduzida nos IDs do nó, diferentes nós são colocados em um *bucket* se as distâncias do nó de origem corresponderem no bit mais alto. Por exemplo, os nós que têm métricas de distância de “110” e “101” serão colocados em um único intervalo; os nós com métricas “110” e “011” pertencem a listas diferentes.

O Kademlia atualiza os *k-buckets* quando recebe mensagens de outros nós. Este processo é otimizado para manter os *peers* mais ativos sempre na tabela de roteamento, já que foi demonstrado que quanto mais tempo o par permanecer na rede, menos provável será que ele falhará no futuro (SAROIU; GUMMADI; GRIBBLE, 2001).

O Kademlia requer mensagens $(O(log_B N) + c)$ para o roteamento de pesquisa, onde c é uma constante pequena. O tamanho da tabela de roteamento é $(Blog_B N + B)$ e o número de mensagens de atualização para os nós que saem ou entram é $(log_B N + c)$. O exemplo mais conhecido de uso do Kademlia é a rede de sobreposição P2P do eDonkey (O’CONNOR et al., 2004).

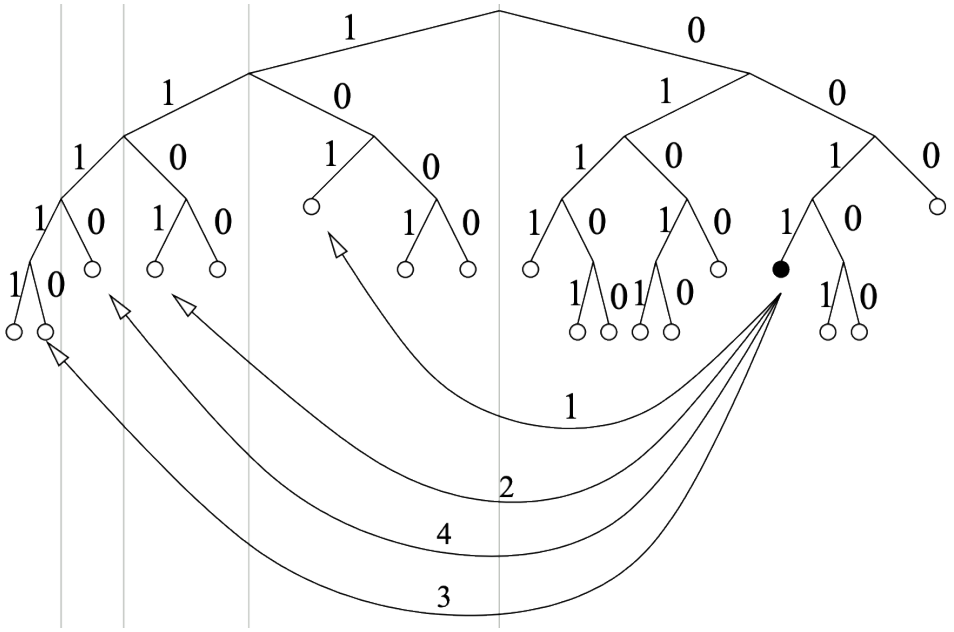


Figura 3 – Representação de busca efetuada pelo Kademlia.

A Figura 3 representa a localização de um nodo por meio de seu identificador numérico. Neste caso, um nodo com o prefixo 0011 busca por um nodo com o prefixo 1110 o qual efetua consultas em nodos sucessivos, atingindo nodos cada vez mais perto do desejado.

### 2.3.5 Propagação de Mensagens

A escolha das estratégias de propagação de mensagens pode depender de uma variedade de parâmetros, como a topologia de rede, os meios de comunicação ou a escala prevista (ASADI; WANG; MANCUSO, 2014). Abaixo, é apresentada uma visão geral dos quatro principais métodos de propagação de mensagens usados na maioria das redes, especialmente aqueles baseados em IP:

- *Unicast*: os protocolos que envolvem apenas um remetente e um receptor, isto é, comunicação um-para-um, são referidos como protocolos *unicast*. *Unicasting* pode ser empregado para enviar solicitações para nós (servidores) dedicados que possuem informações sobre serviços que outros nós fornecem. A técnica também é empregada quando o nó que hospeda o serviço já está localizado, mas as informações sobre o serviço precisam ser atualizadas. O *unicast* é frequentemente utilizado em protocolos de pesquisa complexos, por exemplo, o passeio aleatório e seus derivados;

- *Broadcast*: os protocolos de transmissão usam a comunicação de um para todos, em que o pacote é enviado para todos os nós da sub-rede ou de toda a rede. As estruturas de descoberta de serviço empregam a transmissão para descobrir recursos na ausência de servidores. A transmissão é frequentemente usada

para anunciar um novo serviço que aparece no sistema. A técnica é amplamente utilizada em redes sem fio, devido à natureza de transmissão dos meios de comunicação;

- *Multicast*: os protocolos *multicast* empregam comunicação um-para-muitos que permite que um remetente propague um pacote para o grupo de nós com uma única operação de envio. Este método de propagação de pacotes requer um mecanismo relativamente complexo a ser implementado no nó uma vez que a manutenção do grupo *multicast* impõe claramente sobrecarga extra na rede. Para redes sem fio com mobilidade, o custo do suporte a *multicast* pode ser muito alto e pode ser mais eficiente usar a transmissão simples;

- *Anycast*: funciona da mesma forma que *multicast*, um método de comunicação um-para-muitos. No entanto, os pacotes são entregues não para todos os pontos de destino *anycast*, mas apenas para um deles que está em conformidade com os parâmetros fornecidos. Basicamente, no modo *anycast*, temos uma comunicação ponto-a-ponto com o ponto final escolhido de um conjunto de destinos disponíveis. O nó de destino pode ser determinado usando uma variedade de métricas, como proximidade ao remetente, carga mínima, número de vizinhos, entre outros. Assim como no *multicast*, todos os nós pertencentes ao mesmo grupo *anycast* estão ouvindo um endereço *anycast*.

### 2.3.6 Descrição de Recursos

Propostas de descoberta de recursos, além de possuirem processos para a busca, verificação e disponibilização de recursos, usualmente necessitam de padrões para a descrição através de metadados (FAHEEM et al., 2018).

O uso efetivo de metadados entre aplicativos, no entanto, requer convenções comuns sobre semântica, sintaxe e estrutura. Comunidades de descrição de recurso individuais definem a semântica, ou o significado, de metadados que atendem a suas necessidades específicas. A sintaxe, o arranjo sistemático dos elementos de dados para o processamento de máquinas, facilita a troca e o uso de metadados entre vários aplicativos. A estrutura pode ser pensada como uma restrição formal na sintaxe para a representação consistente da semântica.

Um dos principais frameworks para a descrição de recursos é o RDF (*Resource Description Framework*). O RDF fornece um modelo para descrever recursos como pode ser observado na Figura 4. Os recursos possuem propriedades, atributos ou características. O RDF define um recurso como qualquer objeto exclusivamente identificável por um URI (*Uniform Resource Identifier*).

As propriedades associadas aos recursos são identificadas por tipos e valores. Tipos de propriedade expressam os relacionamentos de valores associados a recursos.

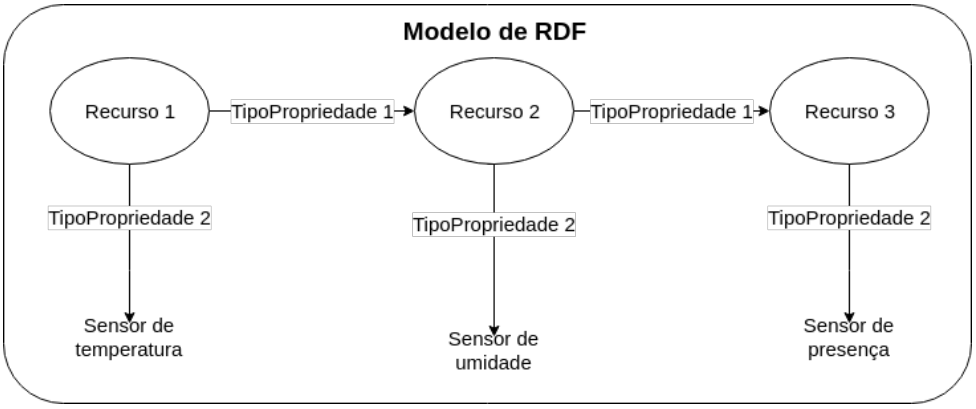


Figura 4 – Descrição RDF genérica.

No RDF, os valores podem ser de natureza atômica (texto, strings, números) ou outros recursos, que por sua vez podem ter suas próprias propriedades. Uma coleção dessas propriedades que se refere ao mesmo recurso é chamada de descrição. No núcleo do RDF está um modelo independente de sintaxe para representar recursos e suas descrições correspondentes.

## 2.4 Considerações Finais do Capítulo

Neste capítulo foi apresentada a fundamentação conceitual necessária para o entendimento da descoberta de recursos na perspectiva do cenário computacional oferecido pela IoT. Para isso, uma revisão dos conceitos, requisitos e desafios de pesquisa associados a descoberta de recursos é apresentada em um cenário de elevada escalabilidade e heterogeneidade.

A descrição da arquitetura do *middleware* EXEHDA e seus subsistemas, foi introduzida, bem como sua organização celular revisada, já considerando a perspectiva de integração proposta EXEHDA-HS ao mesmo. Ao final, os conceitos e procedimentos específicos de arquiteturas P2P para Descoberta de recursos são descritos, apresentando seus requisitos e estratégias de implementação.

# 3 TRABALHOS RELACIONADOS A DESCOBERTA DE RECURSOS

Tendo como base as premissas que motivaram o desenvolvimento desta dissertação de mestrado, foram identificados na literatura trabalhos relacionados. Os trabalhos foram selecionados tendo como critério central o tratamento dos desafios de descoberta de recursos na IoT. Este Capítulo divide os trabalhos em arquiteturas descentralizadas e arquiteturas centralizadas. É apresentada uma descrição dos trabalhos selecionados e ao final é feita uma análise comparativa dos mesmos.

## 3.1 Arquiteturas Descentralizadas para Descoberta de Recursos

Nesta seção serão apresentados trabalhos que utilizam arquiteturas descentralizadas para a descoberta de recursos, ou seja, que possuam as informações utilizadas para a descoberta distribuídas por toda a rede. Os trabalhos serão descritos considerando a sua estrutura de dados, o modo de comunicação empregado, método de busca e o cenário de uso.

### 3.1.1 A DHT-Based Discovery Service for the Internet of Things

O trabalho (PAGANELLI; PARLANTI, 2012) tem como objetivo propor um serviço de descoberta capaz de lidar com buscas utilizando multiatributos, ou seja, fazendo uso de mais de uma característica para a busca.

Neste trabalho, os autores elegeram os seguintes requisitos funcionais para um serviço de descoberta, o expressão é associada ao conceito de recurso:

- Esquema de identificação flexível: o mecanismo de descoberta deve ser transparente em relação ao esquema de identificação adotado. Por exemplo, em aplicações de RFID para logística, o *Eletronic Product Code* (EPC) é um esquema de identificação amplamente adotado. Outros esquemas de identificação disponíveis incluem URIs, endereços IPv6, *Universal Product Code*, apenas para mencionar alguns exemplos;

- *Multiattribute Query*: o mecanismo de descoberta deve ser capaz de manipular uma consulta para uma correspondência exata de um determinado identificador, bem como consultas contendo outros atributos qualificados, por exemplo, local e categoria;
- Intervalo de consulta: além das consultas de correspondência exata, o sistema deve oferecer suporte a consultas especificando limites inferiores e superiores em um único ou vários atributos;
- Múltiplos *Publishers*: dependendo das finalidades do aplicativo, várias entidades podem ser chamadas para produzir e publicar informações sobre um determinado objeto, além do proprietário do objeto;
- APIs de gerenciamento: as entidades autorizadas devem poder adicionar, atualizar e excluir informações associadas a um determinado objeto.

A arquitetura de descoberta utilizada pode ser observada na Figura 5, onde cada nodo faz uso de 4 camadas distintas: (i) APIs específicas de aplicativos para pesquisa e gerenciamento de informações relacionadas à descoberta; (ii) uma técnica de linearização *Space Filling Curves* (SFC) para mapear um domínio multidimensional em um unidimensional; (iii) uma estrutura de busca por meio da *Prefix Hash Tree* (PHT) baseada na interface genérica DHT get/put; e (iv) uma implementação DHT baseada nos algoritmos do Kademlia.

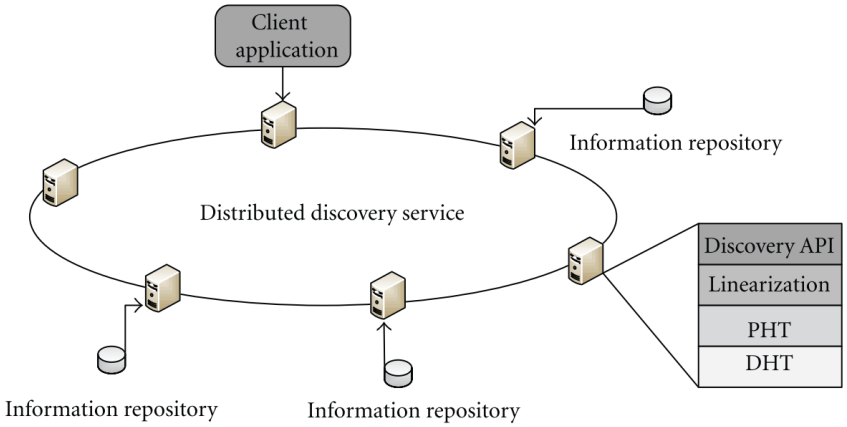


Figura 5 – Visão funcional do serviço de descoberta distribuída proposto.
Fonte: (PAGANELLI; PARLANTI, 2012)

Com relação a organização estrutural utilizada, optaram por usar a estrutura de dados distribuída chamada *Prefix Hash Tree* (PHT). Como uma estrutura de dados PHT pode ser construída sobre uma implementação genérica de DHT, as principais vantagens dessa abordagem são a promoção de um projeto em camadas e, portanto, modularidade, facilidade de projeto, implementação e manutenção.

A API exposta a aplicativos, referente a primeira camada, suporta as seguintes operações:

- Inserir um novo objeto: o cliente deve fornecer os valores dos atributos para cada recurso. Esses valores são processados pelo sistema para gerar a chave PHT correspondente. O sistema insere a chave PHT na estrutura e as informações associadas (URL e timestamp) nos nós DHT subjacentes;
- Adicionar/excluir um registro de informações associado a um objeto: o sistema usa a chave PHT associada ao objeto para localizar os registros de informações armazenados no DHT e atualizá-los adicionando/excluindo o registro fornecido;
- Recuperar os registros de informações para um determinado objeto: o sistema usa a chave PHT associada aos parâmetros de entrada fornecidos para localizar os nós DHT e recuperar registros de informações armazenadas;
- Recuperar os registros de informações para um conjunto de objetos que satisfaçam uma consulta de intervalo sobre (um conjunto de) atributos: o sistema explora a estrutura baseada em trie para recuperar as chaves PHT que estão no intervalo de consulta e reunir os objetos de informações associados com o recuperado.

Supondo que os objetos possam ser caracterizados por n-atributos, incluindo o identificador de objeto (por exemplo, EPC, esquemas de identificação de URI), consequentemente, o domínio de dados de destino é um espaço n-dimensional. Então, na camada de linearização, é aplicado uma técnica baseada em curvas de preenchimento de espaço para mapear um domínio de dados n-dimensional em um domínio de dados unidimensional. O domínio de dados unidimensional derivado pode assim ser facilmente indexado na estrutura PHT.

Por fim, um estudo de caso foi realizado em um cenário para rastreamento de mercadorias em uma cadeia de transporte multimodal. Nesse cenário, é definido um conjunto de atributos, além do identificador de item de mercadorias (ou seja, o número do EPC), que pode caracterizar de maneira útil as informações adquiridas e armazenadas durante as etapas da rota de transporte e que podem ser usadas para facilitar a recuperação de informações.

### 3.1.2 Um Serviço de Descoberta Ciente de Contexto para Internet das Coisas

O trabalho (GOMES, 2016) tem como objetivo apresentar o QoDisco, um serviço de descoberta ciente de contexto para IoT. O QoDisco é estruturado sobre uma arquitetura distribuída (vide Figura 6) composta por repositórios autônomos que armazenam descrições de recursos, incluindo os dados capturados pelos recursos, nesse caso sensores.

O QoDisco provê uma API para realizar tarefas de descoberta em tais repositórios, possibilitando consultas com diversos atributos, e operações síncronas e assíncronas. Além disso, o QoDisco envolve um modelo de informação para a descrição de recursos baseado em ontologias com um vocabulário de conceitos relacionados a recursos e informações de contexto, incluindo o relacionamento entre eles. Esse modelo de informações utiliza ontologias para a descrição semântica de recursos e seus serviços. Como objetivo secundário, o QoDisco foi concebido para ser integrado ao middleware EcoDiF.

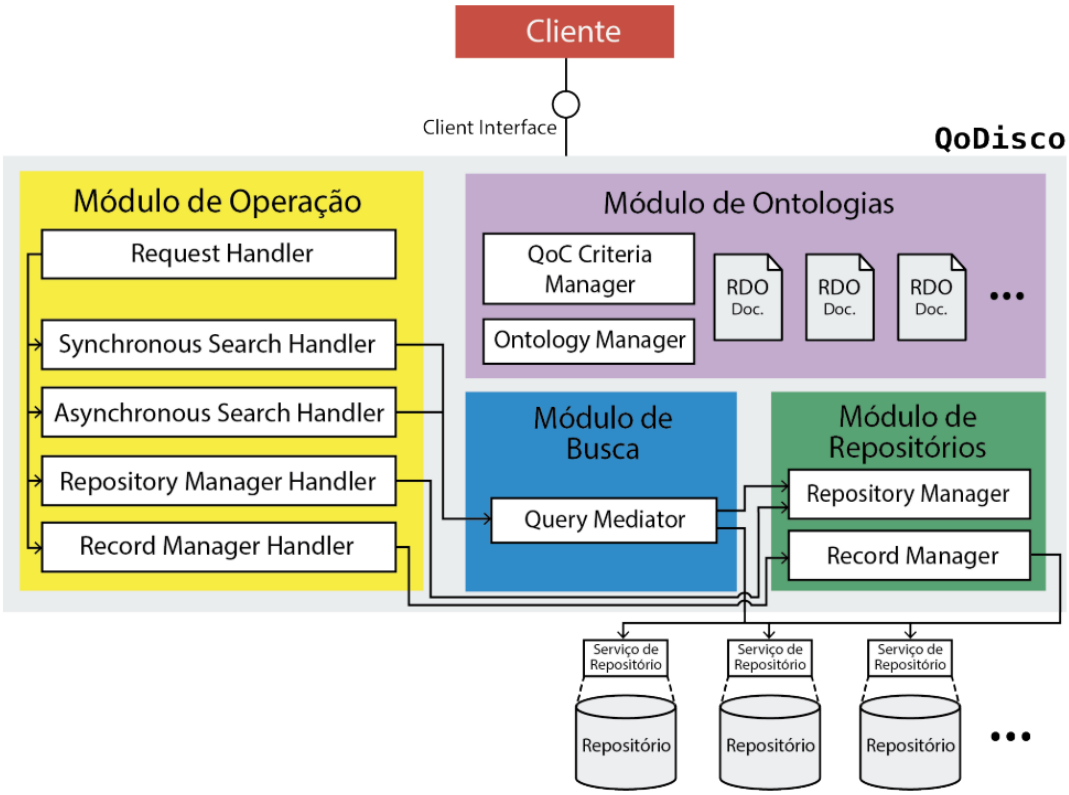


Figura 6 – Arquitetura do QoDisco. Fonte: (GOMES, 2016)

A arquitetura do QoDisco compreende quatro módulos, cada um responsável por um conjunto de funcionalidades específicas providas pela plataforma. A Figura 6 ilustra esses módulos os quais podem ser descritos como a seguir:

- Módulo de Ontologias: este módulo consiste de documentos *Repository Domain Ontology* (RDO), que descrevem os conceitos que compõem o modelo de informação do QoDisco;
- Módulo de Repositórios: a principal funcionalidade do Módulo de Repositórios é gerenciar repositórios, cada um tendo suas funcionalidades expostas por uma API, a qual é chamada de Serviço de Repositório. Nesse módulo, o componente *Repository Manager* é responsável por gerenciar repositórios e mapeá-los para um (ou mais) documentos RDO presentes no Módulo de Ontologias;

- Módulo de Busca: este compreende o componente Query Mediator, que encaminha consultas para os repositórios registrados no QoDisco. Para realizar a busca, esse componente fornece o nome de um documento RDO (nome de domínio) ao componente *Repository Manager*, que provê a URL dos repositórios mapeados para tal domínio;

- Módulo de Operações: o Módulo de Operações provê alguns serviços de alto-nível, tais como a descoberta síncrona e assíncrona de registros, e é composto por cinco componentes os quais tratam requisições de busca, adição, edição e remoção de registros na arquitetura.

Um estudo de caso foi realizado considerando um cenário relacionado ao monitoramento de níveis de poluição de ar. Tal análise foi motivada por dois objetivos: (i) verificar o quanto a quantidade de recursos registrados em um repositório impacta no desempenho do procedimento de busca; e (ii) verificar como o número de repositórios influencia o comportamento do procedimento de busca.

### 3.1.3 A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things

Neste trabalho (CIRANI et al., 2014), é proposto uma arquitetura P2P escalável e autoconfigurável para redes da IoT de larga escala, com o objetivo de fornecer serviços automatizados e mecanismos de descoberta de recursos que não requerem intervenção humana para sua configuração.

Mais especificamente, este trabalho tem foco na descoberta de serviços locais e globais, mostrando como a arquitetura proposta permite que estes mecanismos interajam com sucesso, mantendo sua independência mútua, do ponto de vista operacional.

A arquitetura de descoberta proposta neste trabalho depende da presença de um "IoT Gateway" (Figura 7). As informações sobre os recursos fornecidos por objetos inteligentes conectados a uma rede sem fio local são reunidas por este componente "IoT Gateway", o qual também faz parte de uma sobreposição de P2P usada para armazenar e recuperar essas informações, resultando em uma descoberta de recursos distribuída e escalável (Figura 8).

Combinando diferentes funções, o IoT Gateway fornece tanto nós de IoT quanto nós padrão (não restritos) com descoberta de recursos e serviços e (opcionalmente) funcionalidades de armazenamento em cache e de controle de acesso. O IoT Gateway interage, no nível de aplicação, com outros nós por meio do *Constrained Application Protocol* (CoAP) e pode atuar como cliente e servidor CoAP. Além do comportamento padrão de proxy do CoAP, o IoT Gateway também pode atuar como um proxy HTTP-to-CoAP traduzindo solicitações HTTP para solicitações CoAP, e vice-versa. O IoT

Gateway é, portanto, um elemento de rede que coordena e permite a interoperabilidade entre dispositivos altamente heterogêneos.

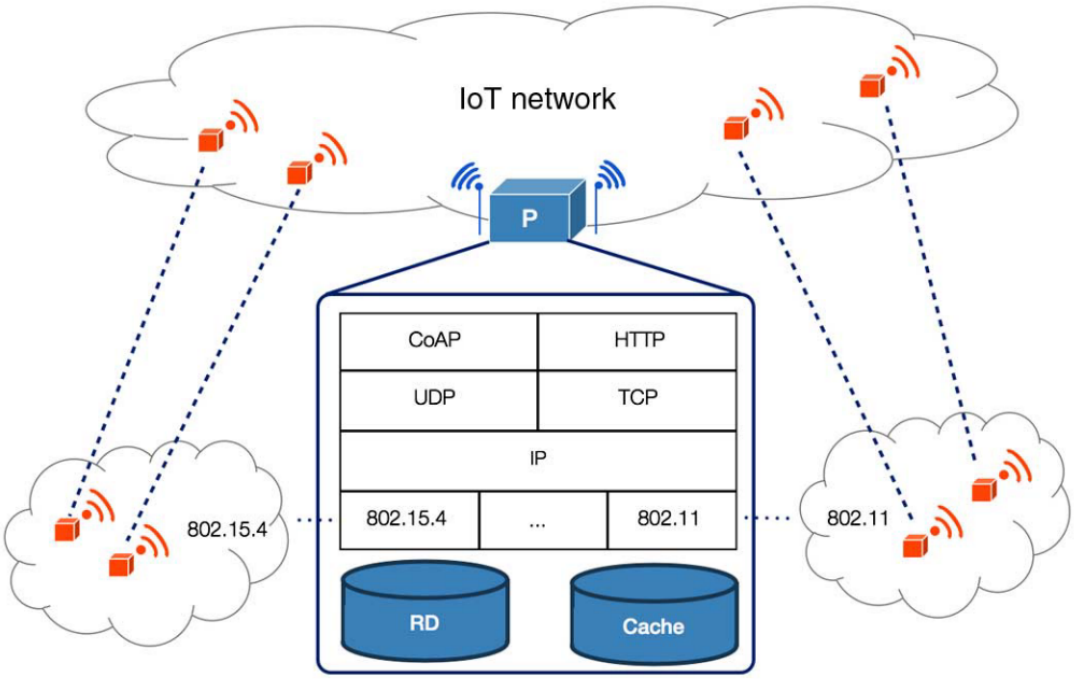


Figura 7 – Arquitetura do IoT Gateway com camadas internas e recursos de armazenamento em cache/RD. Fonte: (CIRANI et al., 2014)

Os IoT Gateways podem ser organizados em uma sobreposição de P2P para fornecer um mecanismo SD em larga escala como pode ser observado na Figura 8. Estes são organizados como *peers* de uma sobreposição P2P estruturada, que fornece uma resolução de nome eficiente para os serviços CoAP. A arquitetura SD de larga escala apresentada neste trabalho baseia-se em duas sobreposições P2P:

- o *Distributed Location Service* (DLS): fornece um serviço de resolução de nomes para recuperar todas as informações necessárias para acessar um recurso, de qualquer tipo, identificado por uma URI;
- a *Distributed Geographic Table* (DGT): constrói um conhecimento geográfico distribuído, baseado na localização dos nós, que pode ser usado para recuperar uma lista de recursos que correspondem aos critérios geográficos.

A combinação desses dois sistemas de sobreposição P2P permite construir uma arquitetura distribuída para a descoberta de serviços em grande escala com os recursos típicos de redes P2P (escalabilidade, robustez e autoconfiguração), permitindo ainda os recursos exclusivos da descoberta de recursos e serviços numa base geográfica.

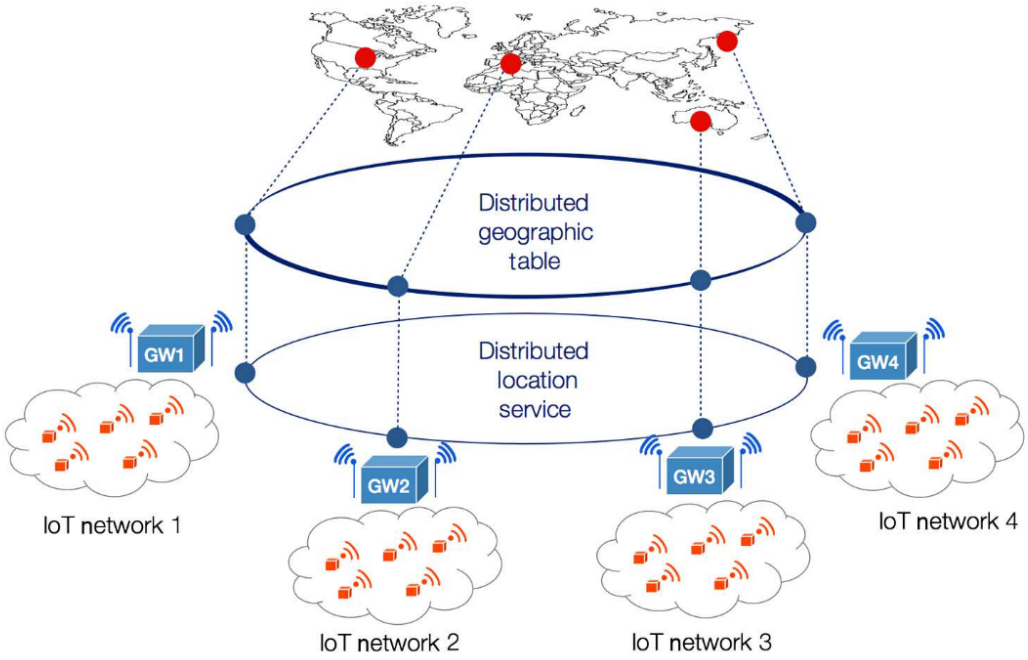


Figura 8 – Arquitetura SD em grande escala. Fonte: (CIRANI et al., 2014)

A fim de validar a viabilidade da solução proposta e avaliar seu desempenho, uma experimentação foi realizada visando o cenário de vigilância de infraestrutura inteligente. A avaliação de desempenho se concentra nos mecanismos de descoberta de serviços locais e de grande escala.

## 3.2 Arquiteturas Centralizadas para Descoberta de Recursos

A seguir serão apresentados trabalhos cujas arquiteturas são definidas como centralizadas. Este tipo de arquitetura se baseia na centralização das informações, facilitando o controle e o acesso sobre as mesmas. Estes, da mesma forma, serão descritos com relação a estrutura de dados utilizada, modo de comunicação, busca e cenário de uso.

### 3.2.1 Mobile Digcovery: discovering and interacting with the world through the Internet of things

O trabalho (JARA et al., 2013) apresenta a arquitetura móvel do Digcovery que permite que os consumidores monitorem e controlem seus produtos a partir de navegadores e smartphones. A proposta apresenta um método de como integrar recursos heterogêneos disponíveis na IoT, que possam ser controlados através de plataformas mobile. Estas plataformas recebem as informações sobre os recursos disponíveis a partir de mecanismos de busca escalonáveis, como por exemplo o Elasticsearch.

A Figura 9 apresenta o papel dos componentes da arquitetura do Digcovery. O

componente *Service Layer* apresenta a camada de serviço, cuja finalidade é principalmente construir as interfaces de comunicação com os aplicativos e usuários por meio de serviços Web RESTFul.

Para o uso da arquitetura de descoberta do Digcovery através de plataformas móveis, é oferecida para os usuários o aplicativo Mobile Digcovery, promovendo uma maior interação com diferentes perfis de usuários.

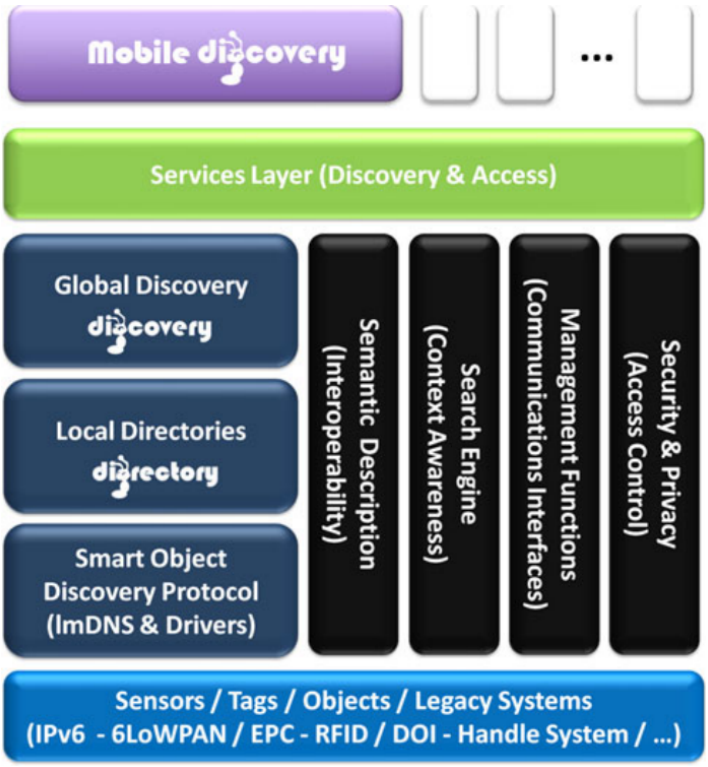


Figura 9 – Arquitetura do Digcovery. Fonte: (JARA et al., 2013)

Os principais componentes do Digcovery para fornecer um ambiente homogêneo e inter operável para descobrir, pesquisar e registrar serviços e recursos são:

- *Global Discovery*: o principal elemento da arquitetura, o qual se responsabiliza pela descoberta global. Esse componente é usado para localizar recursos nos diferentes diretórios;
- *Local Directories*: contêm descrições de recursos e serviços de cada um dos domínios. Esses diretórios não são dependentes de tecnologia, portanto, isso pode ser conectado a qualquer outra plataforma por meio de um driver;
- *Smart Object Discovery Protocol*: é baseado no Multicast DNS (mDNS) e na semântica da Descoberta de Serviço DNS para descrever serviços e recursos sobre o DNS;
- *Semantic Description*: utilizar para descrever semanticamente os recursos;

- *Search Engine*: o motor de busca utilizado neste caso foi o *ElasticSearch*. A escolha deste permite a pesquisa de reconhecimento de contexto para o Digcovery mobile, dando suporte a localização geográfica, identificação, perfis de aplicativos e suporte para vários domínios;

- *Management Functions*: refere-se às funções de gerenciamento e interfaces de comunicação necessárias para interoperar com plataformas e soluções de terceiros. Para isso é utilizado o CoAP o qual é considerado.

Por fim, o trabalho utiliza como estudo de caso uma cadeia de valor de produtos finais para os sistemas *backend* de um fabricante, e o respectivo ecossistema global que o envolve.

### 3.2.2 CADsIoT: Designing a Context-Aware Discovery Service for IoT Devices

Nesta trabalho (KHAN, 2017), foi criado um serviço que permite a descoberta automática de vários dispositivos na IoT com base em um contexto particular. O conceito proposto é composto principalmente de sensores, o que inclui identificadores exclusivos, fabricante e status atual.

O CADsIoT é composto por dois componentes principais: um serviço de *backend* denominado CADsIoT-Core e como cliente um navegador de smartphone. O serviço de *backend* atua como o repositório centralizado de dispositivos, sensores, informações contextuais e outros ativos associados. O Navigator se comunica com o serviço de *backend* por meio de APIs RESTful. Ambos os componentes são discutidos nesta seção.

O CADsIoT-Core possui uma arquitetura multicamadas, construída em serviços Web RESTful. Ele inclui um princípio de acoplamento flexível que fornece benefícios da mesma forma que a maioria das aplicações dinâmicas da web. A implementação baseada em REST disponibiliza operações para gerenciar ativos da IoT, por exemplo, dispositivos, sensores, contexto e assinaturas. Este padrão de comunicação facilita a criação, modificação e gerenciamento de ativos. A Figura 10 retrata cinco camadas de arquitetura, descrevendo vários componentes e serviços em cada camada suportada pelo CADsIoT-Core. Cada camada e seus componentes são definidos da seguinte forma:

- *Presentation Layer*: camada responsável por expor informações de dispositivos, sensores e contextos podendo ser gerenciadas por meio de solicitações HTTP REST. As solicitações independentes usam métodos HTTP, por exemplo, POST, GET, PUT e DELETE, que são mapeados para as operações correspondentes dos recursos e, em seguida, as solicitações são delegadas à Service Layer;

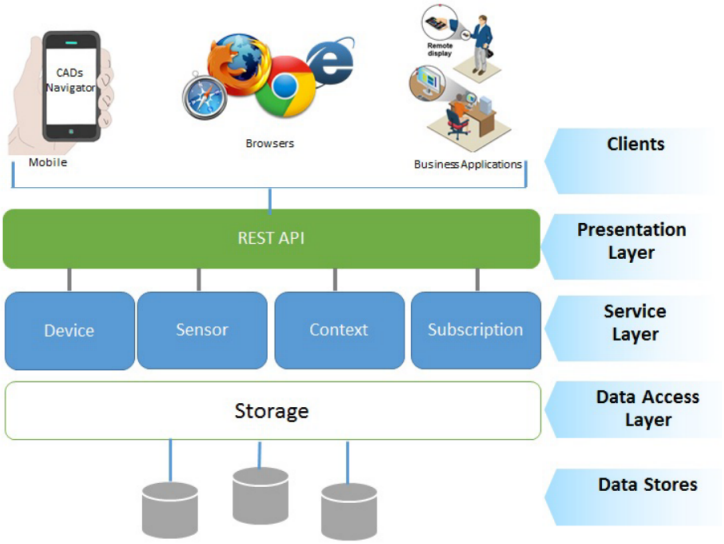


Figura 10 – Arquitetura do CADsIoT-Core. Fonte: (KHAN, 2017)

- *Service Layer*: define as regras de negócios para cada tipo de serviço e atua como um mediador entre a camada de apresentação e a camada de persistência para troca de dados. Esta camada é composta por quatro serviços básicos, disponíveis no componente CADsIoT Core, dispositivo, sensor, contexto e assinatura. As solicitações enviadas por meio de interfaces uniformes (HTTP REST) da camada de apresentação são mapeadas para serviços correspondentes, executam regras de negócios (lógica de negócios) e enviam a funcionalidade necessária para o solicitante;

- *Data Access Layer*: é uma camada de abstração que fornece uma interface para acessar dados de um ou mais armazenamentos de dados. A camada é responsável pelo preenchimento de solicitações relacionadas a dados, solicitadas por serviços da camada de serviços;

- *Data Stores*: geralmente consiste em armazenamentos de persistência, servidores de diretório, sistemas legados e corporativos. Os dados de IoT são normalmente distribuídos em vários mecanismos de armazenamento de dados, por exemplo, BLOB (Binary Large Object) para dados não estruturados, armazenamentos de valores-chave e bancos de dados relacionais. Ao combinar, uma visão consolidada e integrada dos dados é apresentada ao pedido.

O segundo componente do CADsIoT, o CADsIoT-Navigator é um protótipo baseado em Android que é capaz de descobrir dispositivos e sensores automaticamente implantados em um ambiente particular, independentemente de sua heterogeneidade. O navegador interage com o CADsIoT-Core para receber dispositivos registrados, dados contextuais e informações de assinatura para fornecer notificações em tempo real

que informam onde os dispositivos estão instalados. A Figura 11 mostra a arquitetura de alto nível do Navigator, delineando vários blocos de construção os quais são:

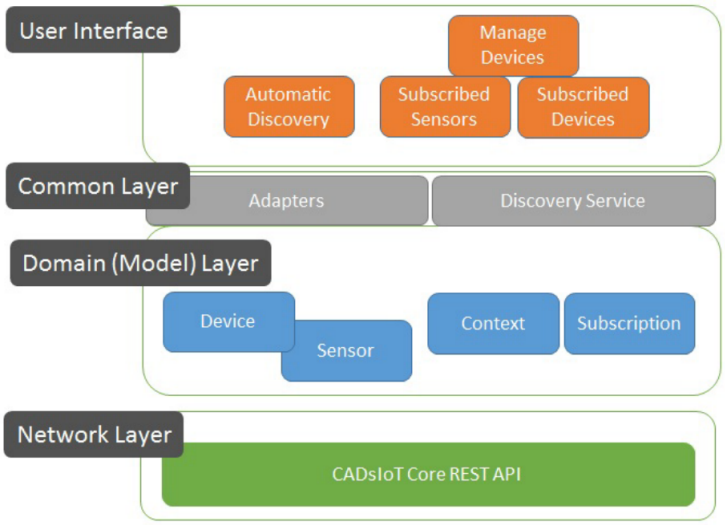


Figura 11 – Arquitetura do CADsIoT-cliente. Fonte: (KHAN, 2017)

- *User Interface*: também chamada de camada de apresentação fornece uma experiência do usuário para interagir com diferentes serviços fornecidos pelo aplicativo. Essa camada consiste em vários componentes que são nativos da plataforma Android e servem como blocos de construção. Esses componentes incluem atividades, fragmentos, layouts, fornecendo uma interface de usuário rica;
- *Commom Layer*: lida com vários componentes do Android, por exemplo, drivers de banco de dados e serviços do Android. O Navigator fornece um serviço de descoberta que reside entre a interface do usuário e a camada de domínio com o objetivo de descobrir dispositivos automaticamente. O serviço usa Serviços Baseados em Localização proativos (LBS), mais conhecido como Geofencing, que permite o monitoramento remoto de objetos. Este notifica o usuário sobre as informações baseadas na localização fornecida pelo GPS, atualizando a base de dados no CADsIoT-Core;
- *Domain Model Layer*: é uma camada de domínio que interage com a camada de rede para solicitar dados e incluir lógica de negócios e modelo de dados. Os respectivos serviços nessa camada correspondem ao CADsIoT-Core (serviço de backend) e solicitam os dados necessários. Quando as informações necessárias são recuperadas, os objetos são enviados para as camadas superiores;
- *Network Layer*: realiza tarefas relacionadas à rede exigidas pelo aplicativo. Na arquitetura do Navegador, essa camada é responsável pela comunicação com o serviço de backend para executar operações HTTP. Ele também gerencia o agendamento de solicitações e fornece recursos de armazenamento em cache.

Este trabalho descreve um cenário de alto nível utilizado pelo trabalho em que um usuário está passando por um prédio específico e interessado em conhecer os sensores disponíveis no prédio empregando um dispositivo móvel. É importante mencionar que o edifício é ainda classificado em andares e quartos. Ao atingir a proximidade dos dispositivos e sensores instalados, o usuário é notificado em seu smartphone sobre a descoberta de dispositivos disponíveis. Além disso, sempre que o usuário passa do mesmo edifício, uma notificação automática é gerada no dispositivo móvel que identifica o dispositivo e seus detalhes.

### 3.2.3 Descoberta Contínua de Serviços em IoT

O trabalho (CARVALHO, 2017) tem como objetivo definir uma arquitetura para descoberta contínua de serviços na IoT de forma a contribuir com o *middleware* M-Hub/CDDL promovendo uma extensão de seus serviços.

Objetivamente, as contribuições se dão através de duas abordagens: (i) uma arquitetura assíncrona para descoberta de serviços na IoT; e (ii) uma API para descoberta contínua de serviços em IoT, que leva em consideração características de QoC expressas através de consultas *Complex Event Processing* (CEP).

CEP é uma tecnologia para processamento dinâmico de fluxos de dados de eventos em quase tempo-real. 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.

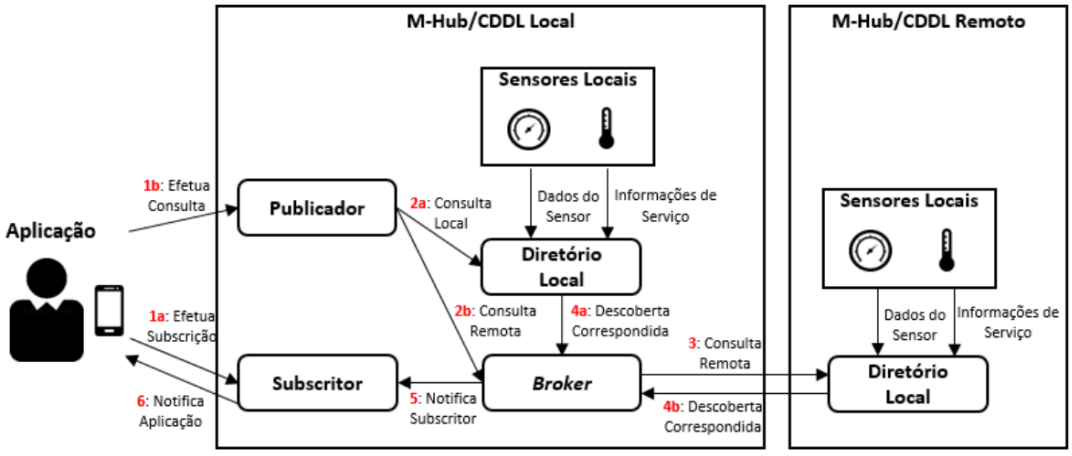


Figura 12 – Fluxo de execução do mecanismo de descoberta contínuo em nível local e em nível global com publicação remota da consulta. Fonte: (CARVALHO, 2017)

A ideia base da arquitetura de descoberta é combinar a detecção em fluxos de eventos proporcionada pelo CEP, e um mecanismo de comunicação *Publish and Subscribe* com base em tópicos. Nessa arquitetura (Figura 12), 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 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.

## 3.3 Análise Comparativa dos Trabalhos

Dentre os trabalhos relacionados discutidos, aqueles que exploram arquiteturas centralizadas, apresentam como característica a situação de que com o crescimento do número de recursos envolvidos, os procedimentos de descoberta tendem a sobrecarregar o diretório central em decorrência do crescimento das demandas de acesso e de processamento. Estes trabalhos estão descritos na Seção 3.2, e tem como contraponto positivo a facilidade de manutenção do estado dos recursos, e a decorrente facilidade de disponibilização dos mesmos para as aplicações clientes.

Neste sentido, este trabalho foi concebido explorando diretórios distribuídos, organizados segundo contextos de interesse das aplicações.

O uso de arquiteturas distribuídas, vide Seção 3.1, em um cenário de elevada escalabilidade de recursos como o da IoT, se mostra necessário para a distribuição do processamento e do acesso as informações dos recursos. A revisão de literatura indica que o emprego de arquiteturas distribuídas para descoberta de recursos, vem crescendo a medida que o uso da IoT se expande. Contudo este tipo de proposta arquitetural requer atenção no que diz respeito aos aspectos de propagação das informações associadas aos procedimentos de busca, sobretudo em função de quesitos de segurança, uma vez que operações distribuídas envolvendo recursos de diferentes naturezas, dificultam a homogenização de procedimentos de segurança, tornando a infraestrutura mais suscetível a ataques e a capturas indevidas de dados.

A análise dos trabalhos relacionados indicou que o emprego de protocolos consolidados como CoAP e MQTT vem se mostrando importante para diferentes grupos de pesquisa, pois dentre outros aspectos, facilita a manutenção e o suporte a sistemas legados. Neste sentido, este trabalho teve como premissa de concepção facultar o uso por parte das aplicações do protocolo que lhe for mais oportuno dentro de um conjunto de opções disponibilizado.

As estruturas de dados para armazenamento dos metadados associados aos recursos estão atreladas são definidas prioritariamente pelo método de busca empregado.

Em arquiteturas distribuídas com base em redes P2P, o uso de tabelas hash distribuídas são a escolha mais usual. As diferenças ficam por conta dos metadados utilizados, os quais podem envolver diferentes tipos de informações referentes aos recursos, inclusive localizações georreferenciadas. Por sua vez, os trabalhos que empregam arquiteturas de busca centralizadas divergem quanto a estrutura empregada para armazenamento dos dados, a maioria empregando soluções proprietárias.

No caso do acesso as funcionalidades de descoberta, o uso de APIs é a abordagem mais usual, disponibilizando interfaces REST ou *Publish/Subscribe* para gerência da troca de informações.

Dos trabalhos discutidos, todos conseguiram atingir seus objetivos gerais, dando suporte a descoberta e busca por recursos em um ambiente de alta escalabilidade e heterogeneidade. Entretanto cada um validou sua proposta empregando aplicações direcionadas a cenários específicos na infraestrutura computacional da IoT. A exceção do trabalho (JARA et al., 2013), que de forma análoga a esta dissertação, também propõe uma abordagem multiaplicação.

Fazendo um comparativo entre o EXEHDA-HS e os trabalhos descritos, este trabalho apresenta o melhor dos mundos para a execução na arquitetura do EXEHDA. Enquanto há uma distribuição das informações e das capacidades de descoberta, há uma centralização contextual das informações, mantendo próximas as células mais relevantes para cada contexto. Este trabalho utilizou técnicas específicas para a busca de recursos mas em compensação levou em consideração buscas multiatributos, o necessário para atender as demandas de localização de recursos. O cenário de uso empregado foi direcionado a agricultura, mas atendendo as demandas da IoT no geral.

Uma comparação dos trabalhos relacionados pode ser observada nas Tabelas 1 e 2, nesta comparação foram elencados aspectos entendidos como centrais para concepção da proposta desenvolvida nesta dissertação.

Tabela 1 – Comparação entre os trabalhos relacionados.

| Trabalho | Arquitetura | Estrutura | Comunicação | |
|---|---|---|---|---|
| | | | Metodo | Protocolo |
| (JARA et al., 2013) | Centralizada | Própria | REST | Coap |
| (KHAN, 2017) | Centralizada | Própria | REST | HTTP |
| (CARVALHO, 2017) | Centralizada | Própria | Publish/Subscribe | MQTT |
| (PAGANELLI; PARLANTI, 2012) | Descentralizada | PHT | P2P | Não informado |
| (GOMES, 2016) | Descentralizada | Própria | HTTP RestFul | HTTP |
| (CIRANI et al., 2014) | Descentralizada | DHT | P2P/REST | Zeroconf, DHCP, COAP |

Tabela 2 – Comparação entre os trabalhos relacionados (continuação)

| Trabalho | Busca | | API | Cenário de uso |
|---|---|---|---|---|
| | Metodo | Multiatributos | | |
| (JARA et al., 2013) | Ellastic Search | Sim | Sim | Multiaplicação |
| (KHAN, 2017) | Não informado | | Sim | Cidades Inteligentes |
| (CARVALHO, 2017) | CEP | Sim | Sim | Estacionamentos inteligentes |
| (PAGANELLI; PARLANTI, 2012) | Sistemática | Sim | Sim | Cadeia de transportes |
| (GOMES, 2016) | SPARQL | Sim | Sim | Poluição de ar em cidades |
| (CIRANI et al., 2014) | Sistemática | Não informado | Sim | Vigilância de infraestrutura |

## 3.4 Considerações Finais do Capítulo

Este Capítulo apresentou seis trabalhos relacionados ao EXEHDA-HS. Estes trabalhos foram divididos considerando seu tipo de arquitetura, centralizadas e descentralizadas. Ao final foi feito um esforço de comparação levando em consideração sua arquitetura, estrutura de dados, método e procolo de comunicação, cenários de uso e método de busca.

O estudo detalhado destes trabalhos foi central para os esforços de concepção do EXEHDA-HS, cuja arquitetura e funcionalidades serão discutidas na próxima sessão.

# 4 EXEHDA-HS: VISÃO GERAL E CONCEPÇÃO

Este capítulo apresenta a concepção do serviço de descoberta de recursos EXEHDA-HS, destinado a integrar o *middleware* EXEHDA. A concepção do EXEHDA-HS foi feita considerando os avanços já obtidos nas pesquisas realizadas em (DILLI, 2010) (AZEVEDO, 2017). Estes dois trabalhos exploraram abordagens semânticas tanto para descrição dos recursos, como para promover inferências quando da realização das buscas.

De modo mais específico, o EXEHDA-HS teve seus componentes organizados buscando um aumento na ubiquidade do serviço de descoberta, provendo uma abordagem escalável e com operação transparente para a descoberta de recursos no ambiente computacional provido pela IoT.

Com o objetivo de associar as funcionalidades disponibilizadas pelo EXEHDA-HS às diferentes tecnologias de software que lhe dão suporte, o texto desta seção está organizado de modo que as citações às mesmas aconteçam no momento em que a caracterização do seu emprego é necessária para entendimento de como ocorre a operação do EXEHDA-HS.

## 4.1 Premissas de Concepção

A premissa central de concepção foi contribuir com o Subsistema de Execução Distribuída do EXEHDA, de forma a qualificar o *middleware* às demandas inerentes às modernas infraestruturas computacionais da IoT, promovendo neste cenário suporte a descoberta de recursos geograficamente dispersos. O EXEHDA-HS prevê o emprego de uma arquitetura estruturada, híbrida e hierárquica, a ser incorporada a organização celular do EXEHDA.

Como as células de execução do *middleware* são independentes, o EXEHDA-HS emprega Tabelas Hash Distribuídas (DHT), com a premissa de organizar o ambiente celular mantendo os recursos descentralizados e, ao mesmo tempo, criando uma hierarquia com o objetivo de otimizar a busca por recursos distribuídos. De maneira mais específica, as células e seus recursos serão organizados em uma estrutura de árvore,

onde a célula com hierarquia superior (raiz) possuirá as referências e descrições de suas células associadas.

Dessa forma, o ambiente celular do *middleware* fica mapeado de forma hierárquica a partir de critérios contextuais, por exemplo células de uma mesma organização, de uma mesma região geográfica ou, colocando de forma mais genérica, todas as células que compartilhem recursos com um mesmo perfil de interesse. Cada uma das possíveis hierarquias no ambiente celular, explora o conceito de uma célula mãe a qual mantém um diretório com as descrições de todas as células presentes na hierarquia sobre sua gerência.

Um outro conceito importante na concepção do EXEHDA-HS é o de célula vizinha. Para uma célula ser considerada vizinha de uma outra, ela já deve previamente ter disponibilizado um recurso buscado por uma Aplicação Cliente localizada na célula de origem. Por padrão, no EXEHDA-HS o número de células vizinhas está limitado a 10, podendo ser modificado pelo administrador do *middleware*.

Na perspectiva do EXEHDA-HS, um recurso pode estar em quatro possíveis estados: (i) ocioso; (ii) ativo; (iii) inativo; ou (iv) inoperante. Considerando estes quatro possíveis estados, os diferentes recursos têm sua utilização considerada pelas aplicações.

**Desafios de Pesquisa**

Diferentes desafios de pesquisa estão associados a área de descoberta de recursos distribuídos. No que diz respeito ao desenvolvimento do EXEHDA-HS, destacam-se os aspectos relacionados a seguir: (i) preservar a organização celular do *middleware* EXEHDA; (ii) criar uma sobreposição de roteamento baseada em contexto a qual facilita a disposição e o acesso aos recursos do middleware EXEHDA; e (iii) prover métodos de busca de recursos considerando o uso de multiatributos.

Além dos desafios ligados ao desenvolvimento do EXEHDA-HS e a integração ao *middleware* EXEHDA, os que dizem respeito as demandas específicas de sistemas de descoberta de recursos podem-se destacar:

- *Bootstrapping* ou procedimento de inicialização e configuração de recursos;
- Segurança e privacidade do tráfego de informações;
- Classificação de resultados de busca;
- Controle da partida e da falha de nós;
- Gerenciamento eficiente de recursos;
- Representação de dados.

Este trabalho procurou de alguma forma contemplar estes desafios e criar mecanismos que atendam as demandas introduzidas pela descoberta de recursos direcionadas à IoT.

**Funcionalidades Contempladas**

As funcionalidades disponibilizadas pelo EXEHDA-HS, podem ser acessadas por meio de uma API e organizadas em três grupos: (i) descoberta síncrona de recursos; (ii) descoberta assíncrona de recursos; e (iii) gerenciamento de recursos e repositórios. Estas funcionalidades do EXEHDA-HS estão descritas a seguir.

*Descoberta síncrona*

Na descoberta síncrona de recursos, os mecanismos utilizados pelo EXEHDA-HS se responsabilizam por criar eventos ligados a entrada e a saída de dispositivos da rede local da célula de execução.

Esta descoberta no EXEHDA-HS é realizada em intervalos temporais definidos. Uma vez descoberto um recurso, um evento associado é disparado, permitindo que um procedimento de avaliação e inclusão do recurso aconteça. Para que um dispositivo seja reconhecido pela célula, este deve aguardar a verificação do gerente da célula, só então poderá ser utilizado pelas aplicações.

A descoberta síncrona também coopera com os esforços de configuração automática. Caso o dispositivo descoberto possua descrições a respeito do seu modo de operação, este poderá ser configurado automaticamente e disponibilizado para uso assim que reconhecido.

*Descoberta assíncrona*

O processo de descoberta assíncrona tem o objetivo de buscar recursos distribuídos no ambiente celular do EXEHDA. Estes recursos podem estar presentes em três locais distintos: (i) na mesma célula de onde a busca foi originada; (ii) em células vizinhas; ou (iii) nas outras células do ambiente celular, ou seja, células que podem conter recursos de interesse.

A busca é realizada utilizando os algoritmos de roteamento do Kademlia, vide 2.3.4.5. O Kademlia cria uma sobreposição a qual usa o protocolo UDP (*User Datagram Protocol*) para comunicar-se com os nodos da rede P2P efetuando buscas na tabela de roteamento de cada nodo. No momento que o recurso é encontrado, o mesmo responde para a origem possibilitando a troca de informações.

A descoberta assíncrona possui um limite de operação, o qual pode ser definido por três parâmetros: (i) utilizando um tempo limite para a realização da busca (TTL); (ii) por meio de um número máximo de repasses da busca para outras células (saltos); e (iii) um limite espacial considerando coordenadas geográficas da célula.

*Gerenciamento de recursos*

As funcionalidades de gerenciamento de recursos dizem respeito aos métodos de inclusão, edição e remoção de recursos do Repositório de Informações Descobertas. No EXEHDA-HS, empregando um conjunto de métodos disponibilizados por meio de uma API RESTFul, é possível gerenciar todas informações referentes aos recursos disponíveis no RID.

Além das tarefas usuais de edição de informações contextuais dos recursos, outra tarefa de gerência disponibilizada pelo gerenciamento de recursos é a verificação e inclusão de dispositivos descobertos. Estas tarefas efetuam buscas síncronas e assíncronas na lista de dispositivos reconhecidos pelo ambiente celular. Para isso é necessário que algum usuário avalie as informações descobertas e indique se este pode ser utilizado e incluído na lista de dispositivos reconhecidos.

## 4.2 Arquitetura Proposta

Considerando as premissas de concepção discutidas na Seção 4.1, a arquitetura do EXEHDA-HS foi desenvolvida com o objetivo de possibilitar que os recursos presentes na célula, provida pelo EXEHDA, possam ser descobertos considerando os protocolos tipicamente empregados na infraestrutura computacional da IoT.

A arquitetura de software para descoberta utilizada pelo EXEHDA-HS pode ser observada na Figura 13, onde cada célula possui um Subsistema de Execução Distribuída, o qual é formado por um conjunto de componentes os quais tem como objetivo buscar recursos distribuídos e promover sua gerência.

A caracterização das funcionalidades e modos de operação destes componentes estão discutidos na Seção 4.4.

A seguir uma discussão da contribuição dos componentes que integram a arquitetura do EXEHDA-HS, com uma caraterização das suas funcionalidades:

***Repositório de Informações Descobertas***

O Repositório de Informações da Descoberta (RID) é responsável por armazenar informações referentes aos recursos e as aplicações clientes existentes na célula. Estas informações consistem de metadados, subscrições, contextos e cache de informações sobre recursos. Este repositório no EXEHDA-HS consiste em um banco de dados estruturado disponibilizado no âmbito de uma célula de execução.

Através de uma API RESTFul, o EXEHDA-HS disponibiliza métodos de inclusão, edição e remoção de dados do RID a nível local. Tal API possibilita que aplicações clientes possam fazer uso das informações armazenadas durante todo o processo de execução de descoberta de recursos.

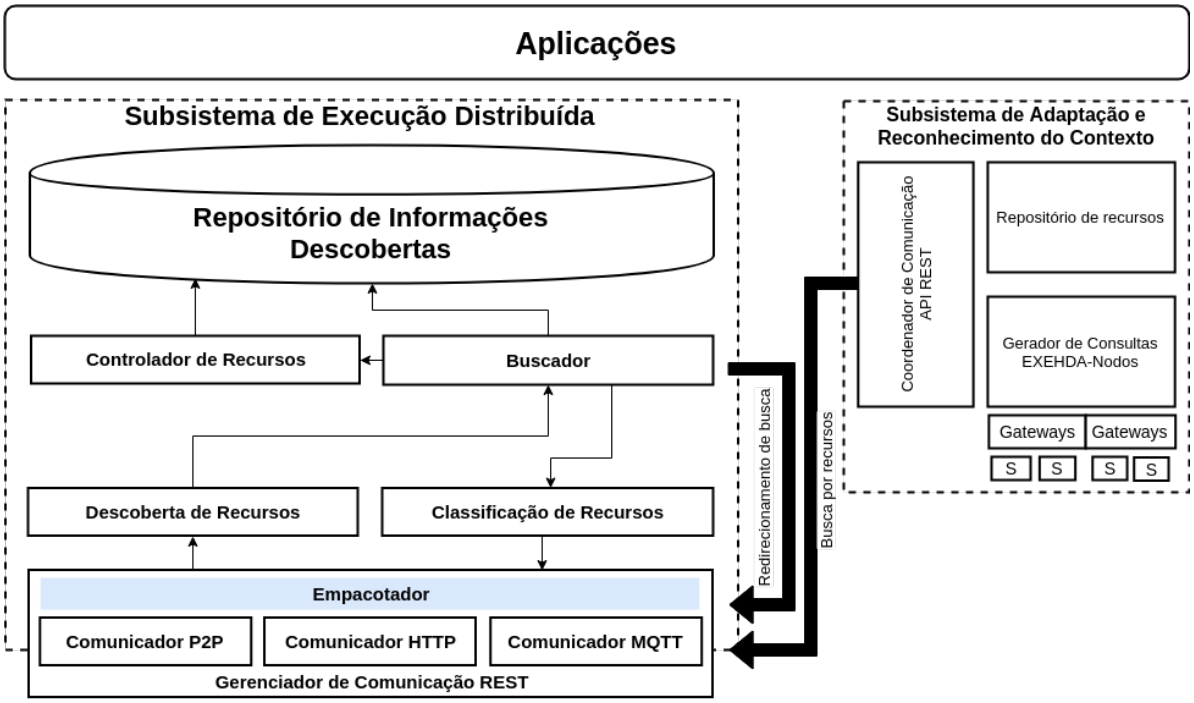


Figura 13 – Visão geral da arquitetura do EXEHDA-HS.

### Controlador de Recursos

O Controlador de Recursos atua sobre aqueles recursos já reconhecidos pela célula, efetuando verificações periódicas de tempo pré-definidos com o objetivo de verificar o estado de operação de um determinado recurso e atualizar as informações de estado presentes no RID. Desse modo, em conjunto com o RID, o componente Controlador de Recursos efetua a tarefa de controlar o estado de cada recurso, tendo como propósito evitar que haja recursos perdidos na cadeia de busca em função de estados inconsistentes. Sempre que um recurso trocar de estado, o qual pode ser ocioso, ativo, inativo ou inoperante, o Controlador de Recursos se responsabiliza por avisar os nodos com com interesse nestes.

### Buscador

O componente Buscador tem como objetivo realizar as buscas assíncronas solicitadas pela Aplicação Cliente. Tais buscas podem acontecer tanto no RID da célula local, ou então, redirecionando para os nodos vizinhos conhecidos. Esta busca ocorre fazendo uso da sobreposição Kademlia a qual, para cada nodo que atinge, verifica a tabela de roteamento, e por consequência seu RID, em busca do recurso solicitado.

Além de buscar os recursos, o Buscador também informa ao componente Controlador de Recursos de seu nodo o estado atual dos recursos que estão sendo buscados.

### Descoberta de Recursos

A função do componente Descoberta de Recursos é reconhecer qualquer recurso que transita pela rede local da célula de execução. De modo geral, este componente recebe todas as requisições provenientes dos outros nodos existente na célula de execução, sejam elas vindas diretamente para o Subsistema de Execução Distribuída ou aquelas requisições de trânsito (entrada/saída) de recursos da rede local.

No que diz respeito as operações na rede local das células de execução, é utilizado o serviço mDNS, vide Seção 4.4. Este serviço faz uso de mensagens *multicast* para solicitar informações descritivas a cerca dos recursos que estejam presentes na célula de execução e assim disponíveis para descoberta. Este componente também é responsável por comunicar ao Controlador de Recursos sobre o estado dos mesmos, para que este solicite a persistência destes estados no Repositório de Informações Descobertas.

### Classificador de Recursos

O componente Classificador de Recursos, de maneira geral, é um componente no qual os resultados das buscas são classificados de acordo com as necessidades de cada Aplicação Cliente. Toda vez que o componente Buscador finaliza uma requisição, este componente recebe os resultados e os classifica de acordo com os critérios escolhidos. Estes critérios podem ser relacionados a diversos aspectos de contexto como localidade, QoS, latência confiabilidade entre outros.

Embora este componente esteja presente na arquitetura, seu detalhamento não faz parte do foco desta dissertação. O trabalho em desenvolvimento (DILLI et al., 2018) faz uma abordagem do seu funcionamento.

### Gerenciador de Comunicação

Este componente é o primeiro contato das aplicações clientes com o Subsistema de Execução Distribuída. Nele são definidos rotas de entrada e saída de dados por requisições REST chamadas *endpoints*. Serviços REST do Gerenciador de Comunicação permitem que as aplicações clientes acessem e manipulem representações textuais de recursos usando um conjunto uniforme e predefinido de operações sem estado. Este componente é capacitado a receber requisições de três diferentes formas:

- Utilizando o protocolo HTTP de comunicação. Atuando por meio de requisições REST;
- Por meio de chamadas RPC utilizando a rede P2P para acessar recursos diretamente para as células.

- Subscrevendo e publicando utilizando o protocolo MQTT, vide Seção 4.4. Este possibilita que os interessados por determinada informação sejam avisados caso haja alguma modificação no estado do recurso ou da aplicação.

Além de lidar com a comunicação, este componente é também responsável por empacotar e criptografar as requisições. Este procedimento reduz o tamanho do pacote enviado junto da requisição além de impossibilitar que aplicações mal-intencionadas consigam visualizar o conteúdo dos formulários que trafegam pela rede celular.

## 4.3 Procedimento de Busca

A dinâmica dos procedimentos de busca do EXEHDA-HS está apresentados na Figura 14, sendo descritos mais especificamente com relação aos estágios de processamento. Parte dos procedimentos de busca acontecem na mesma célula requisitante, podendo a mesma expandir esta busca para células remotas.

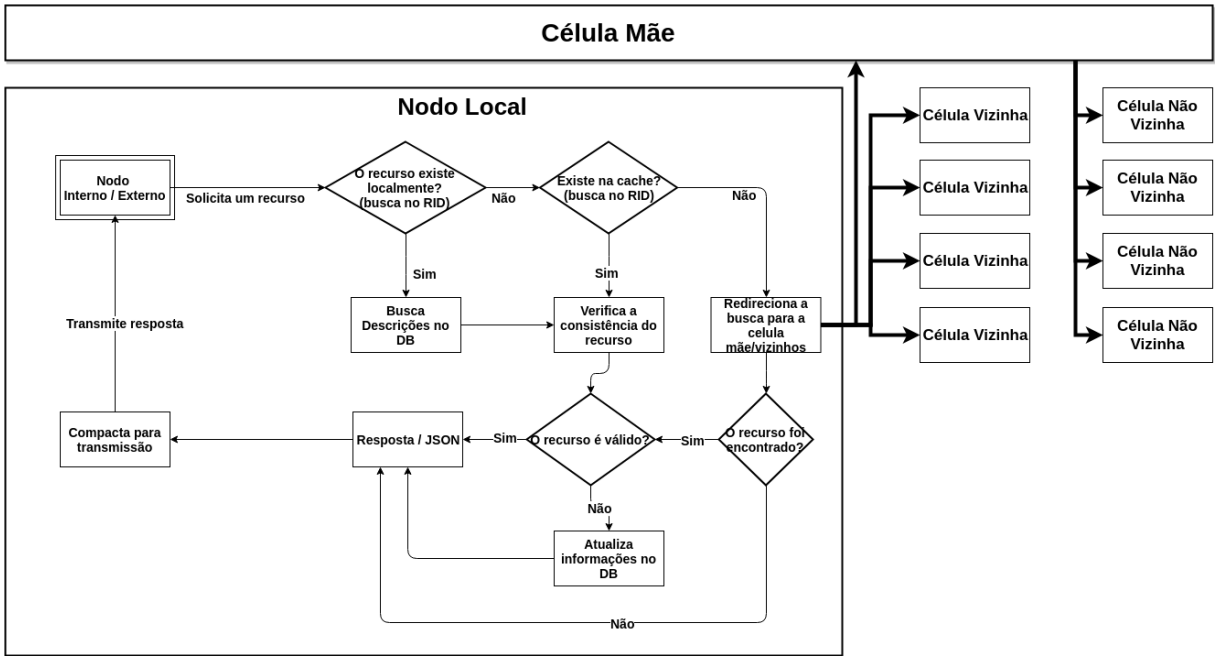


Figura 14 – Fluxo de dados do EXEHDA-HS.

A Aplicação Cliente é a primeira etapa do procedimento de busca. Nesta aplicação são decididos os critérios de busca e de classificação do recurso desejado. Entre os critérios de busca, podem ser destacados os seguintes: (i) Disponibilidade; (ii) Latência de resposta; (iii) QoS; e (iv) Distância física.

Uma vez selecionados os critérios, a aplicação envia uma requisição contendo os parâmetros de busca e os critérios de seleção para o Subsistema de Execução Distribuída de sua própria célula. Esta requisição se propaga pelos componentes

com o objetivo de encontrar os recursos apropriados, sejam eles locais, da própria célula ou de recursos públicos presentes em células remotas. Uma visualização da representação em JSON da busca propagada pelos nodos pode ser observada na Figura 15, onde é possível observar os seguintes atributos:


Figura 15 – Representação em JSON da busca por um recurso.

- Requester: este atributo especifica o requisitante da busca, possuindo sua identificação, endereço de IP, a porta de entrada e saída de dados e a url de endereçamento das respostas;
- Search: este atributo contém os parâmetros de busca, sendo atribuído uma identificação única, o tipo do recurso, se a busca deve se propagar pela rede, o tempo de duração limite da busca, suas coordenadas geográficas, o alcance da busca em quilômetros e os parâmetros de classificação.
- Results: este parâmetro consiste de uma lista de recursos já encontrados, contendo sua identificação, endereço de IP, a porta de acesso, tipo do recurso, endereço de acesso e protocolo utilizado.

O fluxo dos dados no Subsistema de Execução Distribuída começa com a busca em seu diretório local. Em um primeiro momento é verificado no RID, utilizando uma

consulta SQL (*Structured Query Language*). Caso o recurso não seja encontrado, ou seja, não exista na célula local, é feita uma busca na Cache de Respostas do RID. A Cache de Respostas tem como objetivo encurtar o caminho da busca armazenando todas as buscas que foram feitas às células remotas, viabilizando que possam ser reaproveitadas em buscas futuras.

Uma vez que o recurso não seja encontrado por busca local em uma determinada célula de execução, esta busca é encaminhada para sua célula mãe com o propósito de atingir todas as outras células com recursos de mesmo perfil contextual.

A proposta ainda prevê que se a célula em questão não possuir o recurso desejado, a busca deverá ser redistribuída para as células subsequentes. Nestas a busca acontece da mesma forma como descrito anteriormente.

Ao localizar o recurso, a próxima etapa tem como objetivo verificar a consistência das informações do recurso, ou seja, certificar se o estado assim como os metadados dos recursos estão atualizados. Este processo acontece para todo recurso encontrado localmente ou remotamente. O recurso é então acessado e então seus metadados são comparados com os armazenados localmente e, caso necessário, atualiza o repositório. Desta forma é garantido que os dados estejam sempre atualizados.

No momento que os recursos são localizados, uma lista de recursos é criada onde foi originada a busca. Esta lista, ao terminar o tempo de vida da busca, é então classificada no Classificador de Recursos utilizando os critérios de seleção definidos pela Aplicação Cliente.

A última etapa da busca é de responsabilidade do Componente Comunicador. Nele o resultado das buscas é empacotado e transmitido para as aplicações interessadas. Os protocolos utilizados são escolhidos de acordo com a solicitação, ou seja, se a requisição for proveniente do protocolo MQTT, a resposta será transmitida através do mesmo canal. Dentre os canais de comunicação, podemos citar os principais, MQTT, COAP, HTTP/REST e chamadas RPC.

## 4.4 Tecnologias Empregadas

Nesta seção estão descritas as principais tecnologias utilizadas na arquitetura do EXEHDA-HS. As tecnologias são associadas aos componentes da arquitetura, e sumarizadas as motivações para o seu emprego.

As tecnologias para emprego no EXEHDA-HS foram selecionadas a partir de um amplo espectro de opções, a luz dos seguintes critérios: (i) modernidade; (ii) sua adoção no cenário nacional e internacional; e (iii) serem open-source e por fim sua adequação ao cenário da IoT que prevê o emprego de dispositivos de pequeno porte e heterogêneos.

**Interpretador NodeJS**

A linguagem Javascript foi criada em 1995, tornando-se a linguagem padrão dos browsers e consequentemente da Web. Com a rápida evolução da Web e dos navegadores nos últimos anos, a linguagem Javascript e seus motores de execução passaram por diversas melhorias, tornando viável sua execução com outros propósitos além da manipulação de páginas HTML. Com essa nova fase no uso do Javascript, aplicações server-side passaram a ser implementadas e, em 2009 foi criado o primeiro ambiente de execução Javascript com este propósito, o Node.js (PEREIRA, 2014).

A principal característica que diferencia o Node.JS de outras tecnologias, como PHP, Java, C#, é o fato de sua execução ser sigle-thread. Ou seja, apenas uma thread é responsável por executar o código Javascript da aplicação, enquanto que nas outras linguagens a execução é multi-thread. No modelo Node.js, apenas uma thread é responsável por tratar as requisições. Essa thread é chamada de Event Loop, e leva esse nome pois cada requisição é tratada como um evento. O Event Loop fica em execução esperando novos eventos para tratar e, para cada requisição, um novo evento é criado.

Apesar de ser *single-threaded*, é possível tratar requisições concorrentes em um servidor Node.js. Enquanto o servidor tradicional utiliza o sistema *multi-thread* para tratar requisições concorrentes, o Node.js consegue o mesmo efeito através de chamadas de E/S (entrada e saída) não-bloqueantes. Isso significa que as operações de entrada e saída (acesso a banco de dados e leitura de arquivos do sistema) são assíncronas e não bloqueiam a *thread*. Diferentemente dos servidores tradicionais, a *thread* não fica esperando que essas operações sejam concluídas para continuar sua execução.

A Figura 16 representa a diferença de funcionamento de um servidor web tradicional e um Node.JS.

Dentre as vantagens de utilizar o Node.js, pode-se citar a flexibilidade, por possuir um extenso repositório de pacotes utilitários, facilitando a reutilização de código. Criar um ambiente Node.js e instanciar uma aplicação é uma tarefa que não exige muitos recursos computacionais em comparação com outras tecnologias mais tradicionais, tornando-se a melhor opção para aplicações que necessitam escalabilidade, situação usual na IoT. Também, por ser amplamente utilizado pela comunidade científica, bem como por empresas de tecnologia, há uma grande quantidade de material disponível para uso.

No EXEHDA-HS o Node.js é utilizado como base de todos os componentes. Por meio do pacote Express.js[1], a arquitetura disponibiliza uma API RESTFul empregando o protocolo HTTP. A comunicação P2P utilizada é gerenciada pela libP2P, outro pa-

[1] https://expressjs.com/

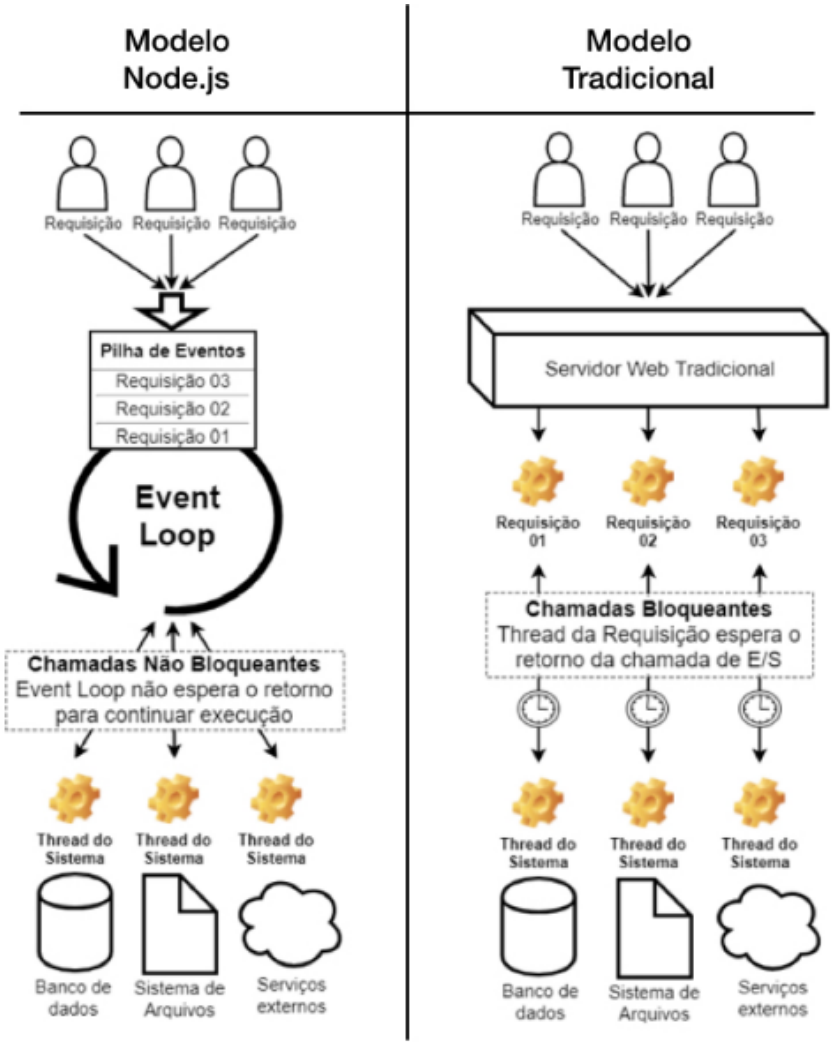


Figura 16 – Comparação do modelo Node.JS com o tradicional. (OPUS, 2019)

cote disponível nos repositórios do Node.js. A gerência do RID também é feita por uma biblioteca, a Sequelize[2]. Por meio desta biblioteca, a arquitetura de software do EXEHDA-HS consegue armazenar e resgatar informações empregando um driver de banco de dados.

## Comunicador LibP2P

A libP2P é uma conjunto de ferramentas para concepção de redes P2P introduzida pela comunidade IPFS (*InterPlanetary FileSystem*) (DIAS; BENET, 2016). A libP2P é capaz de indexar outros clientes e redes sem recorrer a registros centralizados, permitindo que os aplicativos funcionem de maneira off-line.

Com o objetivo de resolver os problemas dos sistemas centralizados de endereçamento, surgiu o IPF, o qual é um protocolo web descentralizado baseado em endereçamento de conteúdo, assinaturas digitais e distribuição P2P. Hoje, o IPFS é usado

[2]https://sequelize.readthedocs.io/

para construir aplicativos da web distribuídos que também estão disponíveis offline. O IPFS salva e distribui conjuntos de dados valiosos e pode movimentar bilhões de arquivos (BENET, 2014).

O IPFS gerou vários outros projetos e o libP2P é um deles. Ele permite que os usuários executem aplicativos de rede livres de tempo de execução e serviços de endereçamento enquanto são independentes de sua localização. A libP2P resolve a complexidade de lidar com vários protocolos em um ambiente descentralizado. Ele efetivamente ajuda os usuários a se conectarem com vários *peers* usando apenas um único protocolo.

A interface libP2P atua sobre um conjunto de subsistemas que são necessários para que os *peers* possam se comunicar. Esses subsistemas podem ser construídos sobre outros subsistemas, desde que respeitem a interface padronizada. As principais áreas em que esses subsistemas se encaixam podem ser visualizadas na Figura 17 e são definidas da seguinte maneira:

- *Peer Routing*: mecanismo para decidir quais *peers* usar para rotear mensagens específicas. Esse roteamento pode ser feito recursivamente, iterativamente ou até mesmo em um modo *broadcast/multicast*;
- *Content Routing*: responsável por manipular recursos no que diz respeito a busca e troca de informações;
- *NAT Traversal*: lida com a travessia de NAT (*Network Address Translation*), sendo capaz de estabelecer e manter conexões de Internet por meio de *gateways*;
- *Transports*: subsistemas que implementam camadas de transporte para troca de mensagens;
- *Resource Location*: encontra e identifica outros *peers* na rede.

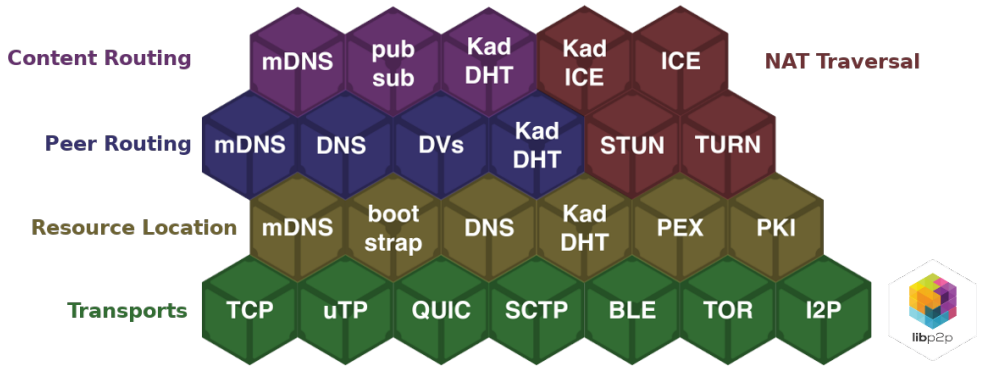


Figura 17 – Conjunto de ferramentas de rede do libP2P. Fonte: (LIBP2P, 2019)

A libP2P, especificamente no EXEHDA-HS, opera através de Remote Procedure Call (RPC) ou Procedimento de Chamada Remota. RPC é uma estratégia de comunicação entre processos que permite a um programa de computador chamar um procedimento em outro espaço de endereçamento (geralmente em outro computador, conectado por uma rede). Utilizando esta estratégia em conjunto com a libP2P, possibilita que o módulo de comunicação P2P do EXEHDA-HS atue na intercomunicação P2P e na gerência da Tabela Hash Distribuída (DHT). Disponibilizando de pontos de entrada e saída de dados criados utilizando o padrão RESTFul, fornecendo métodos primitivos para operações como *put()* e *get()* provendo manutenção e busca de recursos no ambiente celular.

Além das características citadas, a libP2P entrega como experiência a possibilidade de construir um sistema distribuído, no qual permite que desenvolvedores decidam como os aplicativos interagem com outros na rede sem se preocupar com aspectos relacionados a configuração da rede.

## Multicast Domain Name System

O mDNS é um serviço que oferece suporte para o procedimento de identificar dispositivos empregando o serviço local de DNS ativo na célula de execução do EXEHDA (STOLIKJ et al., 2014). Sendo assim, o mDNS solicita informações enviando uma mensagem *multicast* IP para todos os nós no domínio local. Por essa consulta, o cliente solicita que os dispositivos que têm o nome dado respondam de volta. Quando a máquina de destino recebe seu nome, ela envia uma mensagem de resposta que contém seu endereço IP e suas informações descritivas. Neste caso, todos os dispositivos na rede que obtêm a mensagem de resposta atualizam seu cache local usando o nome e o endereço IP fornecidos.

O mDNS é uma opção apropriada para dispositivos baseados na Internet incorporados devido aos seguintes fatores: a) Não há necessidade de reconfiguração manual ou administração extra para gerenciar dispositivos; b) É capaz de funcionar sem infraestrutura; e c) É capaz de continuar trabalhando se ocorrer falha na infraestrutura.

No EXEHDA-HS o mDNS é utilizado como serviço de descoberta local de recursos. Mais especificamente, este serviço opera dentro do modulo de descoberta de recursos enviando solicitações de descrição a todos os nós da rede. Ele também coopera com os modulos gerenciador de recursos e RID, para manter atualizadas as informações acerca do estado atual.

## MQTT

O MQ Telemetry Transport (MQTT) é um protocolo de rede leve usado para publicação/assinatura de mensagens entre dispositivos (YASSEIN et al., 2017). Este protocolo foi desenvolvido pela IBM em 1999. Sua proposta é atuar principalmente, mas não exclusivamente, em ambientes onde há baixa largura de banda ou redes instáveis bem como em dispositivos embarcados com recursos limitados de memória e processamento.

O protocolo MQTT é baseado no princípio de publicar mensagens e assinar tópicos, ou "pub/sub". Vários clientes se conectam a um broker e se inscrevem em tópicos nos quais estão interessados. Os clientes também se conectam ao broker e publicam mensagens nos tópicos. O broker e o MQTT atuam como uma interface simples e comum para tudo se conectar. Isso significa que se você tiver clientes que descarregam mensagens assinadas para um banco de dados, como um arquivo de texto simples, torna-se muito simples adicionar novos sensores ou outra entrada de dados a um banco de dados. A Figura 18 representa a relação entre o os clientes e o broker.

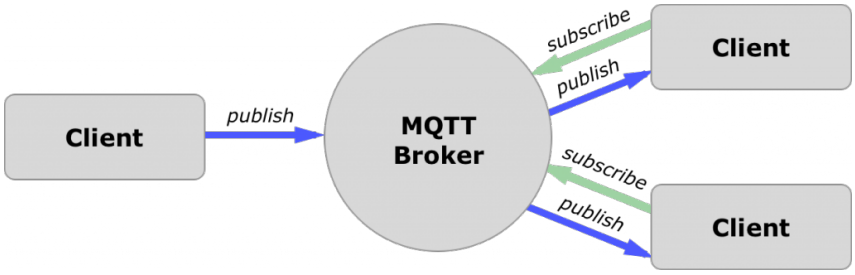


Figura 18 – Relação do broker MQTT com seus publicadores/assinantes.
Fonte: (1SHEELD, 2019)

Mensagens no MQTT são publicadas em tópicos. Os tópicos são tratados como uma hierarquia, usando uma barra (/) como separador. Isso permite que uma organização de temas comuns seja criada, da mesma maneira que um sistema de arquivos. Por exemplo, vários clientes podem publicar suas informações de temperatura ou estado no tópico a seguir:

**sensors/<CLIENT_ID>/temperature/**

Os clientes também podem receber mensagens criando assinaturas. Uma assinatura pode ser para um tópico específico, onde apenas mensagens para esse tópico serão recebidas ou, ainda, pode-se incluir curingas. Dois curingas estão disponíveis + ou #:

- + pode ser usado como curinga para um único nível de hierarquia. Pode ser

usado com o tópico para obter informações sobre todos os sensores de temperatura seguinte maneira: **sensors/+/temperature/+**;

- # pode ser usado como um curinga para todos os níveis restantes de hierarquia. Isso significa que ele deve ser o caractere final de uma assinatura. Pode ser usado com o tópico para obter informações sobre todos os sensores existentes no tópico da seguinte maneira: **sensors/#**.

No EXEHDA-HS, o MQTT é utilizado como um dos protocolos de comunicação disponíveis, possibilitando que os clientes publiquem e atualizem seu estado, bem como subscrever a tópicos os quais podem representar contextos, sensores e tipos de recursos. O MQTT, neste caso, atua utilizando a estratégia de diretório como pode ser observado no seguinte exemplo:

- **celula/<CELULA_ID>/recursos/<RECURSO_ID>**;
- **celula/<CELULA_ID>/descoberta**.

Atuando com esta estratégia de diretórios, a arquitetura disponibiliza a possibilidade de publicar estados, subscrever a recursos, células, tópicos em geral e ainda trocar informações a respeito da descoberta de recursos como anúncios e busca de recursos.

## Banco de dados MongoDB

O MongoDB é um banco de dados baseado em software livre, orientado a documentos desenvolvido na linguagem de programação C++ (ARORA; AGGARWAL, 2013). O MongoDB inovou tornando o armazenamento de dados mais rápido e fácil usando esquemas de banco de dados dinâmicos semelhantes ao JSON em vez de sistemas de bancos de dados relacionais tradicionais de tabelas e SQL. É por esse motivo que o MongoDB é categorizado como o servidor de banco de dados NoSQL, ou seja, que não utiliza SQL. O esquema do banco de dados dinâmico usado no MongoDB é chamado de BSON.

Este banco de dados foi desenvolvido em 2007 por uma organização com sede em Nova York, a 10gen, que agora se chama MongoDB Inc. O MongoDB foi desenvolvido inicialmente como PAAS (*Platform as a service*). Mais tarde, no ano de 2009, o MongoDB foi introduzido no mercado como um servidor de banco de dados de código aberto que foi mantido e apoiado pelo MongoDB Inc. Muitas organizações de grande e médio porte como SourceForge, Foursquare, craigslist e eBay estão usando o MongoDB em desenvolvimento de suas aplicações de banco de dados.

O MongoDB é recomendado em cenários em que o processamento rápido e a simplicidade são a chave. Devido ao seu mecanismo de consulta NoSQL (*not only*

*SQL*), o torna robusto, escalável e altamente eficiente. Existem várias vantagens de usar o MongoDB em servidores de banco de dados contemporâneos.

- Simplicidade: por ser um banco de dado NoSQL, no MongoDB os armazenamentos são orientados a documentos JSON, simplificando as estruturas de dados;
- Replicação e Confiabilidade de Dados: o MongoDB permite que os usuários repliquem dados em vários servidores espelhados, o que garante a confiabilidade dos dados. No caso de um servidor travar, seu espelho ainda estará disponível e o processamento do banco de dados permanecerá inalterado;
- Consultas NoSQL: o MongoDB possui mecanismo de consulta NOSQL que resulta em funcionalidades de armazenamento e recuperação de dados extremamente velozes. As consultas orientadas a documentos com base em JSON são extremamente rápidas em comparação com as consultas SQL tradicionais;
- Migrações de Esquema: no MongoDB, o esquema é definido pelo código. Portanto, no caso de migrações de banco de dados, não ocorre nenhum problema de compatibilidade de esquema;
- Escalabilidade horizontal eficiente: como o MongoDB é um banco de dados não relacional, ele é mais adequado para os cenários em que a escalabilidade horizontal é importante;
- Open Source: por último, mas não menos importante, o MongoDB é um servidor de banco de dados que é de código aberto e personalizável de acordo com os requisitos da organização.

No componente RID, o armazenamento de dados utiliza o MongoDB como banco de dados. Esta escolha aconteceu por esta tecnologia ser amplamente utilizada em aplicações IoT. Aplicações IoT utilizam o MongoDB por possuir baixa latência de aquisição de informações, por utilizar poucos recursos computacionais e pela facilidade de utilização com relação a drivers e servidores.

O desenvolvimento do componente RID faz uso de uma técnica chamada ORM (*Object-Relational Mapping*). Esta técnica é utilizada para converter dados entre tipos de sistemas incompatíveis usando orientação a objetos. Nestes casos, as tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes. Isso cria um “banco de dados de objetos virtuais” que pode ser usado dentro de qualquer linguagem de programação.

### Common Open Research Emulator

O *Common Open Research Emulator*, ou CORE, é um *framework* para emular redes em *desktops*, com a possibilidade de interconectar redes construídas em mais de um equipamento. O CORE emula roteadores, nodos de processamento e simula os links de rede entre eles (AHRENHOLZ, 2010).

Como se trata de uma emulação ao vivo, essas redes podem ser conectadas em tempo real a redes e roteadores físicos externos ao CORE. O CORE usa uma virtualização do FreeBSD a fim de estender redes físicas para planejamento, teste e desenvolvimento, sem a necessidade de implementações de hardware dispendiosas.

Com o emprego do emulador CORE foi viabilizada a construção de cenários de uso para avaliação EXEHDA-HS, levando em consideração aspectos reais das redes de computadores e suas camadas. Assim, com o emprego do CORE foi possível criar células de execução do EXEHDA, com diferentes perfis operacionais.

É importante registrar que para atender as demandas de avaliação do EXEHDA-HS, se fez necessário produzir uma versão operacional do CORE sobre a última versão disponível da distribuição Ubuntu quando da concepção do EXEHDA-HS (versão 18).

Esta versão atualizada de uma máquina virtual para CORE sob Ubuntu foi disponibilizada para a comunidade de desenvolvimento do emulador[3].

## 4.5 Considerações Finais do Capítulo

Este capítulo apresentou a concepção do serviço de descoberta de recursos do EXEHDA-HS, direcionado a atender as demandas do *middleware* EXEHDA, no que diz a respeito a descoberta de recursos na IoT.

Os componentes arquiteturais e suas funcionalidades são apresentados, bem como as tecnologias adotadas para a concepção do serviço. Além destas tecnologias, o EXEHDA-HS considerou outras utilizadas no desenvolvimento do *middleware* EXEHDA. Dentre estas podemos citar: (i) PHP; (ii) Codeigniter; (iii) PostgreSQL; e (iv) Bootstrap. Estas tecnologias foram consideradas quando da especificação de procedimentos para integração da arquitetura do EXEHDA-HS com os outros subsistemas do *middleware*, mais especificamente com o Subsistema de Adaptação e Reconhecimento de Contexto.

[3] https://drive.google.com/open?id=1zPROSoi3XNodOi2hptPShHzr907m9lmm

# 5 EXEHDA-HS: CENÁRIOS DE USO

Neste capítulo são discutidos os cenários de uso empregados para a sistematização dos diferentes procedimentos de busca do EXEHDA-HS. Os cenários de uso exploram diferentes funcionalidades do EXEHDA-HS no que diz respeito a descoberta de recursos, bem como introduzem os diferentes procedimentos de gerência necessários à sua operação.

Considerando aspectos de reprodutibilidade, bem como de potencial de reorganização da infraestrutura computacional empregada nos cenários de uso, foi explorado o emulador CORE. Com este emulador foi possível construir diferentes organizações tanto de topologia como de natureza de equipamentos envolvidos. A organização da topologia de testes pode ser observada na Figura 19.

Para o desenvolvimento destes cenários de uso, além dos códigos inerentes ao EXEHDA-HS, se fez necessário instanciar as diferentes bibliotecas para a operação do EXEHDA-HS. Para atender esta situação se fez necessária uma instanciação do próprio emulador em uma máquina virtual executando a distribuição Ubuntu 18, a fim de compatibilizar as versões das diferentes bibliotecas envolvidas.

## 5.1 Cenário 1: Comportamento do Procedimento de Busca no Ambiente Celular

Nesta seção são descritos os comportamentos adotados pela arquitetura em cenários considerando a localização e o objetivo da ação chamada pela Aplicação Cliente.

**Descrição do Cenário**

Considerando que uma Aplicação Cliente possui a necessidade de buscar um recurso no ambiente celular do EXEHDA, e tendo por base a localização do recurso e a origem da busca, os procedimentos são descritos considerando as seguintes situações: recurso na célula, recurso em célula vizinha e recurso em célula não vizinha.

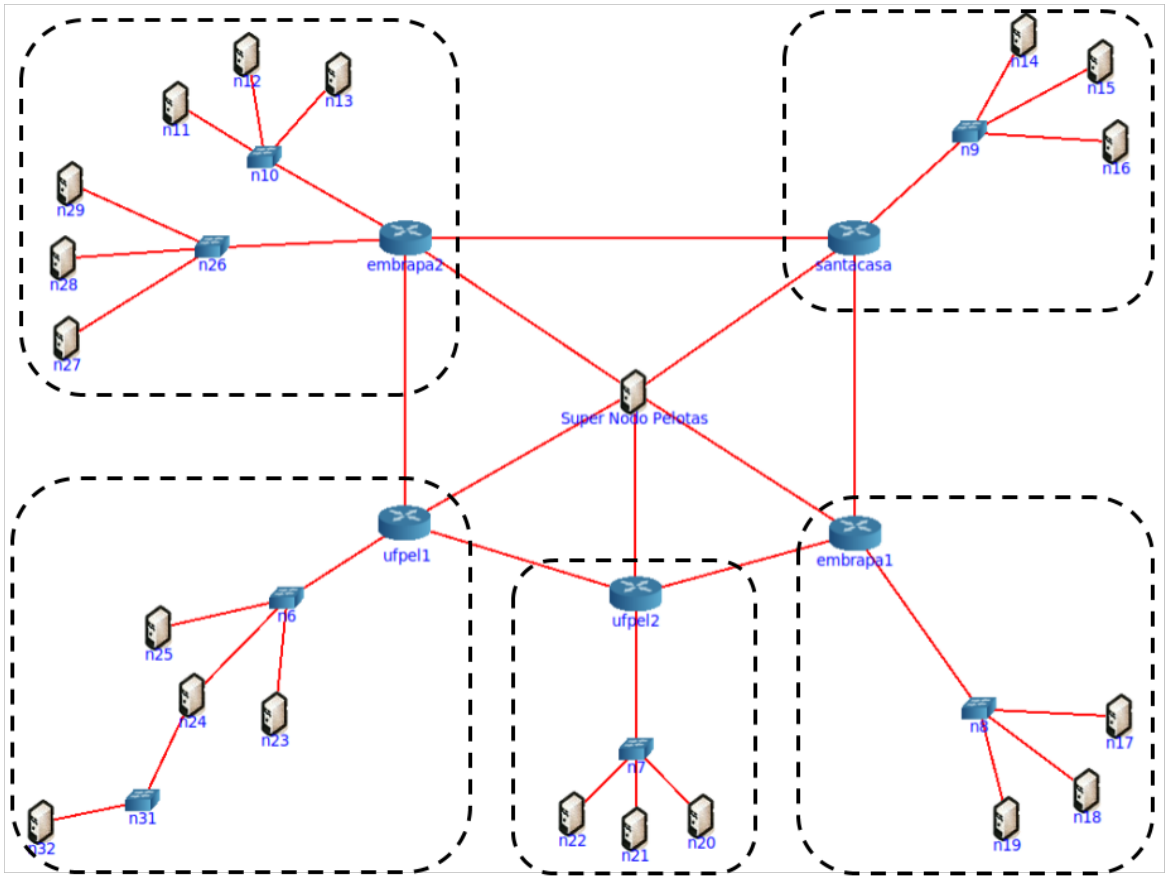


Figura 19 – Topologia utilizada para os cenários de uso do EXEHDA-HS empregando o emulador CORE

### 5.1.1 Caso 1: Recurso na Célula de Execução da Aplicação Cliente

Neste caso, a Aplicação Cliente que solicita a ação de descoberta de recursos está localizada na mesma Célula de Execução em que o recurso está disponível.

A aplicação direciona a requisição de busca ao Componente Buscador do EXEHDA-HS, localizado na EXEHDA-base da célula. O Buscador, por sua vez, consulta o componente RID (Repositório de Informações Descobertas) da célula em busca do recurso solicitado.

Caso o Componente Buscador encontre o recurso solicitado pelo cliente, as informações relacionadas a este recurso são agregadas e retornadas para o mesmo.

Se o retorno à Aplicação Cliente não ocorrer em um tempo máximo especificado como TTL, o recurso é entendido como não disponível. Particularmente nas buscas não locais, o parâmetro TTL apresenta um significado maior.

Uma representação deste caso pode ser observado na Figura 20 onde: (i) as setas azuis representam a requisição feita ao Subsistema de Execução Distribuída da célula em busca de um recurso; (ii) as setas verdes representam a busca pelo recurso em um nó presente na mesma célula; e por fim (iii), as setas amarelas representam a

resposta direcionada tanto à origem da requisição, como ao Subsistema de Execução Distribuída com o intuito de manter as informações salvas e atualizadas no RID.

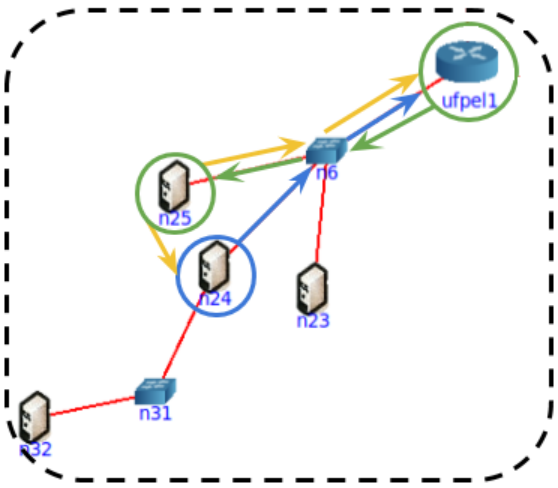


Figura 20 – Representação de busca por recurso que está presente na mesma célula de origem da requisição.

### Componentes Arquiteturais envolvidos

Para este caso 1 do cenário apresentado, os seguintes componentes arquiteturais do EXEHDA-HS interoperam para provimento da funcionalidade desejada:

- Gerenciador de Comunicação: este componente recebe as requisições de busca feitas pelas aplicações clientes e aciona os componentes necessários;
- Buscador: este componente verifica qual RID deve ser empregado (neste caso o RID local à célula de execução) e transfere a solicitação ao mesmo;
- Repositório de Informações Descobertas (RID): através de consultas em banco, este componente busca por recursos utilizando os parâmetros definidos pela Aplicação Cliente;
- Classificador de Recursos: este componente classifica os resultados levando em consideração também os parâmetros definidos pela Aplicação Cliente.

### Fluxo de dados

O diagrama de fluxo de dados que desencadeia a busca para o caso 1 pode ser observado na Figura 21.

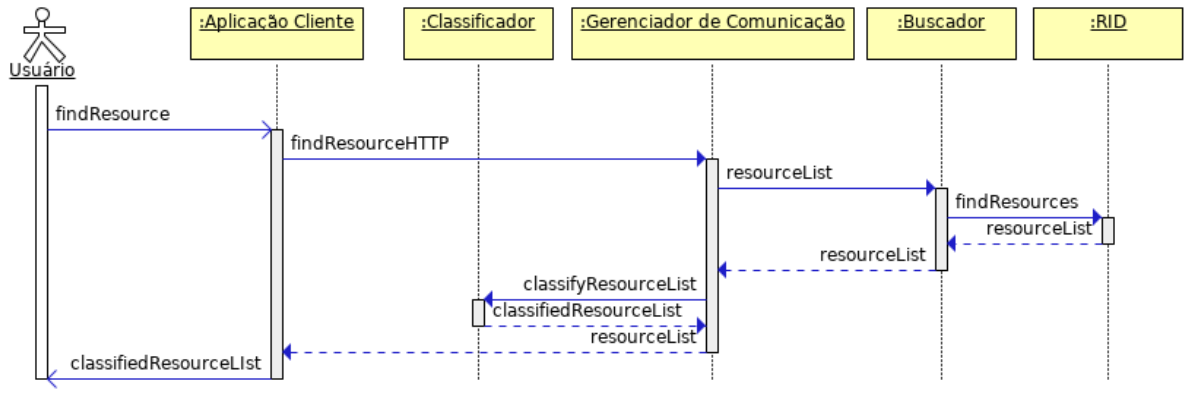


Figura 21 – Representação do fluxo de dados do cenário 1, caso 1.

### 5.1.2 Caso 2: Recurso presente em Célula Vizinha a de Execução da Aplicação Cliente

Neste caso, a Aplicação Cliente que solicita a descoberta de recursos está localizada em uma célula vizinha.

A aplicação direciona a requisição de busca ao Componente Buscador do EXEHDA-HS, localizado na EXEHDA-base. O Buscador, por padrão, redireciona a mesma para células vizinhas toda vez que o recurso não for encontrado localmente ou que seja explicitamente definido que a busca também deve percorrer células vizinhas.

Este cenário pode ocorrer, por exemplo, quando o cliente deseja encontrar todos os recursos disponíveis que contenham sensor de temperatura ambiente nas células vizinhas.

Neste caso, uma requisição de busca é enviada a todas as células vizinhas e, ao ser recebida por estas células, também é direcionada ao componente Buscador do EXEHDA-HS. Este componente, por sua vez, consulta seu RID local em busca do recurso solicitado.

Caso a Componente Buscador encontre o recurso solicitado pelo cliente, as informações relacionadas a este recurso são agregadas e retornadas para o cliente.

Se o retorno à Aplicação Cliente não ocorrer em um tempo máximo especificado como TTL, o recurso é entendido como não disponível nas células vizinhas.

A representação deste caso pode ser observada na Figura 22 onde: (i) as setas azuis representam a requisição feita pelo Nodo 23 ao Subsistema de Execução Distribuída da célula em busca de um recurso; (ii) a seta verde representa a busca pelo recurso em uma célula vizinha conhecida pela célula de origem; (iii) as setas roxas indicam a busca local da célula vizinha pelo recurso desejado; e por fim (iv) as setas amarelas representam a resposta direcionada à origem da requisição e ao Subsistema de Execução Distribuída com o intuito de manter as informações salvas e atualizadas no RID.

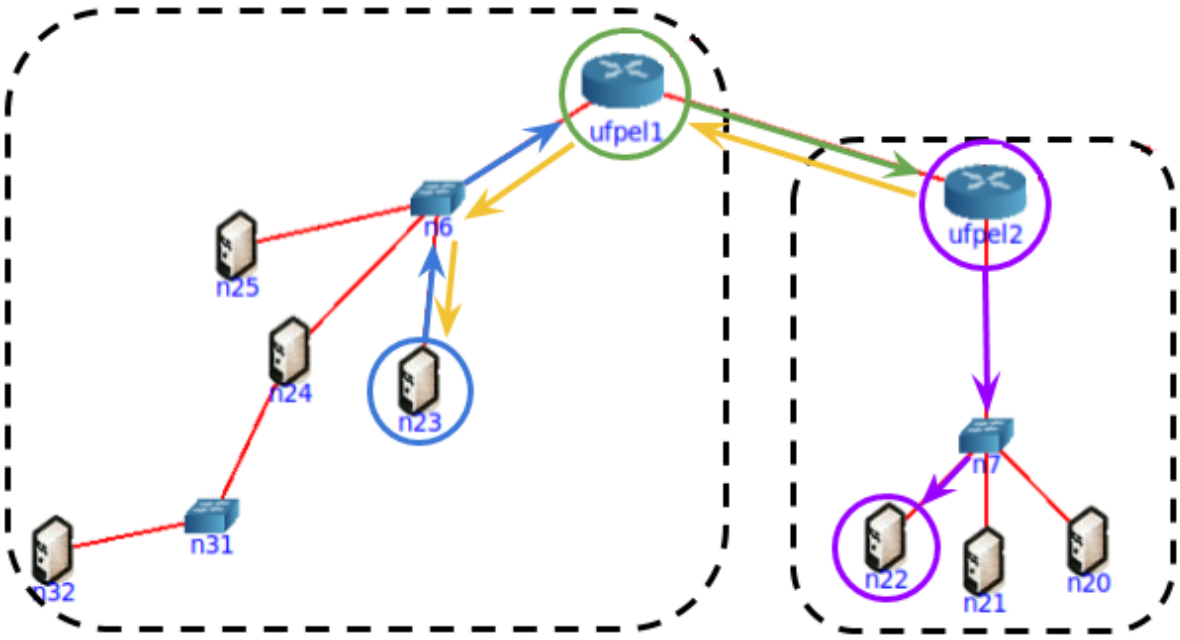


Figura 22 – Representação de busca por recurso presente em uma célula vizinha.

### Componentes Arquiteturais envolvidos

Para este caso 2 do cenário apresentado, os seguintes componentes arquiteturais do EXEHDA-HS interoperam para provimento da funcionalidade desejada:

- Gerenciador de Comunicação: este componente recebe as requisições de busca da Aplicação Cliente e de outras células quando da busca por um recurso;
- Buscador: este componente busca pelos recursos em seu repositório local ou a redireciona para células vizinhas;
- Repositório de Informações Descobertas: através de consultas SQL, este componente busca por registros utilizando os parâmetros definidos pela Aplicação Cliente;
- Classificador de Recursos: este componente classifica os resultados levando em consideração os parâmetros definidos pela Aplicação Cliente quando da especificação do recurso a ser buscado.

### Fluxo de dados

O diagrama de fluxo de dados que desencadeia a busca para o caso 2 pode ser observado na Figura 24, onde as flechas de cor vermelha representam as requisições feitas para a célula vizinha.

### 5.1.3 Caso 3: Recurso presente em Célula não Vizinha a de Execução da Aplicação Cliente

Neste caso, o recurso solicitado pela Aplicação Cliente não está localizado em uma célula vizinha. Para uma célula ser considerada não vizinha ela deve estar ativa no ambiente celular do EXEHDA, porém ainda não ter disponibilizado recurso para uma Aplicação Cliente localizada na célula originária da busca.

A requisição de busca, ao ser recebida por uma célula não vizinha, também é direcionada ao componente Buscador do EXEHDA-HS. Este componente, por sua vez, consulta seu RID local em busca do recurso solicitado.

Uma vez que seja encontrado o recurso solicitado em uma determinada célula, a mesma precisa ser certificada pela célula originária da busca. Certificar uma célula significa verificar seus metadados e após adiciona-la a uma lista de células certificadas. Este procedimento de validação, por questões de segurança, deve ser feito manualmente. Uma vez certificada uma célula, esta permanece nesta condição até deliberação em contrário pelo administrador do *middleware*.

Três critérios são considerados para limitar a busca em células não vizinhas:

- Limite por TTL: se o retorno à Aplicação Cliente não ocorrer em um tempo máximo especificado como TTL, a busca é interrompida e o recurso é entendido como não disponível;
- Limite por número de Saltos: o repasse da busca irá se estender considerando um número máximo de saltos a partir da célula originária da busca;
- Dispersão geográfica: a busca irá se estender a todas as células de execução cuja localização geográfica esteja dentro de um raio especificado pela Aplicação Cliente.

A representação deste caso pode ser observada na Figura 23, na qual: (i) as setas azuis representam a requisição feita ao Subsistema de Execução Distribuída de uma célula em busca de um recurso; (ii) a seta verde representa o redirecionamento desta busca a uma de hierarquia superior, célula mãe; (iii) as setas vermelhas mostram as requisições encaminhadas às outras células do escopo hierárquico desta célula mãe; (iv) as setas roxas indicam a busca pelo recurso desejado nas várias células gerenciadas pela célula mãe; e por fim (v) as setas amarelas representam a resposta direcionada à célula originária da requisição, bem como ao Subsistema de Execução Distribuída da célula mãe com o intuito de manter as informações consistentes e atualizadas para buscas futuras.

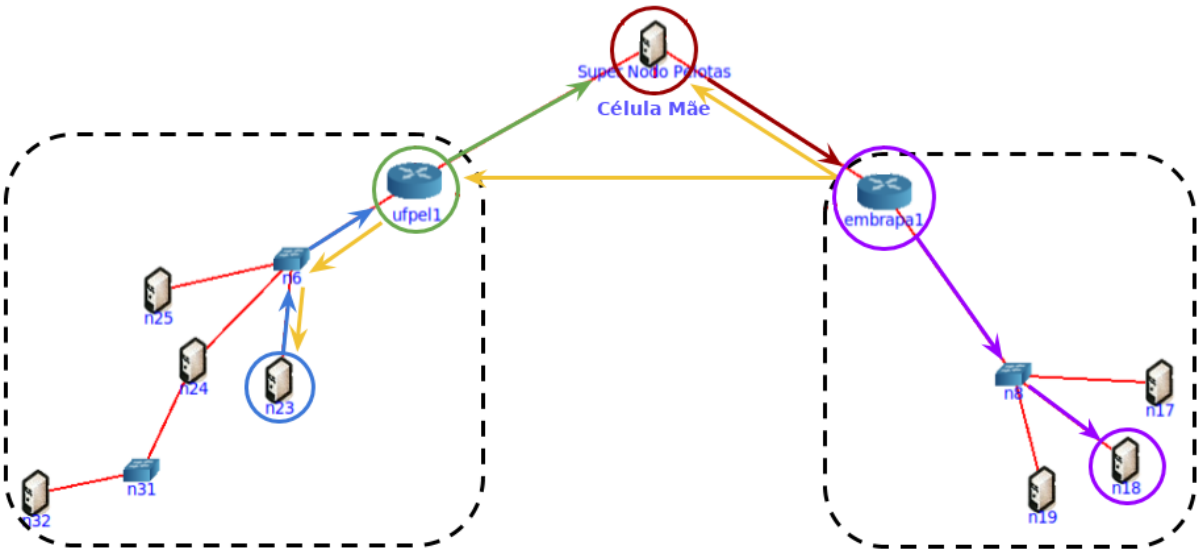


Figura 23 – Representação de busca por recurso em célula não vizinha.

### Componentes Arquiteturais envolvidos

Para este caso 3 do cenário apresentado, os seguintes componentes arquiteturais do EXEHDA-HS interoperaram para provimento da funcionalidade desejada:

- Gerenciador de Comunicação: este componente recebe as requisições de busca da Aplicação Cliente, bem como de outras células em busca de um recurso;
- Buscador: este componente busca pelos recursos em seu RID e no super nodo (célula mãe) por células não vizinhas;
- Repositório de Informações Descobertas (RID): através de consultas SQL, este componente busca por registros utilizando os parâmetros definidos pelo cliente.
- Classificador de Recursos: este componente classifica os resultados levando em consideração os parâmetros definidos pelo Aplicação Cliente. Considerando que a busca em células não vizinhas pode retornar recursos com diferentes aspectos e perfis, o Classificador de Recursos ganha ainda mais significado retornando uma lista ordenada para a Aplicação Cliente que considera aspectos não funcionais como precisão, disponibilidade, latência e etc.

### Fluxo de dados

O diagrama de fluxo de dados que desencadeia a busca para o caso 3 pode ser observado na Figura 25, onde as flechas de cor verde representam a solicitação de busca a uma célula mãe, e as flechas de cor vermelha representam as requisições feitas para as células gerenciadas por esta célula mãe.

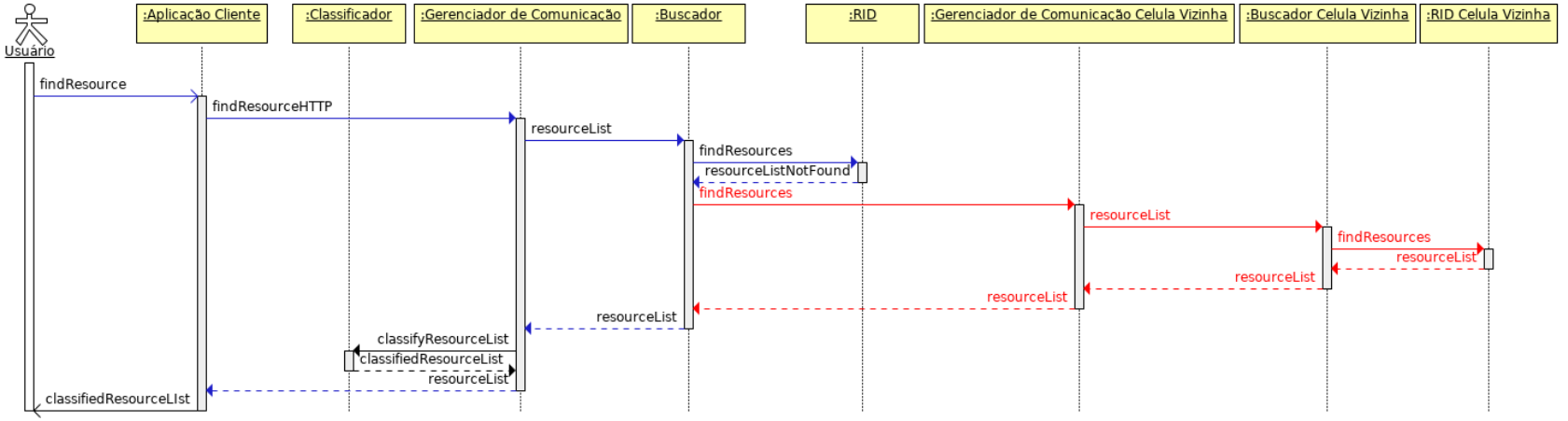


Figura 24 – Representação do fluxo de dados do cenário 1, caso 2.

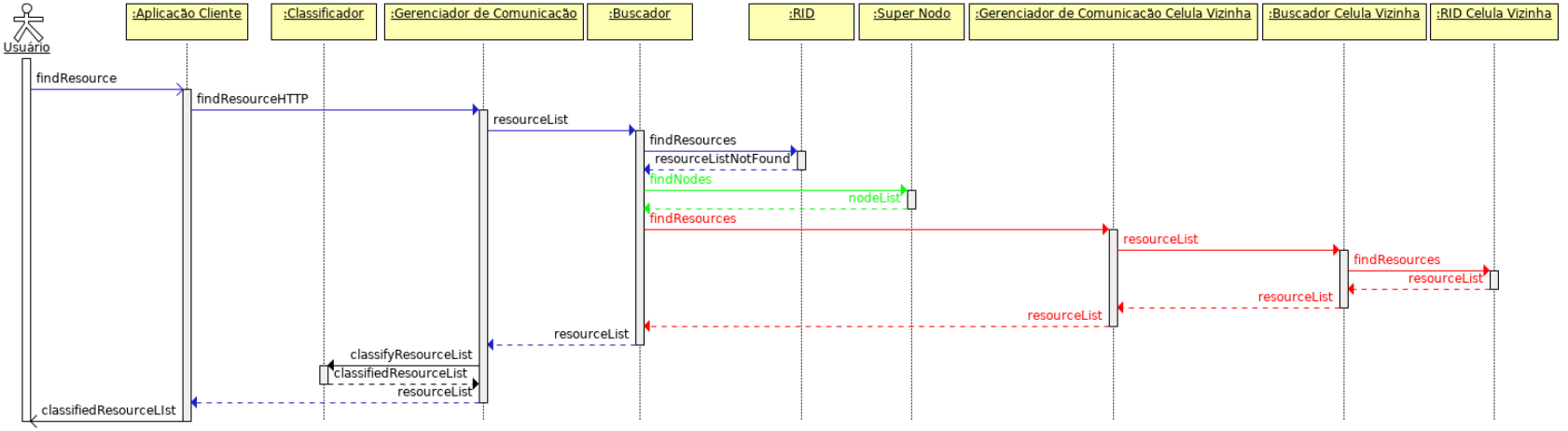


Figura 25 – Representação do fluxo de dados do cenário 1, caso 3.

## 5.2 Cenário 2: Aplicação no Cenário da empresa Embrapa Clima Temperado

Este cenário de uso do EXEHDA-HS apresenta uma aplicação direcionada a empresa Embrapa (Empresa Brasileira de Pesquisa Agropecuária), particularmente a unidade Clima Temperado. Este estudo de caso envolve a descoberta e busca de recursos, promovida pelo EXEHDA-HS, considerando a localização do recurso desejado, sua disponibilidade, perfil e a preferências do pesquisador.

A Embrapa Clima Temperado, vinculada ao Ministério da Agricultura, Pecuária e Abastecimento, está localizada em Pelotas, Rio Grande do Sul, Brasil. Esta unidade da Embrapa é comprometida com a Pesquisa Eco-regional e desenvolve atividades que buscam viabilizar soluções de pesquisa, desenvolvimento e inovação para a sustentabilidade da agricultura na região de clima temperado, em benefício da sociedade.

A escolha da Embrapa Clima Temperado deve-se ao fato da mesma fazer parte do projeto plenUS (*plentiful of Ubiquitous Systems*). Este projeto tem por missão disponibilizar sistemas computacionais que explorem relações proativas entre usuários, software e equipamentos, com intuito de promover soluções computacionais que contribuam de forma sinérgica no atendimento das atividades fins da empresa.

### 5.2.1 Caso 1: Possibilitando a intercomunicação entre sensores com dificuldades de conectividade

Em regiões onde não existe infraestrutura de rede para promover a interoperabilidade entre equipamentos, o acesso às informações produzidas de forma distribuída constitui um problema recorrente. Situação típica de estações de sensoriamento distribuídas no meio rural, cujas informações muitas vezes são obtidas por inspeção manual, exigindo deslocamentos do usuário.

No caso do Projeto plenUS, que utiliza o EXEHDA, uma célula de execução compreende um conjunto de estações de sensoriamento distribuídas em uma região da propriedade rural, monitorando uma parte das zonas de cultivo.

A comunicação entre os equipamentos de uma célula deve acontecer através de uma camada física de rede, seja ela com ou sem fios. Considerando que estes equipamentos se encontram em regiões de difícil acesso e sem suporte de redes estruturadas, uma alternativa é a criação de uma rede onde cada célula teria comunicação com sua vizinhança e a partir dela, com o restante do ambiente celular.

O EXEHDA-HS, por empregar o conceito de rede P2P, disponibiliza a possibilidade de criação de redes com topologias específicas, que podem ser definidas na configuração inicial das células de execução envolvidas.

Este caso pode ser analisado na Figura 26, onde observamos sete células de

execução, cada uma contendo uma estação rádio-base empregando o padrão Lora[1]. Estas estações rádio-base tem alcance limitado atingindo somente suas células vizinhas.

Esta infraestrutura de testes considerando demandas da Embraba Clima Temperado foi discutida no artigo (JOAO et al., 2019), no qual foi explorada a possibilidade de comunicação entre células vizinhas, permitindo a exploração de uma abordagem baseada em *Fog Computing*.


Figura 26 – Sensoriamento de propriedade rural empregando o EXEHDA-HS

Usando a rede provida pelo EXEHDA-HS, é feita uma descoberta das estações de sensoriamento disponíveis nas várias células, empregando o procedimento de busca em células vizinhas do EXEHDA-HS, já introduzido anteriormente.

Deste modo, além da disponibilização das informações coletadas de forma distribuída nas diferentes células de execução para o administrador da propriedade rural, um outro uso desta conectividade oferecida pelo EXEHDA-HS é a decisão de quando se mostra oportuno irrigar. Neste caso, são consideradas como variáveis contextuais a tensão de água no solo; a temperatura; e a umidade do ar, sendo os comandos de atuação disparados por meio do processamento de regra envolvendo estas variáveis, que precisam ser coletadas em células vizinhas.

### 5.2.2 Caso 2: Ferramenta de gerência do EXEHDA-HS

O caso 2 deste estudo de caso envolvendo a Embrapa Clima Temperado, tem como objetivo mostrar a integração do EXEHDA-HS com a aplicação de gerência de recursos concebida para uso junto ao plenUS na Embrapa Clima Temperado.

A aplicação de gerência de recursos desenvolvida para Embrapa está capacitada a cadastrar, ler, atualizar e remover dados do Repositório de Informações Contextuais do EXEHDA (RIC) (JOAO et al., 2018), bem como em outras bases de dados de

[1] https://lora-alliance.org/


Figura 27 – Ferramenta de visualização de informações contextuais do plenUS.

gerência empregadas pelo *middleware*.

Com isto, esta ferramenta pode oferece funcionalidades de visualização textuais e gráficas de informações produzidas por recursos presentes nas células de execução ativas na Embrapa. Um exemplo das funcionalidades de visualização pode ser observado na Figura 27, e por sua vez, funcionalidades de gerência na Figura 28.

O EXEHDA-HS, por meio da sua funcionalidade de certificação de célula e/ou recurso, evita que recursos de sensoriamento ou atuação não autorizados sejam incluídos no plenUS. Toda vez que um recurso é descoberto, para que ele seja adicionado como um recurso reconhecido pela célula, ele deve ser aceito pelo gerente do plenUS. Esta ação pode ser observada na Figura 29, onde a aplicação de gerência solicita a confirmação do gerente para adicionar o recurso na base de dados do plenUS, evitando assim o cadastramento de dispositivos não certificados.

Esta aplicação de gerência possibilita que os usuários tenham controle do ambiente celular provido pelo EXEHDA-HS e que efetuem tarefas administrativas por meio de uma aplicação com interface responsiva e acessível.

## 5.3 Considerações Finais do Capítulo

Neste capítulo foram apresentados dois cenários de uso com o objetivo de avaliar as funcionalidades do EXEHDA-HS para descoberta de recursos para na IoT: (i) cenário demonstrando os comportamentos adotados pela arquitetura considerando a

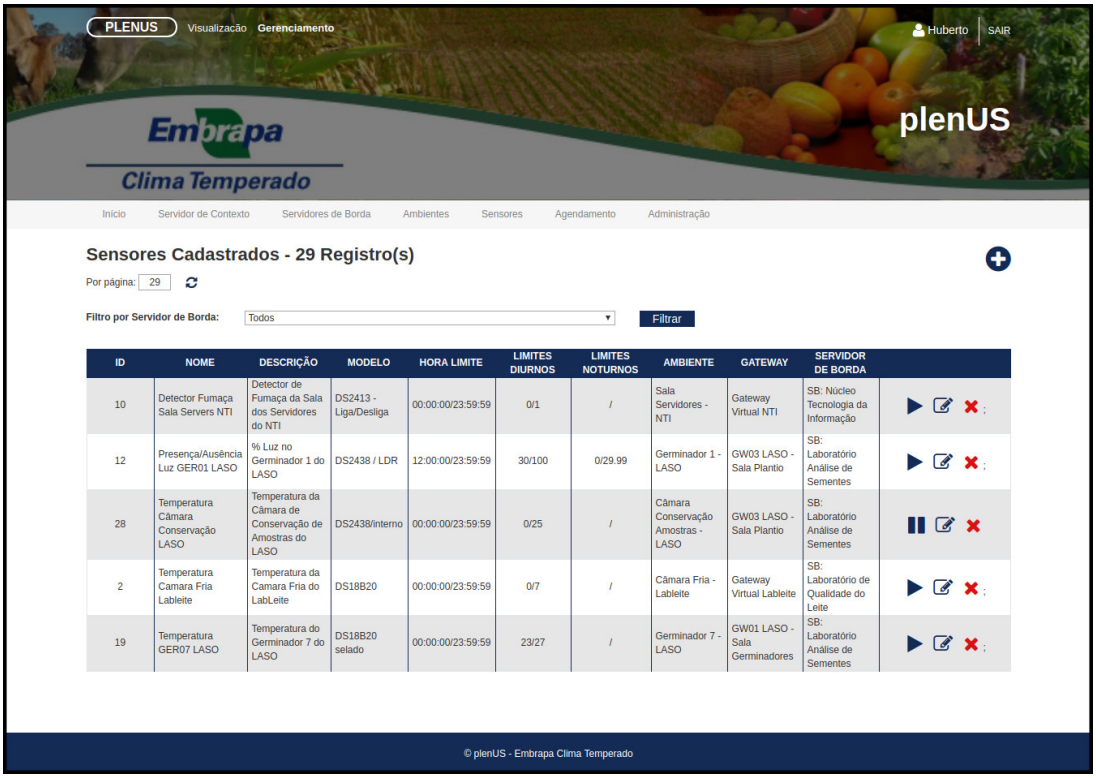


Figura 28 – Ferramentas de gerência de informações contextuais do plenUS.

localização dos recursos e os objetivos das ações tomadas; e (ii) cenário de uso que apresenta aplicações direcionadas a empresa Embrapa Clima Temperado que exploram a descoberta e busca de recursos, promovida pelo EXEHDA-HS.

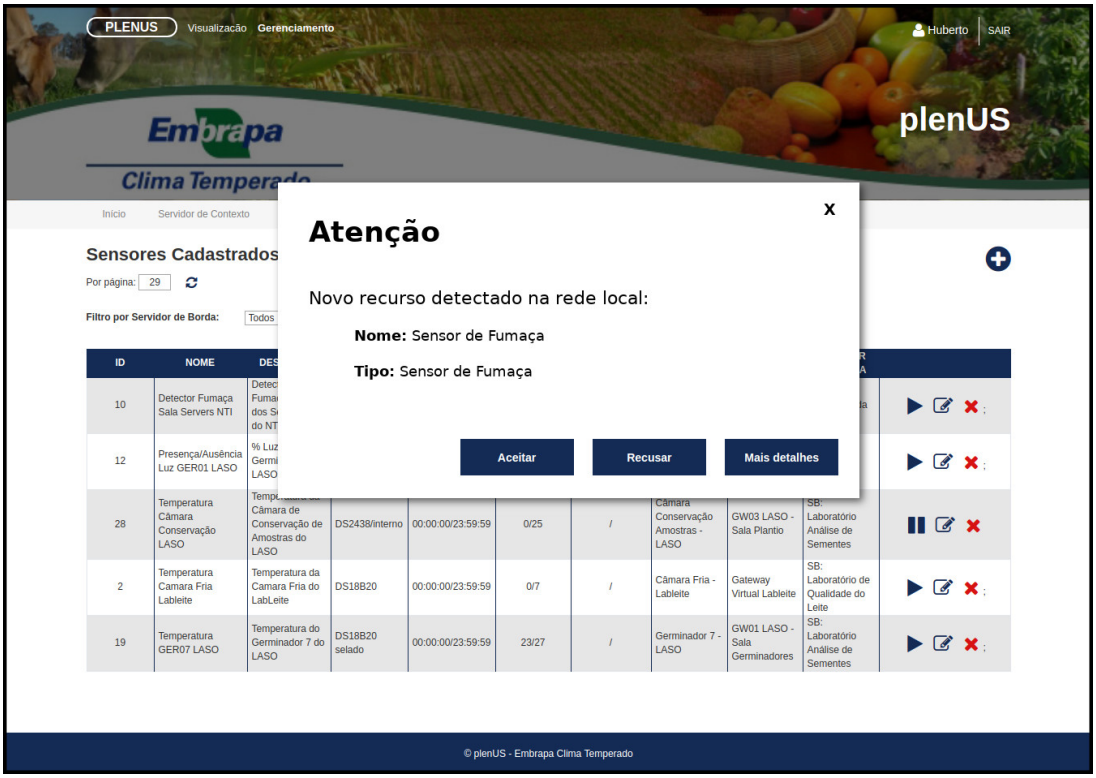


Figura 29 – Mensagem de novo recursos descoberto pelo EXEHDA-HS.

# 6 CONSIDERAÇÕES FINAIS

Este capítulo discute as principais conclusões decorrentes do trabalho desenvolvido nesta dissertação de mestrado, apresenta algumas publicações realizadas, bem como relacionar possíveis trabalhos futuros.

## 6.1 Principais Conclusões

Como principal contribuição desta dissertação destaca-se a concepção de uma proposta de arquitetura para Descoberta de Recursos, destinada a infraestrutura computacional típica da Internet das Coisas, denominada EXEHDA-HS.

Com esta proposta, foi incorporado ao EXEHDA suporte arquitetural com funcionalidades necessárias para a descoberta de recursos na IoT, bem como foram incorporados ao *middleware* padrões e tecnologias praticados na área.

Como decorrência de seu crescimento, a IoT vem ganhando destaque enquanto forma de materializar as premissas da Computação Ubíqua. Entretanto, para que seja possível a oferta de serviços computacionais de forma o mais transparente possível, premissa básica para viabilização prática da ubiquidade, a minimização da participação do usuário é um aspecto central.

Deste modo, dentre os diversos aspectos inerentes a operação da IoT, que tem forte alinhamento com as premissas de concepção do EXEHDA-HS, destacaríamos a necessidade que os procedimentos de agregação de recursos às aplicações aconteçam de forma o mais automatizada possível, reduzindo atividade de configuração por parte dos usuários. Outrossim, os esforços de concepção do EXEHDA-HS indicaram que a infraestrutura computacional típica da IoT apresenta elevada dinamicidade quanto a sua composição e/ou disponibilidade momentânea de recursos, o que potencializa os aspectos inerentes a sua elevada escalabilidade.

Outro aspecto necessário para minimizar os esforços de configuração, potencializando o foco dos usuários nas suas atividades de efetivo interesse, é o emprego de interfaces do mais alto nível possível, evitando especificações e/ou manipulações por parte do usuário de particularidades dos diferentes recursos que irão compor a plata-

forma de trabalho da sua aplicação. Entende-se que esta é outra contribuição que o EXEHDA-HS oferece, substituindo esforços de caracterização dos detalhes dos recursos, pela definição de critérios de busca empregando metadados caracterizados por uma maior abstração.

Nesta perspectiva, o *middleware* EXEHDA vem sendo desenvolvido no LUPS/UFPEL com o objetivo de reduzir o esforço de desenvolvimento de aplicações para a IoT. O EXEHDA tem sua organização dividida células de execução, as quais fornecem serviços. Os serviços fornecidos pelo *middleware* estão organizados em quatro subsistemas, sendo um destes o subsistema responsável pela execução distribuída com o qual esta dissertação contribui diretamente.

Para concepção do EXEHDA-HS foi realizada uma sistematização de conceitos relativos a descobertas de recursos, particularmente aqueles referentes a descoberta de recursos distribuídos em rede. Neste particular, foram identificados aspectos funcionais e não funcionais que prioritariamente devem ser atendidos. A observância destes aspectos foi central para escolha da arquitetura P2P a ser utilizada como base para concepção da arquitetura do EXEHDA-HS. Neste sentido, foi adotada uma arquitetura híbrida, baseada em Tabelas Hash Distribuídas, a qual propiciou o recurso de sobreposição de diferentes redes lógicas sobre o ambiente celular do EXEHDA. Estas redes lógicas são organizadas de forma hierárquica e tem como critério para composição um aspecto contextual, por exemplo, localização, tipos de sensores, etc.

O emprego de redes lógicas com o recurso de sobreposição, cada uma delas podendo ser gerenciada por uma célula mãe diferente, constitui uma abordagem central do EXEHDA-HS para tratar a elevada escalabilidade da IoT.

O esforço de empregar no EXEHDA-HS somente tecnologias abertas em sua concepção e atualizadas no que diz respeito a adoção pela comunidade, entende-se como uma abordagem promissora para o emprego do *middleware* em diferentes cenários de uso e para a continuidade do desenvolvimento.

Para a avaliação das funcionalidades do EXEHDA-HS foram abordados dois cenários de uso: (i) um cenário direcionado a caracterização dos comportamentos dos diferentes procedimentos de busca; e (ii) outro cenário direcionado ao domínio da agricultura, tendo como base o projeto plenUS em desenvolvimento na Embrapa Clima Temperado. Os resultados atingidos com o cenário de uso se mostraram promissores, fornecendo um *feedback* para avanços futuros, o que estimula a continuidade das pesquisas na área.

## 6.2 Publicações Realizadas

Publicações como autor ou coautor relacionadas aos tópicos de pesquisa do EXEHDA-HS:

- EXEHDA-FC: An Architectural Approach Integrating Cloud Computing and Fog Computing; International Federation for Information Processing - IFIP; 2019; avaliação em andamento;
- Localização De Recursos Na IoT: Uma Proposta Hierárquica Direcionada À Organização Celular Do Exehda; Escola Regional de Redes de Computadores - ERRC; 2018;
- Uma Abordagem Dinâmica para Descoberta de Recursos na IoT Explorando Processamento Semântico. - 9º Simpósio Brasileiro de Computação Ubíqua e Pervasiva- SBCUP; 2017;
- EXEHDA-RR: Machine Learning and MCDA with Semantic Web in IoT Resources Classification; 23rd Brazillian Symposium on Multimedia and the Web - WebMedia; 2017;
- Uma Contribuição à Pesquisa Agropecuária; 43º Seminário Integrado de Software e Hardware - SEMISH; 2016;
- Uma arquitetura para IoT direcionada à ciência de contexto baseada em eventos distribuídos; 8º Simpósio Brasileiro de Computação Ubíqua e Pervasiva - SBCUP; 2016.

## 6.3 Trabalhos Futuros

Os resultados obtidos com o desenvolvimento desta dissertação de mestrado se mostraram oportunos, estimulando a continuidade dos esforços de pesquisa em relação ao EXEHDA-HS junto ao grupo. Dentre os aspectos levantados para continuidade do trabalho, destacam-se:

- Avaliar quantitativamente o desempenho das funcionalidades do EXEHDA-HS em situações de elevada escalabilidade;
- Dotar o EXEHDA-HS de suporte a outros protocolos de comunicação praticados na IoT;
- Prover o suporte a descoberta de recursos legados de fabricantes diversos;
- Avaliar as funcionalidades desenvolvidas para o EXEHDA-HS em outros cenários de uso;

- Considerar os diferentes quesitos de segurança associados as operações distribuídas de busca, quando da concepção de novas versões do EXEHDA-HS.

Além destes trabalhos futuros, artigos estão sendo escritos direcionados a congressos da área para divulgar os últimos resultados obtidos.

# REFERÊNCIAS

1SHEELD. **MQTT Protocol – How it Works**. https://1sheeld.com/mqtt-protocol/: 1Sheeld, 2019. Disponível em: < https://1sheeld.com/mqtt-protocol/ >. Acesso em Julho de 2019. Disponível em: <https://1sheeld.com/mqtt-protocol/>.

AHRENHOLZ, J. Comparison of CORE network emulation platforms. In: MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE, 2010., 2010. **Anais...** [S.l.: s.n.], 2010. p.166–171.

ARORA, R.; AGGARWAL, R. R. Modeling and querying data in mongodb. **International Journal of Scientific and Engineering Research**, [S.l.], v.4, n.7, p.141–144, 2013.

ASADI, A.; WANG, Q.; MANCUSO, V. A survey on device-to-device communication in cellular networks. **IEEE Communications Surveys & Tutorials**, [S.l.], v.16, n.4, p.1801–1819, 2014.

ASHTON, K. **That 'internet of things' thing in the real world, things matter more than ideas**. Disponível em: <http://www.r?djournal. com/article/print/4986 >. Acesso em dezembro de 2013.

AZEVEDO, M. M. de. **Descoberta de Recursos para o Middleware EXEHDA na Perspectiva da Internet das Coisas**. 2017. 79p. Dissertação (Mestrado em Ciência da Computação) — Universidade Catolica de Pelotas.

BALAKRISHNAN, H. et al. Looking up data in P2P systems. **Communications of the ACM**, [S.l.], v.46, n.2, p.43–48, 2003.

BANDYOPADHYAY, S.; SENGUPTA, M.; MAITI, S.; DUTTA, S. Role of middleware for internet of things: A study. **International Journal of Computer Science & Engineering Survey (IJCSES)**, USA, v.2, n.3, p.94–105, 2011.

BASSI, A. et al. **Enabling things to talk**. [S.l.]: Springer, 2016.

BENET, J. Ipfs-content addressed, versioned, p2p file system. **arXiv preprint arXiv:1407.3561**, [S.l.], 2014.

BIS Research. **Global Sensors in Internet of Things (IoT) Devices Market, Analysis & Forecast: 2016 to 2022**. [S.l.: s.n.], 2017.

BROCK, D. L. The Electronic Product Code (EPC): A Naming Scheme for Physical Objects. **Auto-ID Center White Paper WH-002**, [S.l.], 2001.

CARVALHO, F. O. **Descoberta Contínua de Serviços em IoT**. 2017. Tese (Doutorado em Ciência da Computação) — PUC-Rio.

CHAQFEH, M.; MOHAMED, N. et al. Challenges in middleware solutions for the internet of things. **Collaboration Technologies and Systems (CTS), International Conference on**, USA, p.21–26, 2012.

CHEN, Y.; LIU, B.; LU, Q.; ZHANG, H. A rapid evaluation of peers session length in Kademlia P2P networks. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER AND COMMUNICATIONS (ICCC), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.2402–2406.

CHOUDHARY, G.; JAIN, A. K. Internet of Things: A survey on architecture, technologies, protocols and challenges. **2016 International Conference on Recent Advances and Innovations in Engineering, ICRAIE 2016**, [S.l.], v.17, n.4, p.1–8, 2017.

CIRANI, S. et al. A scalable and self-configuring architecture for service discovery in the internet of things. **IEEE Internet of Things Journal**, [S.l.], v.1, n.5, p.508–521, 2014.

DAVET, P. T. **EXEHDA-IoT**: Uma Contribuição à Arquitetura do Middleware EXEHDA Direcionada à Internet das Coisas. 2016. Dissertação de Mestrado — Programa de Pós-Graduação em Computação - Universidade Federal de Pelotas.

DIAS, D.; BENET, J. Distributed web applications with ipfs, tutorial. In: INTERNATIONAL CONFERENCE ON WEB ENGINEERING, 2016. **Anais...** [S.l.: s.n.], 2016. p.616–619.

DILLI, R. et al. Fuzzy Logic and MCDA in IoT Resources Classification. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 33., 2018, New York, NY, USA. **Proceedings...** ACM, 2018. p.761–766. (SAC '18).

DILLI, R. M. **Uma Proposta para Descoberta de Recursos na Computação Ubíqua com Suporte Semântico**. 2010. 111p. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pelotas.

EJAZ, W.; ANPALAGAN, A. Internet of Things for Smart Cities: Overview and Key Challenges. In: **Internet of Things for Smart Cities**. [S.l.]: Springer, 2019. p.1–15.

FAHEEM, M.; SATTAR, H.; BAJWA, I. S.; AKBAR, W. Relational Database to Resource Description Framework and Its Schema. In: INTERNATIONAL CONFERENCE ON INTELLIGENT TECHNOLOGIES AND APPLICATIONS, 2018. **Anais...** [S.l.: s.n.], 2018. p.604–617.

GOMES, P. D. **Um serviço de descoberta ciente de contexto para internet das coisas**. 2016. Dissertação (Mestrado em Ciência da Computação) — Brasil.

JARA, A. J. et al. Mobile Digcovery: A global service discovery for the internet of things. **Proceedings - 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2013**, [S.l.], p.1325–1330, 2013.

JOAO, L. et al. Uma Contribuição á Ciência de Contexto no Middleware EXEHDA Explorando Fog Computing. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE 2018 (SEMISH 2018), 45., 2018. **Anais...** [S.l.: s.n.], 2018. v.45, n.1/2018.

JOAO, L. et al. **EXEHDA-FC**: An Architectural Approach Integrating Cloud Computing and Fog Computing. [S.l.]: International Federation for Information Processing - IFIP, 2019. Aguardando aprovação.

JOGUNOLA, O. et al. Comparative analysis of P2P architectures for energy trading and sharing. **Energies**, [S.l.], v.11, n.1, p.62, 2017.

KARIMI, M.; KHAYYAMBASHI, M. R. A Survey of web service discovery considering non-functional requirements. **IJCCER**, [S.l.], v.3, n.3, p.41–45, 2015.

KHAN, M. A. **Designing a Context-Aware Discovery Service for ioT Devices**. 2017. Master Thesis — University of Stuttgart. (0838).

KRITIKOS, K.; PLEXOUSAKIS, D. Towards combined functional and non-functional semantic service discovery. In: EUROPEAN CONFERENCE ON SERVICE-ORIENTED AND CLOUD COMPUTING, 2016. **Anais...** [S.l.: s.n.], 2016. p.102–117.

LEE, G. M.; KIM, J. Y. The Internet of Things?A problem statement. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY CONVERGENCE (ICTC), 2010., 2010. **Anais...** [S.l.: s.n.], 2010.

LIBP2P. **A modular network stack**. https://libp2p.io/: libp2p, 2019. Disponível em: $<$ https://libp2p.io/ $>$. Acesso em Julho de 2019. Disponível em: <https://libp2p.io/>.

LOPES, J. B. **Uma Arquitetura para Provimento de Ciência de Situação Direcionada às Aplicações Ubíquas na Infraestrutura da Internet das Coisas**. 2016. 123p. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

LOPES, J. et al. A Middleware Architecture for Dynamic Adaptation in Ubiquitous Computing. **j-jucs**, [S.l.], v.20, n.9, p.1327–1351, sep 2014.

LUA, E. K. et al. A survey and comparison of peer-to-peer overlay network schemes. **IEEE Communications Surveys & Tutorials**, [S.l.], v.7, n.2, p.72–93, 2005.

MALATRAS, A. State-of-the-art survey on P2P overlay networks in pervasive computing environments. **Journal of Network and Computer Applications**, [S.l.], v.55, p.1–23, 2015.

MAYMOUNKOV, P.; MAZIERES, D. Kademlia: A peer-to-peer information system based on the xor metric. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, 2002. **Anais...** [S.l.: s.n.], 2002. p.53–65.

MESHKOVA, E.; RIIHIJÄRVI, J.; PETROVA, M.; MÄHÖNEN, P. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. **Computer Networks**, [S.l.], v.52, n.11, p.2097–2128, 2008.

OCEGUEDA-MIRAMONTES, V.; SANCHEZ, M. A.; AGUILAR, L. Towards Intelligent Systems for Ubiquitous Computing: Tacit Knowledge-Inspired Ubicomp. In: **Applied Decision-Making**. [S.l.]: Springer, 2019. p.65–94.

OPUS. **Node.js – O que é, como funciona e quais as vantagens**. Disponível em: <https://www.opus-software.com.br/node-js/ >. Acesso em Fevereiro de 2019.

O'CONNOR, A.; BRADY, C.; BYRNE, P.; OLIVRÉ, A. Characterising the eDonkey peer-to-peer file sharing network. **Computer Science Department, Trinity College Dublin, Ireland, Tech. Rep**, [S.l.], 2004.

PAGANELLI, F.; PARLANTI, D. A DHT-based discovery service for the internet of things. **Journal of Computer Networks and Communications**, [S.l.], v.2012, 2012.

PEREIRA, C. R. **Aplicações web real-time com Node. js**. [S.l.]: Editora Casa do Código, 2014.

PIRES, P. F. et al. Plataformas para a Internet das Coisas. **Livro Texto de Minicursos - SBRC 2015**, Vitória - ES, 2015.

RATNASAMY, S. et al. **A scalable content-addressable network**. [S.l.]: ACM, 2001. v.31, n.4.

ROWSTRON, A.; DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: IFIP/ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS AND OPEN DISTRIBUTED PROCESSING, 2001. **Anais...** [S.l.: s.n.], 2001. p.329–350.

RUSLAN, R. et al. Routing performance of structured overlay in Distributed Hash Tables (DHT) for P2P. **Bulletin of Electrical Engineering and Informatics**, [S.l.], v.8, n.2, p.389–395, 2019.

SAROIU, S.; GUMMADI, P. K.; GRIBBLE, S. D. Measurement study of peer-to-peer file sharing systems. In: MULTIMEDIA COMPUTING AND NETWORKING 2002, 2001. **Anais...** [S.l.: s.n.], 2001. v.4673, p.156–171.

SAVANAH, S.; WRIGHT, C. S. **A method and system for verifying ownership of a digital asset using a distributed hash table and a peer-to-peer distributed ledger**. [S.l.]: Google Patents, 2019. US Patent App. 16/300,514.

SEZER, O. B.; DOGDU, E.; OZBAYOGLU, A. M. Context-Aware Computing, Learning, and Big Data in Internet of Things: A Survey. **IEEE Internet of Things Journal**, IEEE, v.5, n.1, p.1–27, Feb 2018.

SHEN, B.; GUO, J. Efficient Peer-to-Peer Content Sharing for Learning in Virtual Worlds. **arXiv preprint arXiv:1906.09438**, [S.l.], 2019.

SILVA, J. d. C. et al. Management Platforms and Protocols for Internet of Things: A Survey. **Sensors**, [S.l.], v.19, n.3, p.676, 2019.

SOLDATOS, J.; SERRANO, M.; HAUSWIRTH, M. Convergence of utility computing with the internet-of-things. In: INNOVATIVE MOBILE AND INTERNET SERVICES IN UBIQUITOUS COMPUTING (IMIS), 2012 SIXTH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** [S.l.: s.n.], 2012. p.874–879.

STOICA, I. et al. Chord: A scalable peer-to-peer lookup service for internet applications. **ACM SIGCOMM Computer Communication Review**, [S.l.], v.31, n.4, p.149–160, 2001.

STOLIKJ, M.; VERHOEVEN, R.; CUIJPERS, P. J.; LUKKIEN, J. J. Proxy support for service discovery using mDNS/DNS-SD in low power networks. In: PROCEEDING OF IEEE INTERNATIONAL SYMPOSIUM ON A WORLD OF WIRELESS, MOBILE AND MULTIMEDIA NETWORKS 2014, 2014. **Anais...** [S.l.: s.n.], 2014. p.1–6.

YASSEIN, M. B.; SHATNAWI, M. Q.; ALJWARNEH, S.; AL-HATMI, R. Internet of Things: Survey and open issues of MQTT protocol. In: INTERNATIONAL CONFE-

RENCE ON ENGINEERING & MIS (ICEMIS), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.1–6.

ZHAO, B. Y. et al. Tapestry: A resilient global-scale overlay for service deployment. **IEEE Journal on selected areas in communications**, [S.l.], v.22, n.1, p.41–53, 2004.