UNIVERSIDADE CATÓLICA DE PELOTAS
CENTRO DE CIÊNCIAS SOCIAIS E TECNOLÓGICAS
MESTRADO EM ENGENHARIA ELETRÔNICA E COMPUTAÇÃO

MAURÍCIO MADRUGA DE AZEVEDO

Descoberta de Recursos para o Middleware EXEHDA na Perspectiva da Internet das Coisas

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Engenharia Eletrônica e Computação

Orientador: Prof. Adenauer Correa Yamin

Pelotas
2017

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Azevedo, Maurício Madruga de

Descoberta de Recursos para o Middleware EXEHDA na Perspectiva da Internet das Coisas / Maurício Madruga de Azevedo. – Pelotas: 2017.

80 f.: il.

Dissertação (mestrado) – Universidade Católica de Pelotas. 2017. Orientador: Adenauer Correa Yamin.

I. Yamin, Adenauer Correa. II. Título.

“Ser feliz é encontrar força no perdão, esperanças nas batalhas, – segurança no palco do medo, amor nos desencontros. É agradecer a Deus a cada minuto pelo milagre da vida.” — AUGUSTO CURY

AGRADECIMENTOS

AGRADECIMENTOS

Agradeço primeiramente a Deus, que é o responsável pela nossa existência e é aquele que torna todos nossos sonhos possíveis de realizar. Sou profundamente grato ao acompanhamento de minha esposa Hélen durante todo o período do mestrado, sempre me incentivando e motivando a buscar sempre dar o melhor em todos momentos, também agradecendo a Deus por ter nos presenteado no período final com o nascimento de nossa filha Laura, iluminando ainda mais nossas vidas. Agradeço a todo suporte dado pelo Renato Dilli nesse período de pesquisa, cujos quais foram essenciais para o sucesso final deste trabalho. Também meus sinceros agradecimento ao Professor Adenauer Yamin, que com uso de seu vasto conhecimento e experiência esclareceu sempre todas dúvidas prontamente, e da melhor forma possível. Por fim, deixo meus agradecimentos a todos demais colegas que de forma indireta auxiliaram para que este projeto se tornasse possível com sucesso.

RESUMO

Temos presenciado dia-a-dia a evolução da Internet das Coisas, onde o número de objetos conectados vem tendo um crescimento constante. Estes objetos interoperam explorando diferentes recursos de comunicação e disponibilizam informações para uma ampla gama de serviços e aplicações. Este avanço na área da Internet das Coisas faz com que os recursos computacionais se tornem cada vez mais integrados às rotinas do dia-a-dia de seus usuários, facilitando a obtenção e o tratamento de informações de diferentes naturezas em áreas como agricultura, saúde, dentre outras. Este crescimento, dentre outros aspectos, introduz a necessidade dos recursos serem identificados, para que o usuário possa ter as informações que estão disponíveis no ambiente computacional de seu interesse. Neste sentido, as pesquisas para gerenciamento de recursos em ambientes computacionais, como os providos pela IoT, estão incorporando frentes de estudo buscando alternativas para que ocorra a identificação dos recursos da forma mais transparente possível, permitindo a descoberta e seleção de recursos disponibilizados nas infraestruturas computacionais modernas. O G3PD (Grupo de Pesquisa em Processamento Paralelo e Distribuído) vem articulando seus estudos relacionadas à IoT considerando um *middleware* para provimento de soluções nesta área, denominado EXEHDA (*Execution Environment for Highly Distributed Applications*). A presente dissertação tem por objetivo central a proposta de uma arquitetura para Descoberta de Recursos, direcionada ao cenário da Internet das Coisas, a ser integrada ao *middleware* EXEHDA. As principais funcionalidades da arquitetura proposta foram validadas por meio de prototipação e simulação, onde seus resultados se mostraram promissores fazendo uso de tecnologias e padrões voltados para a Internet das Coisas já utilizadas pelo G3PD, corroborando com os desafios de pesquisa na área de descoberta de recursos na perspectiva da Internet das Coisas.

Palavras-chave: Internet das Coisas. Ubicomp. Descoberta de Recursos.

EXEHDA-RD: Resource Discovery Mechanism at Internet of Things Perspective

ABSTRACT

We have witnessed the day-to-day evolution of the Internet of things, where the number of connected objects has had a steady growth. These objects interoperate exploring different communication features and provide information to a wide range of services and applications. This advance in the area of Internet of Things causes the computer resources become increasingly integrated into the routines of everyday life of its users, facilitating the collection and treatment of information of different natures in such areas as agriculture, health, among others. This growth, among other things, introduces the need of resources be identified, so that the user can have the information that is available in the computing environment of your interest. In this sense, the research for resource management in ubiquitous environments, such as those provided by the IoT, are incorporating study fronts seeking alternatives for resource identification occurs as transparent as possible, enabling the discovery and selection of resources available in the modern computing infrastructures. The G3PD (Group of research on Parallel and distributed Processing) has been articulating his studies related to IoT considering a *middleware* for providing solutions in this area, named EXEHDA (*Execution Environment for Highly Distributed Applications*). This dissertation aims to the central proposal of an architecture for Resource Discovery, directed to the scenario of the Internet of things, to be integrated into the *middleware* EXEHDA. The main features of the proposed architecture have been validated through prototyping and simulation, where their results turned out to be promising using technologies and Internet standards of Things already used by G3PD, corroborating research challenges in the area of resource discovery in the context of the Internet of Things.

Keywords: Internet of Things, Ubicomp, Resource Discovery.

LISTA DE ABREVIATURAS E SIGLAS

CoAP	*Constrained Application Protocol*
CORE	*Constrained RESTful Environment*
CRUD	*Create, Read, Update and Delete*
DA	*Directory Agent*
DGT	*Distributed Geographic Table*
DHCP	*Dynamic Host Configuration Protocol*
DLS	*Distributed Location Service*
DPWS	*Device Profile for Web Services*
DTLS	*Datagram Transport Layer Security*
EXEHDA	*Execution Environment for Highly Distributed Applications*
G3PD	Grupo de Pesquisa em Processamento Paralelo e Distribuído
GLs	*Group Leaders*
GM	*Group Members*
HTTP	*Hypertext Transfer Protocol*
IERC	*IoT European Research Cluster*
IETF	*Internet Engineering Task Force*
IoT	*Internet of Things*
IP	*Internet Protocol*
ITU	*International Telecommunication Union*
JSON	*JavaScript Object Notation*
MQTT	*Message Queue Telemetry Transport*
OCF	*Open Connectivity Foundation*

P2P	*Peer-to-Peer*
RDF	*Resource Description Framework*
REST	*Representational State Transfer*
RRA	*Repositório de Recursos Ativos*
RRN	*Repositório de Recursos Negados*
SAs	*Service Agents*
SOAP	*Simple Object Access Protocol*
SPARQL	*SPARQL Protocol and RDF Query Language*
SSDP	*Simple Service Discovery Protocol*
TLS	*Transport Layer Security*
UAs	*User Agents*
UPnP	*Universal Plug and Play*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
WSDL	*Web Service Description Language*
XML	*eXtensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Quando empregamos o termo *Internet das Coisas*, estamos nos referindo a um cenário onde a Internet e o mundo estão cada vez mais conectados entre si, com objetos compartilhando serviços e informações remotamente. Levando em consideração as características relacionadas a Internet das Coisas, destacamos a capacidade dos objetos proverem serviços por meio da Internet (MATTERN; FLOERKEMEIER, 2010). Esta visão contribui para materializar as afirmações feitas em 1999 por *Weiser*, o qual em seus trabalhos já citava que a computação em sua evolução se tornaria cada vez mais ubíqua, estando disponível em todo local e a qualquer momento (WEISER, 1999).

O termo IoT (do inglês *Internet of Things*) é definido pela *International Telecommunication Union* (ITU) e pela *IoT European Research Cluster* (IERC), entidades de pesquisas de padrões e tecnologias relacionadas à comunicação, como uma infraestrutura de rede global com capacidade de auto configuração baseadas em protocolos interoperáveis onde *coisas* possuem atributos físicos e/ou virtuais e fazem uso de interfaces inteligentes tornando-se perfeitamente integrados às redes de informações globais (VERMESAN; FRIESS, 2014).

Estimam-se resultados cada vez mais promissores envolvendo o cenário da Internet das Coisas, destacando-se frentes de pesquisas como a Computação em Nuvem, Internet do Futuro, *Big Data*, Robótica e o emprego de Tecnologias Semânticas (PERERA et al., 2013).

O G3PD (Grupo de Pesquisa em Processamento Paralelo e Distribuído) da UCPEL tem como o centro de suas pesquisas o *middleware* EXEHDA, aprimorando constantemente sua arquitetura a fim de desenvolver soluções direcionadas ao Processamento Paralelo e Distribuído. Este *middleware* é responsável pela criação e manutenção de um ambiente computacional, bem como pela gerência da execução de aplicações sobre o mesmo (YAMIN, 2004).

A Descoberta de Recursos é um requisito importante para modelagem de arquitetura em um ambiente computacional na perspectiva da Internet das Coisas, pois a presença destes recursos se dá de uma forma dinâmica, necessitando o mínimo de interferência do usuário nesse processo. Garantir o controle de estado destes recursos, segura situações onde erros podem acarretar sérios problemas como perda de plantações na área da agricultura ou erros médicos na área da saúde.

O EXEHDA-RD foi implementado para incorporar no *middleware* EXEHDA um serviço de Descoberta de Recursos, o qual deverá contemplar as características operacionais pertinentes aos modernos cenários computacionais providos pela IoT.

Assim, o presente trabalho tem como tema central a concepção de uma arquitetura que

atenda aos requisitos necessários para a Descoberta de Recursos nos cenários computacionais modernos, nos quais os dispositivos tem uma elevada dinâmica quanto à sua disponibilidade, uma vez que em contraste com demandas dos usuários, a qualquer momento podem ter seu estado de conexão alterado com relação a infraestrutura computacional.

1.1 Motivação e Objetivos

A presença de recursos em um ambiente computacional, capacita à arquitetura obter informações que podem ser utilizadas por várias áreas, como agricultura e saúde. Por meio de aplicações, essas áreas realizam acessos a serviços identificados e disponibilizados pela arquitetura (MATTERN; FLOERKEMEIER, 2010).

Estes recursos, em sua maioria possuem características semelhantes, podem recursos diferentes proverem os mesmos serviços, sendo de grande importância o uso de ontologias nesse contexto a fim de obter um maior resultado em resultados de consultas, garantido uma maior gama de resultados nessas buscas.

Os recursos computacionais geralmente não tem suas informações disponibilizadas de forma automática ao ingressarem em um determinado ambiente computacional, portanto, um Serviço de Descoberta de Recursos é essencial para que os recursos possam ser localizados e utilizados pelas aplicações, exigindo a menor intervenção possível dos usuários.

O número de soluções que exploram a Internet das Coisas tem crescido continuamente, tanto por meio de aplicações *Open Sources*, quanto soluções comerciais (MINERAUD et al., 2015). Este constante esforço de pesquisa em busca de soluções e também a necessidade de Protocolos e Padrões específicos principiam as motivações centrais da presente dissertação.

Considerando estes conceitos, esta dissertação tem como objetivo geral propor um Arquitetura de Descoberta de Recursos para o *middleware* EXEHDA na perspectiva da Internet das Coisas. Com base na proposta, o *middleware* EXEHDA fez uso de tecnologias e padrões relacionados com Internet das Coisas, considerando que sensores e atuadores se encontram cada vez mais dependentes de uma arquitetura moderna para utilização de seus serviços.

Em razão desta necessidade, define-se como objetivos específicos do trabalho os seguintes tópicos:

- Sistematizar informações sobre Serviços de Descoberta de Recursos, organizando conceitos e identificando trabalhos relacionados na área;
- Identificar padrões e tecnologias adequadas ao Cenário da IoT;

- Prover um mecanismo para gerenciar e manter o estado dos recursos presentes no ambiente computacional gerenciado pelo *middleware* EXEHDA;
- Estruturar uma arquitetura na perspectiva da Internet das Coisas para o *middleware* EXEHDA, denominado EXEHDA-RD;
- Definir as diferentes tecnologias integram a proposta do EXEHDA-RD;
- Modelar e avaliar as funcionalidades do EXEHDA-RD;
- Validar o serviço através da simulação da presença de vários recursos no ambiente computacional gerenciado pelo *middleware* EXEHDA.

1.2 Estrutura do Texto

O texto é composto por seis capítulos. O primeiro capítulo apresenta o tema, as motivações e objetivos propostos para o presente trabalho.

No segundo capítulo são apresentados conceitos relacionados ao trabalho proposto, englobando os protocolos e padrões utilizados no Cenário da IoT, revisão teórica sobre o tema de Descoberta de Recursos. São discutidas também ferramentas usadas para construção e processamento de ontologias. É apresentado o *middleware* EXEHDA, núcleo de estudo do G3PD, destacando a contribuição do presente trabalho para o mesmo.

No terceiro capítulo alguns trabalhos considerados relevantes para comparativos e análise de tecnologias utilizadas, foram citados e discutidos, a fim de selecionar padrões e tecnologias que se adaptem à modelagem proposta e que esteja de acordo com as já utilizadas pelo G3PD, tornando possível a implementação da arquitetura no ambiente computacional gerenciado pelo *middleware* EXEHDA, possibilitando a continuidade de estudos baseando-se no conteúdo da presente dissertação.

No quarto capitulo é apresentada a Modelagem Proposta, características essenciais para sua eficiência, o que engloba todas as tecnologias envolvidas, a metodologia aplicada ao Serviço de Gerenciamento e Manutenção dos Recursos e definição do formato utilizado para troca e envio de mensagens entre os componentes presentes na arquitetura.

No quinto capítulo são apresentados três casos simulados que serão utilizados para validar e avaliar os requisitos funcionais do modelo proposto. Cenários simulados em ambientes computacional com linguagens de programação e tecnologias já utilizadas pelo G3PD no ambiente computacional gerenciado pelo *middleware* EXEHDA.

O sexto capítulo descreve as considerações finais do trabalho, apresentando as conclusões alcançadas após as realizações de testes na arquitetura, seus resultados e discussões. Foram definidos também os trabalhos futuros a serem estudados e pesquisados, baseados nas premissas e características presentes no trabalho apresentado.

2 FUNDAMENTAÇÃO CONCEITUAL

Este capítulo tem como finalidade caracterizar as frentes que compõem o escopo de estudo desenvolvido nessa dissertação de mestrado. A primeira seção apresenta o estado da arte da Descoberta de Recursos.Na segunda e terceira seções, são apresentados conceitos baseados nas necessidades presentes em trabalhos direcionados a Internet das Coisas. A quarta seção apresenta conceitos relacionadas às ferramentas e tecnologias necessárias para o trabalho com ontologias. O capítulo é finalizado com a seção descrevendo o *middleware* EXEHDA, dando destaque o serviços presentes no *middleware* que possuem relação direta com o escopo deste trabalho.

2.1 Descoberta de Recursos

Seção que explora os conceitos da descoberta de recursos, suas características, vantagens e desvantagens.

2.1.1 Características Gerais

O Serviço de Descoberta é responsável pela comunicação entre vários dispositivos, a forma como interagem entre si, com computadores e com o ser humano (KINDBERG; FOX, 2002). UPnP é um conjunto de protocolos que permite que recursos se comuniquem entre si e anunciem sua presença em um determinado ambiente computacional, geralmente havendo uma arquitetura modelada para realizar o controle de entrada e saída destes recursos.

De acordo com (Marin Perianu; Hartel; Scholten, 2005) há duas entidades ativas em um serviço de descoberta, o cliente, que é aquele interessado em acessar o serviço e buscar as informações, e o servidor, que se responsabiliza por manter e oferecer os recursos. Muitas vezes há a existência de um intermediador de recursos, havendo desta forma um ganho de desempenho.

Para o serviço de descoberta, as seguintes propriedades são importantes, conforme afirma (INGMARSSON, 2013):

- Independência da rede em uma camada de alto nível;
- Em rede *Peer-to-Peer* (P2P), é importante a existência de de um componente intermediário que sirva como mediador;

- Recursos devem ser flexíveis e expressivos;
- O serviço deve ser autossustentável, capaz de tomar decisões de forma automatizada e pró ativa.

2.1.2 Fundamentos e Requisitos

Um serviço de descoberta engloba informações necessárias para dispositivos se encontrarem e intercomunicarem entre si, e este serviço pode ser disponibilizado tanto em dispositivos remotos ou embarcados, conforme necessidade, afirma (OVIDIU, 2011).

Um serviço de descoberta tem como objetivos principais:

- **Descoberta Dinâmica:** um serviço tem como objetivo principal encontrar recursos de acordo com um conjunto de propriedades requisitadas. A partir daí os Protocolos de Descobertas de Recursos precisam fazer uso de uma linguagem para descrever esses recursos, uma forma de armazenamento destes dados e uma forma para busca destes recursos;
- **AutoConfigurável:** ele necessita ser autoconfigurável de acordo com o ambiente computacional em que se encontra presente.

Conforme cita (Marin Perianu; Hartel; Scholten, 2005), o protocolo identifica a forma como entidades devem se comunicarem entre si, possuindo participação ativa o Cliente, que possui o interesse de busca de informações de dispositivos de um determinado contexto. O Servidor por sua vez provê esse recurso ao cliente.

Os serviços de descobertas podem ser realizados em sistemas que trabalhem de forma centralizada, onde um nó contém todas informações de recursos e descentralizada, onde vários nós interligados entre si compartilham informações de recursos, comumente usado onde existe uma grande escala de recursos.

De acordo com (BROWN, 2008), um mecanismo de descoberta de recursos deve seguir alguns requisitos para se tornar eficiente, estando entre eles: escalabilidade, tolerância a falhas, eficiência, compreensível sistema de busca, segurança e funcionalidades extensíveis, que são descritas com detalhes a seguir:

- **Escalabilidade**: se caracteriza pela preocupação em não perder desempenho no provedor de recursos à medida que há um aumento do volume de dados, mais encontrado

em sistemas centralizados, pois dispositivos descentralizados aumentam seus nós à medida que incrementa os dados, havendo uma distribuição de informações equalizadas entre eles;

- **Tolerância a Falhas**: 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 crucial para o bom funcionamento de todo o sistema;
- **Garantia de Resultados de Busca**: a arquitetura precisa sempre garantir que o serviço que seja buscado pelo cliente seja provido, sendo este um erro presente em arquiteturas descentralizadas ondem não há uma boa intercomunicação entre todos os nós;
- **Eficiência**: deve ser garantido que as requisições realizadas não sobrecarreguem o sistema sobre um todo, podendo causar desta forma uma parada total do mesmo;
- **Segurança**: todo e qualquer sistema que armazena informações valoráveis, deve garantir que estas informações sejam acessadas apenas para aqueles que às solicitou;
- **Nível de Carga de recursos**: é comum se preocupar com o nível de carga de carregamento de recursos, erro mais comum em arquiteturas centralizadas onde não há uma equalização de importância. Já o mesmo não se verifica em arquiteturas descentralizadas pelo fato de ser equalizada as responsabilidades entre os servidores.

2.1.3 Etapas do Processo de Descoberta de recursos

Um Processo de Descoberta de recursos passa por vários estágios, que são discutidos a seguir.

Descrição do Serviço

Se faz necessária a existência de um meio de identificação de recursos para os mesmos serem descobertos em um ambiente computacional. Sendo esta descoberta algo essencial para um Processo de Descoberta de recursos se tornar bem-sucedido. O formato como os recursos são identificados é uma particularidade que varia de acordo com o protocolo que está sendo

usado e o tipo de serviço que se deseja descobrir (DOBREV et al., 2002). De forma geral são atributos essenciais um identificador e um tipo de serviço, sendo os demais atributos opcionais.

Estes recursos normalmente são classificados em tipo por meio de classes, que por sua vez contem duplas de atributos que identificam estes recursos e diferem um do outro (Marin Perianu; Hartel; Scholten, 2005).

Esta convenção de nomes normalmente se correlaciona com a descrição do serviço. Há, todavia, a definição de convenção de nomes para atributos por vários protocolos, reduzindo o conflito de nomes (TEAM, 2009)

Disponibilização e Registro de recursos

Para os recursos serem disponibilizados aos clientes eles devem ser anunciados no ambiente computacional e terem suas informações armazenadas em algum tipo de base de dados. Isto pode ser realizado de mais de uma maneira, onde alguns recursos armazenam todas informações em único local, enquanto em outros cenários, algumas destas informações são disponibilizadas em vários nós na rede, a fim de otimizar resultados de buscas pela limitação da quantidade de recursos armazenados em cada nó, garantindo sua eficiência mesmo que haja um crescimento na quantidade de recursos.

Em arquiteturas de nós de redes distribuídos o processo inicia com a distribuição de recursos entre os vários nós. Na medida que o cliente solicita um determinado serviço, o nó que estiver mais próximo dele disponibiliza o recurso, caso contrário ele possui indexado informações que determinam qual nó mais próximo possui determinado recurso, considerando que ele não encontrou resultados satisfatórios à consulta. Estas operações são realizadas via broadcast, ou seja, parte de um nó para os demais nós presentes no ambiente.

Descoberta do Recursos

O Estágio da Descoberta é onde os recursos são retornados de acordo com determinados critérios de busca. Como premissa básica para que determinado recurso seja encontrado, o mesmo precisa estar registrado. A forma como estes recursos são encontrados dependem diretamente da forma que eles são registrados.

Esta descoberta pode ser realizada de forma passiva ou ativa. (BARTLETT; HEIDEMANN; PAPADOPOULOS, 2007) descreve como forma ativa, àquela onde um servidor faz

o envio para todos os nós e fica aguardando sua resposta. Para otimizar o Serviço de Descoberta, o servidor pode identificar quais nós se encontram presentes, não realizando requisições para àqueles que estiverem ausentes. Ao contrário da forma ativa, na forma passiva existem identificadores de estado nos nós clientes, que rodam em background, não sendo necessária a necessidade de envio de requisição de resposta por parte do servidor para saber o estado de seus nós clientes.

Seleção de Recurso

Em razão de vários recursos poderem satisfazer determinada condição de busca em uma requisição, é necessário a existência de um Serviço responsável pela Seleção do Serviço que mais se adapta às necessidades do cliente. Serviço este que pode ser realizado de forma manual pelo usuário ou por meio de algoritmos de otimização implementados nos nós onde são distribuídos os recursos no lado do servidor.

Conforme caracterizado por (VERVERIDIS; POLYZOS, 2008), protocolos automáticos realizam a seleção de recursos baseados em critérios e métricas específicas. As métricas avaliam diversos fatores dinâmicos, onde podemos citar como exemplo a localização do recurso e caracterização dos recursos.

Para que uma seleção de recursos seja realizada de uma forma eficiente, se faz necessário o conhecimento dos recursos presentes, definindo técnicas que capacitem à arquitetura retornar resultados pertinentes às aplicações.

Invocação do Recurso

Como estágio final do Processo de Descoberta de recursos, temos a invocação do serviço. Ela se realiza logo após o cliente obter o identificador URI do recurso requisitado, que determina como deve ser acessado os serviços disponibilizados por tal recurso. Para inicialização de invocação do serviço se faz necessária uma interface de serviço. Cada protocolo apresenta características distintas, quanto a forma de realizar a invocação dos serviços disponibilizados pelos recurso. (ZHU; MUTKA; NI, 2005), subdivide em três, a quantidade de níveis de suporte a serviços disponibilizados pelos protocolos:

- **Primeiro Nível**: protocolos apenas distribuem os recursos, ficando as aplicações responsáveis pelo trabalho de comunicação e manipulação dos recursos;

- **Segundo Nível**: no segundo nível os protocolos definem o mecanismo de comunicação entre os recursos. Por exemplo, mecanismos de comunicação UPnP e *Web Service Description Language* (WSDL) são baseados em *eXtensible Markup Language* (XML), *Simple Object Access Protocol* (SOAP) e protocolos *Hypertext Transfer Protocol* (HTTP) (CHRISTENSEN et al., 2001);

- **Terceiro Nível**: no terceiro nível, em adição aos mecanismos de comunicação são determinadas operações específicas para as aplicações que se comunicam com recursos.

2.1.4 Redes Confinadas

Redes confinadas se caracterizam pela presença de dispositivos com recursos restritos, devendo a partir deste ponto serem consideradas algumas constantes:

- **Escalabilidade:** solução capaz de gerenciar e controlar descoberta, mantendo um rastreamento dos recursos ativos, considerando uma ampla margem na flutuação do total de recursos envolvidos;

- **Tamanho Compacto:** capacidade de implementação em dispositivos com capacidade de recursos restritos;

- **Comunicação Compacta:** preocupação com o tamanho reduzido de banda em redes composta de dispositivos com recursos restritos, por meio de fragmentação de pacotes, permitindo seu envio por toda rede com menos consumo desta banda;

- **Nós em estado de espera:** existências de nós com capacidade de entrarem em modo de espera para redução de gasto de energia, por meio de algoritmos RDC, algoritmo responsável por sincronização de conteúdo capaz de realizar comparação entre informações e processar apenas informações novas;

- **Interoperabilidade:** serviço de descoberta sobre uma aplicação que seja possível utilizar em diversas redes, em razão da heterogeneidade de dispositivos presentes no ambiente contextual;

- **Eficiência de Energia:** envio apenas de informações essenciais, reduzindo sobrecarga de rede com informações não pertinentes para a Descoberta e Invocação de recursos para o Usuário final;

- **Reconhecimento de Recursos:** não se pode dar prioridade apenas para dispositivos ricos em recursos. Todavia o protocolo deve ser eficiente ao ponto de ter como fator de ranqueamento importante o reconhecimento de dispositivos com maior riqueza de recursos em um determinado conjunto;

- **Composição de Recursos:** recursos descobertos podem ser invocados juntos em determinadas situações. Nestes casos o Serviço de Seleção e Descoberta deve ter a capacidade de agrupamento de recursos diferentes com o intuito de oferecer novos recursos.

2.2 Internet das Coisas

O termo "Internet das Coisas"foi utilizado pela primeira vez por Kevin Ashton (MUKHOPADHYAY; SURYADEVARA, 2014). A idéia de dispositivos conectados e compartilhando informações na Internet é alvo antigo de estudos, sendo os termos Computação Ubíqua os que mais fazem referência a esta linha de pesquisa na literatura (LUETH, 2017).

A Internet das Coisas, do inglês *Internet of Things* (IoT), é um paradigma caracterizado por um cenário onde qualquer "coisa", desde objetos do nosso dia a dia como também pessoas e animais, podem se comunicar pela Internet. Esta comunicação é possibilitada por meio de dispositivos computacionais embarcados às "coisas"que as capacitam a capturar variáveis ambientais, realizar variados processamentos e reagir a estímulos externos, como também interoperarem com outros dispositivos via Internet com o mínimo de interação humana (PIRES et al., 2015).

A IoT tem despertado interesse mundial devido ao crescente número de dispositivos que vem sendo interconectados à Internet, gerando novas informações que podem agregar conhecimento para os mais diferentes tipos de aplicações. Por ser uma área relativamente nova, ainda possui desafios de pesquisa, tais como definição de tecnologias e padrões específicos, a serem endereçados para a sua plena materialização, o que estimula o envolvimento da comunidade científica no tema.

Há vários conceitos chaves relacionados à Internet das Coisas segundo (MUKHOPADHYAY; SURYADEVARA, 2014):

- **Redes de Sensores**: define o conjunto de sensores e atuadores interconectados distribuídos a fim de monitorar condições físicas e ambientais e compartilhando os dados por meio da Internet;

- **Indústria da internet das Coisas**: termo inserido pela General Electrics, previamente conhecido como M2M, visto que a comunicação inclui também interfaces humanos;

- **Internet**: conceito mais amplo relacionado à forma global de comunicação mundial entre as pessoas;

- **Web das Coisas**: termo relacionado a arquitetura de software que relaciona protocolos e tecnologias Web com a forma de conectar sensores e atuadores para disponibilizar seus dados globalmente;

- **Internet de Tudo**: internet de Tudo é um termo mais amplo que engloba o conceito da Internet das Coisas, visto que defende a premissa de que todos dispositivos e pessoas conectados entre si são capazes de realizar tomadas de decisões;

- **Indústria 4.0**: são processos industriais que realizam pesquisas relacionadas às novas tecnologias, capacitando os ambientes computacionais a fim de que atendam aos requisitos operacionais da Internet das Coisas.

2.3 Padrão UPnP

Recursos precisam se comunicar entre si, bem como os dados coletados por estes dispositivos precisam ser enviados a um servidor e por sua vez servidores devem se comunicar entre si para publicar estes dados, tornando-os visíveis para dispositivos móveis, aplicações e para pessoas.

Para que esta intercomunicação exista e ocorra de uma forma correta, devem ser seguidos um conjunto de padrões definidos pelo ITU-GSI e outras organizações responsáveis em definir padrões de comunicação. Esta seção resume as características do padrão UPnP empregado na interoperação entre Gateways e o EXEHDABorda.

O padrão UPnP é um conjunto de protocolos de rede voltados para descoberta de recursos desenvolvido pela Microsoft para habilitar o reconhecimento, descoberta e controle de dispositivos e serviços em rede por meio de um conjunto de protocolos. A *Open Connectivity Foundation* (OCF) é uma das entidades atuais que tem apresentado estudos de evoluções relacionadas à conectividade de dispositivos no Cenário da IoT. esta rede realiza sua comunicação por meio de endereços *Internet Protocol* (IP) e faz uso de protocolos padrões tais como HTTP,

XML e SOAP para descoberta, descrição e controle de dispositivos. Podem ser definidos como dispositivos, entidades em redes que contém serviços ou qualquer dispositivo embarcado.

O Processo de descoberta de um dispositivo que segue o padrão UPnP, se dá da seguinte forma:

- **Endereçamento**: no momento que um dispositivo se conecta em rede ele tenta obter um IP, por meio de *Dynamic Host Configuration Protocol* (DHCP) ou é determinado de forma automática. É feita uma verificação se este IP não está ocupado, logo após sua obtenção;

- **Descrição**: os detalhes de configuração dos dispositivose suas características são definidos em arquivos no formato XML;

- **Descoberta**: o serviço de descoberta é baseado no protocolo *Simple Service Discovery Protocol* (SSDP). Este protocolo permite que este dispositivo anuncie sua descoberta e descubra outros serviços e dispositivos. Sua mensagem é associada a um tempo de vida que contém o tipo de anúncio e a *Uniform Resource Locator* (URL) de descrição do serviço;

- **Controle**: com a utilização da URL provida pelo serviço, é possível realizar uma troca de mensagens com o recurso a fim de realizar o controle de seu estado;

- **Eventos**: a existência de eventos possibilita que o controle seja notificado quanto à modificação do estado do recurso.

2.4 Padrão Arquitetural REST

A presente seção demonstra o estilo arquitetural *Representational State Transfer* (REST), direcionado a sistemas distribuídos, descrevendo sua arquitetura de software, bem como as premissas de sua modelagem, em contraste com as características de outros estilos arquiteturais.

REST se trata de um estilo híbrido derivado da combinação de características de várias arquiteturas, para capacitá-la como uma interface de conexão uniforme partindo de estilos nulos onde os componentes são iniciados sem a existência de um limite entre suas comunicações.

Em razão de possuir padrões comuns à várias tecnologias, ele se torna essencial para a construção de uma arquitetura direcionada para a descoberta de recursos.

2.4.1 Visão Geral

REST não é um protocolo, mas sim uma arquitetura. Foi primeiramente introduzido por (FIELDING, 2000) em 2000, e vem sido usado amplamente desde então em sistemas distribuídos.

Assim como outras modelagens arquiteturais, o estilo REST foi estruturado partindo de um ponto inicial zero, por meio da união de constantes necessárias para tornar a arquitetura funcional de modo geral.

REST foi um termo definido por Roy Thomas Fielding, em sua tese de doutorado (FIELDING, 2000), onde ele modela um estilo de arquitetura para construção de *web services* consistentes e coesos. O estilo de arquitetura REST é baseado em recursos e nos estados desses recursos.

O modelo arquitetural REST é implementado utilizando o protocolo HTTP e utiliza *Uniform Resource Identifier* (URI), que se trata de de URL para nomear recursos, para realizar a comunicação entre cliente e servidor. Sistemas modernos são complexos e exigem que suas funcionalidades sejam distribuídas em módulos e que haja uma forma concreta de comunicação entre estes módulos. Outra questão relevante que se observa é que uma aplicação, em sua maioria, não obedece às restrições impostas pelas arquiteturas, onde a arquitetura deve ser maleável com o intuito de simplificar a sua comunicação com a aplicação.

O REST tem como premissa principal se tornar um modelo global a partir de estudos e validações de incompatibilidades existentes em arquiteturas anteriores, substituindo-as por funcionalidades genéricas.

O estilo arquitetural REST apresenta as seguintes características:

- Estrutura arquitetural modelada a partir de estilos arquiteturais existentes e um conjunto consistente de terminologias de definição de sua estrutura;
- Arquitetura voltada para sistemas distribuídos;
- Aplicação do estilos arquitetural REST, visando aplicações modernas.

2.4.2 Elementos da Arquitetura REST

Não há definições concretas de arquitetura de software de âmbito geral, em contraste defende-se que uma arquitetura de software se trata de um conjunto de elementos em tempo

de execução se intercomunicando entre si a fim de tornar um modelo arquitetura consistente (FIELDING, 2000);

Uma arquitetura é composta por um conjunto de componentes, conectores e dados, onde cada um deles desempenha um papel específico dentro deste conjunto:

- **Componentes:** são unidades abstratas dentro de um modelo arquitetura responsável em realizar transformações em dados. São exemplos de transformações, modificar o formato de um dado, mesclar dados, etc;
- **Conectores:** são mecanismos responsáveis em realizar a comunicação e transferência de dados entre componentes. Como exemplos destes mecanismos, podemos citar protocolos de transferência de mensagens, compartilhamento de informações e fluxos de dados;
- **Dados:** é um elemento contento um conjunto e informações que são transferidos pelos componentes por meio dos conectores.

O Comportamento e a forma como os componentes se comunicam entre si são definidos por meio da **configuração** da arquitetura, enquanto o papel funcional de cada componente e como ele é organizado são definidos pelas **propriedades** da arquitetura.

Estilos em arquitetura, definem as regras e restrições que aplicações devem seguir ao fazer o uso da arquitetura. Uma arquitetura bem modelada deve possuir também padrões bem definidos, bem como uma forma de comunicação entre componentes coesas, havendo a especificação de como são realizadas e as linguagens utilizadas em sua concepção.

2.4.3 Características do REST

A arquitetura REST, como arquitetura híbrida, é composta por um conjunto de características particulares que descrevem uma arquitetura.

- **Cliente-Servidor:** a adoção do estilo arquitetural Cliente-servidor tem por objetivo separar as responsabilidades entre cliente e servidor, onde o servidor fica responsável pelas operações de manipulação e armazenagem de dados, enquanto o cliente fica responsável pela criação de interfaces para integração com o aplicações. Este princípio capacita a arquitetura para cumprir o requisito de escalabilidade na Internet em múltiplos domínios organizacionais;

- **Sem estado:** esta característica determina que o servidor não armazena informações do cliente, onde o mesmo realiza o envio todas informações necessárias para o entendimento das requisições ao servidor. Isto induz a uma Viabilidade e Escalabilidade. Pode haver uma desvantagem em relação a esta característica, pelo fato de que pode haver uma sobrecarga na rede quando há uma série de requisições com grande volume de informações;

- **Cache:** o sistema de cacheamento reflete em uma maior eficiência na utilização da rede, pois quando uma requisição é rotulada como cacheável, ela permite que seus dados sejam armazenados no servidor para uso em requisições equivalentes. Redução esta perceptível ao usuário final, no momento que há uma menor latência nas séries de iterações realizadas;

- **Interface Uniforme:** é a característica que distingue o estilo arquitetura REST de outros estilos baseados em rede por meio uma interface uniforme entre os componentes. A construção das interfaces entre os componentes seguem as seguintes premissas: identificação de recursos, manipulação de recursos através de representações, mensagens auto descritivas, e *hypermedia* como o motor de controle de estado da aplicação;

- **Sistema Modular:** a existência de sistema modular, permite que a arquitetura seja composta por camadas hierárquicas onde os componentes só iteram com os dados de seu interesse em um operação realizada entre cliente e servidor. A combinação de um sistema modular com uma interface uniforme permite que componentes individuais sejam configurados de uma forma particular, tornando possível sua manipulação sem necessidade de observar a arquitetura inteira;

- **Código Sobre Demanda:** característica relacionada com a escalabilidade de aplicações REST, que permite que as funcionalidades dos clientes sejam estendidas por meio de download e execução de códigos em forma de *applets* ou *scripts*.

2.4.4 Web Service RESTful

RESTful é um *Web Service* baseado no padrão arquitetural REST. A forma mais utilizada de transmissão e recebimento de mensagens se dá através dos métodos usados pelo protocolo HTTP: GET, POST, PUT e DELETE, comumente utilizado em aplicações CRUD. Devido ser

um formato global de descrição de mensagens, o JSON é o formato padrão escolhido para descrição das mesmas.

Entidades como a IETF e OMA têm investido esforço de estudos na busca de padrões a serem utilizados na construção de *Web Services* RESTful (BELQASMI; GLITHO; FU, 2011). A ITU-T prevê uma nova geração de redes, onde além de dados, serão compartilhadas também informações por voz e também multimídia denominada NGN, defendendo a necessidade da existência de um padrão no trasporte de dados via web.

JSON

De acordo com (RICHARDSON; RUBY, 2007), JSON é o formato mais indicado para utilização em *Web Services* REST, devido à sua ampla utilização em aplicações de diversas naturezas.

De acordo com (ECMA, 2013), JSON se trata de um formato de texto estruturado criado para facilitar a troca de mensagens entre todas as linguagens de programação. Sua sintaxe é composta de chaves, colchetes, ponto e vírgula e aspas, tornando-o simplificado para todas linguagens. Um valor JSON pode ser *object*, *array*, *number*, *string*, *true*, *false*, ou *null*, com a peculiaridade dos valores numéricos serem na base decimal, tornando entendível por todas linguagens de forma simples.

2.5 Ontologias: Ferramentas, Padrões e Linguagens

Ontologia é um modelo de dados que representa um conjunto de conceitos compartilhados. Ontologias são utilizadas para compartilhar informações de um domínio, composto de um vocabulário bem definido e com um entendimento comum e não ambíguo dos termos e conceitos utilizados pelas aplicações. Os elementos que formam uma ontologia são definidos pela tripla: Classes, Atributos e Relacionamentos. Um grande benefício no uso de ontologias é a possibilidade de descrever os relacionamentos entre os objetos. O conjunto destes relacionamentos, chamado de semântica, facilita às máquinas o entendimento de conceitos que não estão declarados de forma explícita. Em razão disso, pode-se obter uma maior expressividade em resultados de buscas por recursos do que quando se utilizando uma base de dados relacionais, que não consideram a inferência de conhecimento em suas consultas. Gruber (GRUBER, 1993) cita quatro componentes necessários para a especificação de uma ontologia: classes, relações,

funções e axiomas, definindo-os da seguinte forma:

- **Classes**: organização das ontologias por axiomas, que podem ser definidos como a forma de classificar e identificar os termos de uma ontologia;
- **Relações**: são interações entre os conceitos e domínios, sendo exemplos "subclasse de" e "conectado a";
- **Funções**: estruturas mais complexas para serem utilizadas na substituição de um termo ou instrução;
- **Axiomas**: sentenças formais que possuem resultados lógicos sempre verdadeiros.

2.5.1 Raciocinadores

Raciocinadores se tratam de componentes de softwares responsáveis por inferir conhecimento lógico a partir de um conjunto de fatos ou axiomas. As regras de inferências são especificadas por meio de ontologias. Temos avaliados na API Jena, de possível utilização pelo Apache Fuseki, os seguintes:

- ***Transitive Reasoner***: permite percorrer classes e suas propriedades, acessando apenas as propriedades transitivas e reflexivas;
- ***RDF Rule Reasoner***: permite implementar um subconjunto configurável das ocorrências RDFs;
- ***OWL, OWL Mini, OWL Micro Reasoners***: conjunto incompleto suportado pela linguagem OWL-Lite;
- ***Generic rule reasoner***: raciocinador que permite regras definidas pelo usuário com suporte de encadeamento e técnicas de execuções híbridas.

2.5.2 Padrão RDF/RDF-Schema

Resource Description Framework (RDF) é um padrão recomendado pela W3C para descrição de recursos Web, primeiramente citado em 2004. Projetado para ser interpretado por computadores por meio de escrita em XML, não construído para ser visualizado por pessoas. A

comunicação entre objetos descritos por meio de RDF e a Web se dá por meio de URIs (Uniform Resources Identifiers), sendo URIs considerado também a forma principal de acesso aos serviços providos por uma modelagem arquitetural.

Os seguintes objetos complementam RDF:

- **Recursos**: é qualquer coisa que tem uma URI, por exemplo, "http://www.w3schools.com/rdf";
- **Propriedades**: é um Recurso que tem um nome, por exemplo "homepage";
- **Literais**: é o valor da propriedade, por exemplo "http://www.w3schools.com";
- **Declaração**: é a declaração de um recurso mais as propriedades desse recurso e o valor dessas propriedades.

2.5.3 Linguagem OWL

A OWL é uma linguagem recomendada pelo W3C para representação de ontologias na Web Semântica. Projetada para representar categorias de objetos e as relações entre eles, além de informações sobre os próprios objetos (HORROCKS; PATEL-SCHNEIDER; HARMELEN, 2003). Linguagem amplamente usada para representação de ontologias em diversos domínios como medicina, biologia, geografia, astronomia.

Sua proposta surgiu baseando nas seguintes restrições:

- Ser compatível com o padrão RDF, estendendo a capacidade de expressar o conhecimento dito "ontológico";
- Possuir sintaxe e semântica bem definidas, com um poder de expressividade, mantendo propriedades computacionais desejáveis (ANTONIOU; HARMELEN, 2009).

2.5.4 Linguagem SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) refere-se a uma linguagem de consulta e protocolo de acesso a dados em RDF, recomendada pela W3C (PRUD'HOMMEAUX; SEABORNE, 2008).

A linguagem SPARQL possibilita que as aplicações acessem dados representados em RDF de uma forma semelhante ao modo como se utiliza a linguagem SQL (Structured Query

Language) para consultar dados em bases de dados relacionais, abstraindo as complexidades envolvidas nas consultas.

2.6 O Middleware EXEHDA

Esta seção é destinada a apresentar os conceitos relacionados ao *middleware* EXEHDA e seus subsistemas, citado inicialmente em (YAMIN, 2004).

O EXEHDA é conceituado como um*middleware* adaptativo ao contexto e baseado em serviços, com objetivo de criação e gerenciamento de um ambiente computacional, provendo alternativas de execução neste ambiente. Estas aplicações são distribuídas, móveis e adaptativas ao contexto em que seu processamento ocorre, estando disponíveis a partir de qualquer lugar, todo o tempo (LOPES et al., 2012).

2.6.1 Arquitetura e Funcionalidades

O *middleware* EXEHDA tem como premissa principal a definição de uma arquitetura que capacite aplicações se comunicarem em um ambiente computacional, de uma forma sensível ao contexto gerindo as informações, sendo seus aspectos funcionais e não-funcionais mutáveis a partir deste controle. Aplicações alvos do *middleware* EXEHDA são distribuídas, sensíveis ao contexto e abrangem a mobilidade lógica e física. Mobilidade física e lógica neste cenário, refere-se à mobilidade de dispositivos enquanto a mobilidade física se refere ao deslocamento do usuário (LOPES et al., 2014).

- **Arquitetura de Software:** a arquitetura de software do *middleware* EXEHDA é apresentada na Figura 2.1. Os principais requisitos que o EXEHDA deve atender são: a. gerenciar tanto aspectos não funcionais como funcionais da aplicação, de modo independente; b. dar suporte à adaptação dinâmica de aplicações; c. disponibilizar mecanismos para obter e tratar informações de contexto; d. empregar informações de contexto na tomada de decisões; e. decidir as ações adaptativas de forma colaborativa com a aplicação; e f. 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;

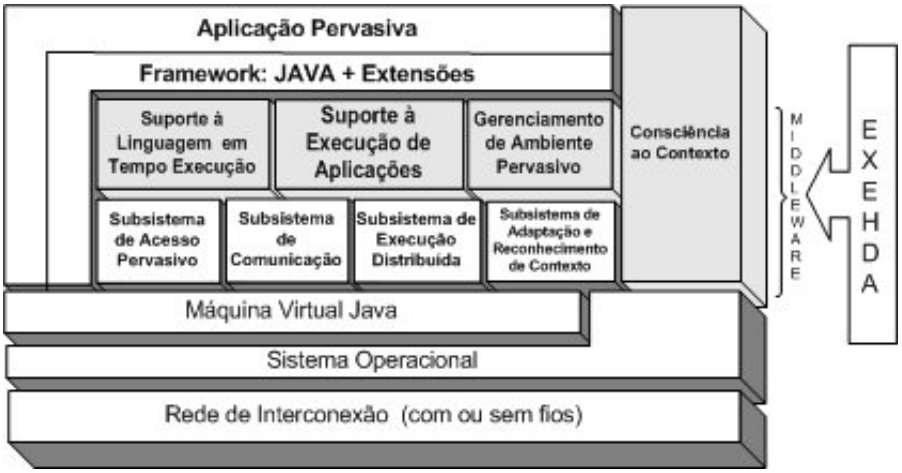

Figura 2.1: Arquitetura de Software do *middleware* EXEHDA
Fonte: (LOPES et al., 2014)

- **Ambiente Ubíquo Disponibilizado:** o ambiente ubíquo se equivale ao ambiente computacional, visto que recursos e serviços gerenciados pelo EXEHDA tem como premissa cumprir os requisitos propostos em um ambiente computacional. Sistema este, composto por todos dispositivos e usuários relacionados à sua execução no *middleware*.

  Os recursos desta infraestrutura são mapeados em três abstrações básicas, empregadas na composição do ambiente computacional (LOPES et al., 2014):

  - **EXEHDAcels**: indica a área de atuação de uma EXEHDAbase, e é composta por esta célula e por EXEHDAnodos. Os principais aspectos considerados na definição da abrangência de uma célula são: o escopo institucional, a proximidade geográfica e o custo de comunicação;
  - **EXEHDAbase**: é o ponto de convergência para os EXEHDAnodos. É responsável por todos os serviços básicos do ambiente computacional e, embora constitua uma referência lógica única, seus serviços, sobretudo por aspectos de escalabilidade, poderão estar distribuídos entre os vários dispositivos;
  - **EXEHDAnodo**: são os dispositivos de processamento disponíveis no ambiente computacional, sendo responsáveis pela execução das aplicações. Um subcaso deste tipo de recurso é o EXEHDAnodo móvel. São os nodos do sistema com elevada portabilidade, tipicamente dotados de interface de rede para operação sem fio e, neste caso, integram a célula a qual seu ponto-de-acesso está subordinado. São funcionalmente análogos aos

EXEHDAnodos, porém eventualmente com uma capacidade mais limitada (por exemplo, PDAs).

- **Composição de Serviços:** a existência de dispositivos amplamente heterogêneos em um ambiente computacional, torna necessária a adoção de um núcleo com funcionalidades mínimas, estendidas através de serviços carregados sob demanda. Esta abordagem para padronização de projetos é referenciada na literatura como *microkernel*.

Sistemas modernos em ambientes computacionais 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 Computação Ubíqua.

2.7 Emulador CORE

O *Constrained RESTful Environment* (CORE) se trata de um conjunto de ferramentas utilizadas para emulação de infraestruturas computacionais distribuídas, fruto de pesquisa do laboratório de pesquisas navais do EUA. Ele é executado em plataformas Open Source como FreeBSD e/ou Linux. Sua emulação e virtualização podem ser realizadas tanto por interface gráfica, quanto com a utilização da linguagem de programação Python através de uma interface de texto (CORE, 2015).

A existência da interface gráfica capacita a ferramenta para a criação de topologias de uma forma simples, permitindo que sejam construindo ambientes emulados, possíveis serem conectados com redes reais para simulações de diversas naturezas (AHRENHOLZ, 2010).

Através da interface gráfica temos acesso a um conjunto de recursos (computadores, hubs, switches, roteadores) dentre outros componentes que podem compor uma infraestrutura computacional, podendo ser utilizados livremente. Cada um desses itens pode ser configurado individualmente, para endereçamento IP, limite de banda, dentre outros parâmetros de configuração. Ao ser acionado o botão *Start* da interface, a topologia criada é instanciada e é possível inicializar o Prompt de comando a partir de um duplo clique em qualquer um dos itens presentes na topologia (AHRENHOLZ et al., 2008).

A vasta gama de funcionalidades providas pelo ferramenta, possibilitou que o emulador CORE fosse utilizado para criação de um ambiente virtual, a fim de avaliar as funcionalidades da arquitetura do EXEHDA-RD, simulando um ambiente computacional por meio da criação de uma topologia.

2.8 Considerações Sobre o Capítulo

Este capítulo apresentou a fundamentação teórica sobre a Internet das Coisas, seus padrões e protocolos; Descoberta de Recursos; contextualização de Redes Confinadas; conceitos, ferramentas, padrões e linguagens para construção e processamento de ontologias, bem como apresentado o EXEHDA, *middleware* responsável por prover o ambiente computacional necessário para a concepção do EXEHDA-RD e um resumo do emulador CORE, essencial para o desenvolvimento dos cenários apresentados a fim de validar as funcionalidades da arquitetura proposta. Esta fundamentação conceitual é necessária para a compreensão dos próximos capítulos.

3 TRABALHOS RELACIONADOS

A literatura registra trabalhos de pesquisa em descoberta de recursos, tais como Jini, OSGi [OSGi Alliance, 2010], SLP [The Internet Engineering Task Force, 1999], UPnP, Salutation, *Device Profile for Web Services* (DPWS) etc.

Apesar de alguns destes trabalhos estarem em evolução, os mesmos apresentam características que podem comprometer seu emprego no cenário da IoT. Dentre estas características destacaríamos: o atrelamento ao emprego de uma linguagem específica de programação, descrições dos recursos unicamente sintáticas, funcionamento apenas em redes locais, etc.

Tendo por base uma revisão de literatura na área de Descoberta de Recursos que identificou os principais trabalhos na área, este capítulo apresenta três trabalhos que foram selecionados tendo como critério sua similaridade com o EXEHDA-RD.

O estudo destes trabalhos foi central para concepção do EXEHDA-RD, contribuindo significativamente para sistematização de sua proposta de modelagem arquitetural.

3.1 Trend-based Service Discovery Protocol

O TRENDY é o acrônimo para Trend-based Service Discovery Protocol, que tem por premissa principal a criação de um Protocolo de descoberta de recursos, baseando-se em tendências dos usuários, empregadas por um algoritmo de classificação e seleção de recursos otimizado.

A arquitetura apresentada por (BUTT et al., 2013), foi modelada usando web services RESTful, considerando um agrupamento de recursos com características semelhantes e permanência de consultas a recursos em cache, seguindo premissas identificadas por estudos que apontam características consideradas essenciais às aplicações voltadas para a Internet das Coisas.

Arquitetura do TRENDY

A presente seção apresenta os vários aspectos relacionados à arquitetura utilizada na solução TRENDY.

A Figura 3.1 mostra a arquitetura híbrida que foi proposta pela solução. A solução mantém um registro através do Diretório*Directory Agent* (DA), e os Recursos *Service Agents*

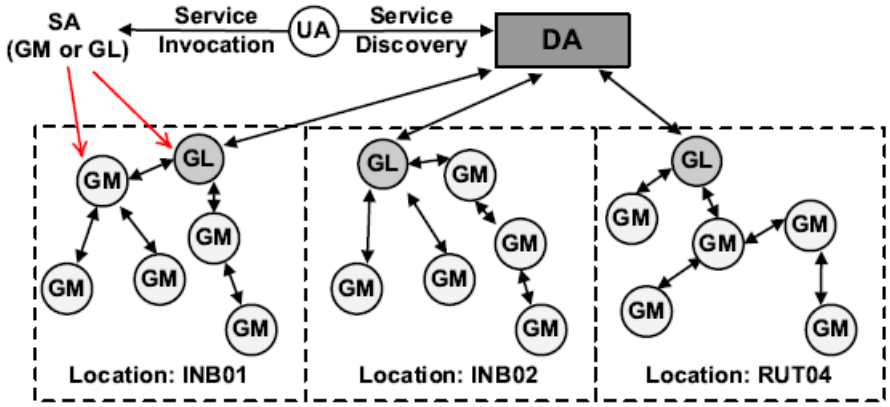

Figura 3.1: Arquitetura do TRENDY
Fonte: (BUTT et al., 2013)

(SAs) por sua vez fazem seu armazenamento. Clientes *User Agents* (UAs) consultam o DA, para localizar o serviço. Após os mecanismos de agrupamento categorizarem os SAs em Líderes *Group Leaders* (GLs) e Membros *Group Members* (GM).

- **Diretório:** dentro da arquitetura do TRENDY, diretórios mantém um registro e fazem uso de um temporizador para aumentar ou diminuir o tempo de atraso entre as atualizações de manutenção. Sua tendência é prover o serviço ideal ao usuário levando em conta as informações contextuais;

- **Membros:** representam os recursos resultantes de uma determinada consulta ao diretório cruzando informações contextuais e atributos enviados;

- **Líderes:** em um mecanismo de agrupamento, um líder possui uma função principal. O modelo do TRENDY permite que os líderes selecionem diferences grupos de recursos de acordo com os recursos disponíveis ou necessidades da aplicação. Suas responsabilidades dependem do conjunto de regras definidos pelo administrador, podendo apenas ser responsável pela manutenção de estado, enviar uma consulta para outro nó, agrupar os resultados, ser apenas um registro local ou atuar com um *gateway*;

- **Clientes:** denomina os clientes que descobrem os recursos baseado em consultas aos Diretórios. Pode fazer parte de um grupo de recursos ou ser algum objeto aleatório realizando uma requisição de qualquer lugar por meio da Internet.

Características do TRENDY

Nesta seção, são detalhadas as características do TRENDY, conforme descrito por (BUTT et al., 2013):

- **WebServices:** o TRENDY tem sua comunicação realizada por meio de protocolos COAP ou HTTP, sendo este utilizado quando o diretório de recursos está atuando como uma camada de Proxy;
- **Consciência do Contexto:** no TRENDY, os diretórios mantêm armazenados os recursos e todas informações contextuais, sendo por sua vez ranqueados os que possuem informações contextuais de cada consulta específica;
- **Agrupamentos:** agrupamentos baseado em contexto servem para vários propósitos, desde agrupar por localização ou manutenção de estado, oferecendo assim a otimização e melhora de resultados oferecidos;
- **Arquiteturas Híbridas:** a arquitetura do TRENDY converge para uma arquitetura distribuída no momento em os diretórios fazem uso das informações contextuais para agrupar os Recursos;
- **Descrição de Recursos:** considerando redes confinadas, as descrições de recursos são separadas apenas por virgulas, pela sua compactação;
- **Gerenciamento de Recursos:** todos recursos no TRENDY são mantidos com estado inativos, sendo o intervalo de manutenção e verificação de estado de cada um controlado por meio de um temporizador que avalia a demanda de cada um destes recursos;
- **Descoberta de Recursos:** a descoberta de recursos se dá por meio da combinação de atributos descritos nas URLs de consulta, retornando àqueles que satisfaça algum desses atributos de consulta;
- **Seleção de Recursos:** para a seleção de recursos é realizado o cruzamento dos recursos com maior número de casualidades com àqueles que possuam informações contextuais de rede, como localização, bateria, dentre outros;
- **Invocação de Recursos:** o serviço de descoberta se faz completo no momento que a aplicação obtém a resposta da consulta contendo a localização destes recursos, possibilando ao TRENDY invocar os serviços por meio de webservice RESTful.

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

A Internet das Coisas tem como objetivo principal conectar diversos dispositivos, promovendo novas oportunidades e formas de interações entre pessoas e coisas (CIRANI et al., 2014). Com isso é explorado o conceito de objetos inteligentes, que tem como principal premissa diminuir a intervenção de humanos em suas configurações e manutenções de estado.

Tendo como motivação a escalabilidade desses dispositivos, o presente trabalho apresenta uma alternativa de arquitetura autoconfigurável baseado em P2P, provendo recursos automatizados e mecanismos de descoberta de recursos.

Presença de Gateway IoT na Arquitetura

A arquitetura proposta apresenta Gateways IoT responsáveis em anunciar dispositivos, nós com descoberta de recursos, proxies, cacheamento e funcionalidade de controle de acesso.

São apresentadas a seguir um resumo das funcionalidades presentes nos Gateways IoT:

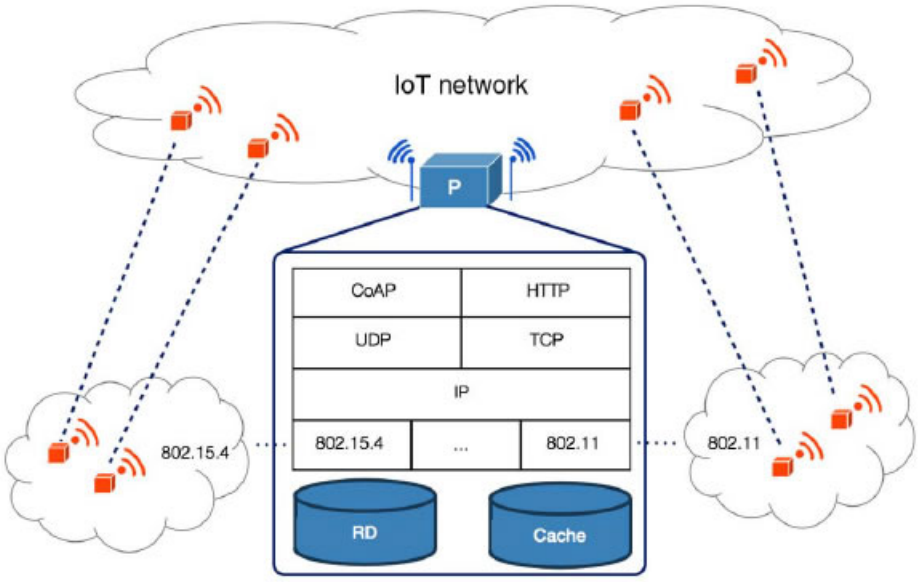

Figura 3.2: Arquitetura de um Gateway IOT
Fonte: (CIRANI et al., 2014)

- **Funcionalidade de Proxy**: os *gateways* interagem em nível de aplicação com demais nós IoT, através de um protocolo CoAP, podendo atuar como servidor ou cliente. Algumas razões defendem a presença de proxies em nós de borda em uma rede IoT: proteção de ataques externos contra redes confinadas; integração web por meio de protocolo HTTP; redução de carga de rede em redes confinadas; suporte para formatos de dados globais, como XML ou JSON. A Figura 3.2 representa sua arquitetura. De um ponto de vista arquitetural o Gateway IoT se compõe dos seguintes elementos: Gateway IP IPv4/IPv6 para suporte de redes heterogêneas, Servidor COAP e proxy reverso HTTP-CoAP que disponiliza serviços e recursos em uma rede confinada;

- **Descoberta de Serviços e Recursos**: a camada de serviço busca as portas de servidores CoAP em rede, enquanto a camada de descoberta permite descobrir dispositivos gerenciados por um servidor CoAP.

Arquitetura baseada em P2P

As infraestruturas para IoT podem ser disponibilizadas sobre o protocolo P2P, provendo um mecanismo de descoberta de recursos em larga escala. Dessa forma disponibilizando as seguintes funcionalidades: Escalabilidade; Alta Variabilidade; e Autoconfiguração. O presente trabalho é organizado considerando duas arquiteturas do protocolo P2P: *Distributed Location Service* (DLS) e *Distributed Geographic Table* (DGT), descritos a seguir:

- **DLS**: arquitetura baseada em DHT, que disponibiliza um serviço de resolução de nomes baseados em dados recuperados por meio de URI. Sua funcionalidade principal é prover informações de localização, que podem vir junto com outras informações, tais como tempo de expiração, prioridade de acesso, etc;

- **DGT**: *schema* que provê informações de localização dos nós, permitindo localização de nós próximos à determinada posição geográfica conforme pode ser visto na Figura 3.3.

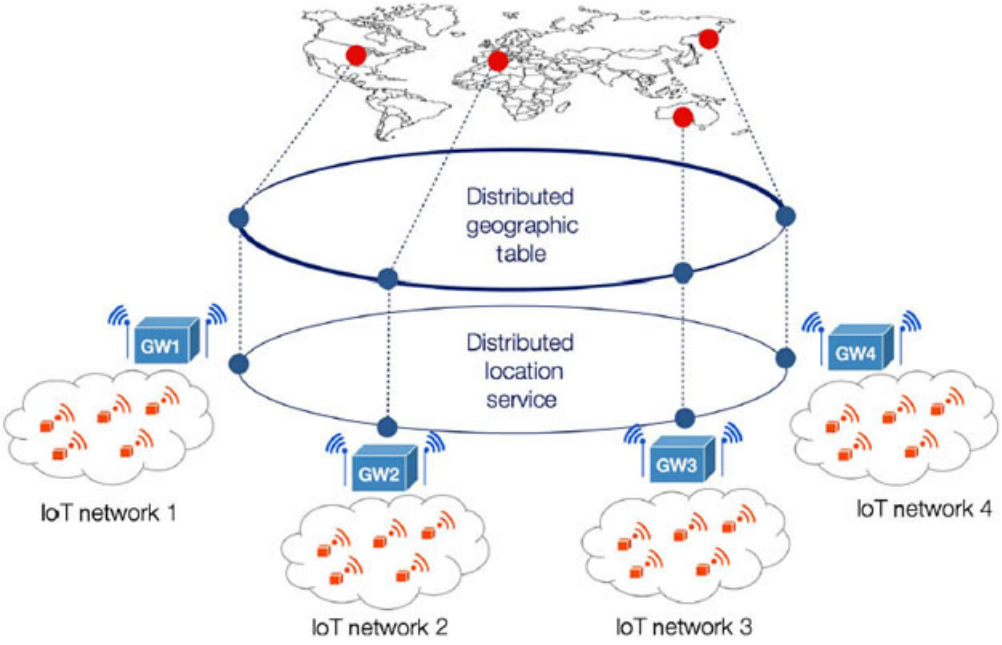

Figura 3.3: Arquitetura de Serviços em Alta Escala
Fonte: (CIRANI et al., 2014)

3.3 Web of Things: Automatic Publishing and Configuration of Devices

Em paralelo com o crescimento da computação embarcada, aumenta a quantidade de dispositivos físicos com capacidade de se conectar de forma inteligente, tornando-se visíveis na internet. A IoT se faz mais importante tendo como premissa conectar estes dispositivos na Internet por meio do protocolo IP. Em contrapartida há desafios quanto à forma de comunicação pelo fato da existência de uma grande heterogeneidade de dispositivos (JUNIOR; BASTOS; PRAZERES, 2014).

O presente trabalho defende o uso de protocolos abertos da Web (HTTP, SOAP, HTML, XML, JSON, etc) para desenvolvimento de uma arquitetura para dispositivos que possuam os mais diferentes hardwares e softwares, havendo desta forma uma maneira comum de realizar a comunicação entre todos eles.

Sua arquitetura, como apresentada na Figura 3.4, pode ser descrita da seguinte forma:

- **Communication**: prover comunicação automática por meio de um conjunto de protocolos, e em casos de dispositivos smartphones, possibilitar que o usuário especifique que tipo de serviço deseja compartilhar (sensor, GPS, etc...);

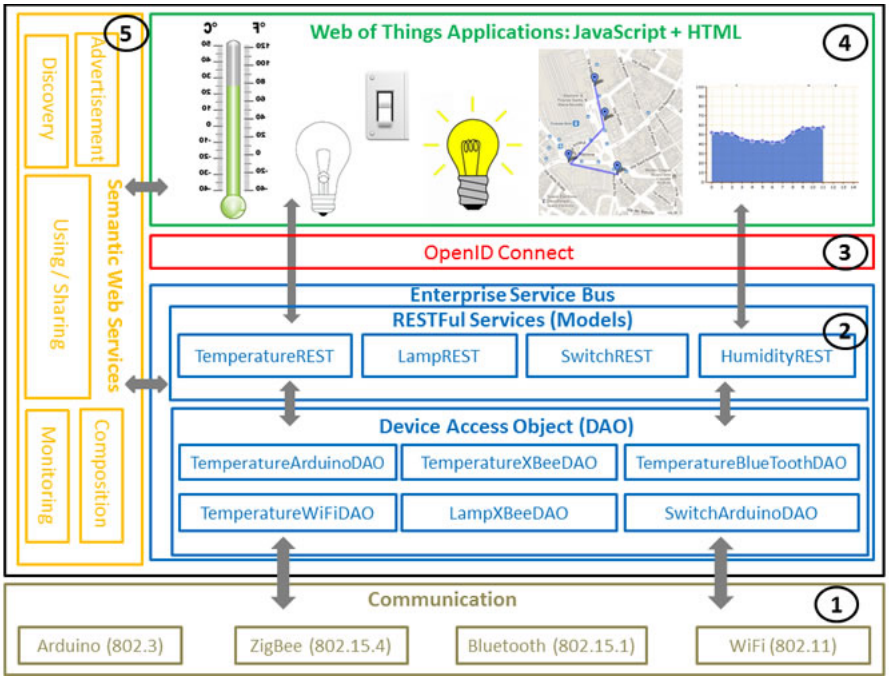

Figura 3.4: Arquitetura para a Web das Coisas
Fonte: (JUNIOR; BASTOS; PRAZERES, 2014)

- **Entreprise Service Bus - ESB**: *middleware* orientado a serviços facilitadores para integração, havendo modelos de acessos baseados no meio de comunicação do dispositivo;
- **OpenID Connect**: componente para possibilitar compartilhamento de dispositivos na arquitetura;
- **Web of Things Applications**: interfaces visíveis ao usuário, que se comunicam com a arquitetura;
- **Semantic Web Services**: serviço semântico de automação de tarefas da arquitetura.

3.4 Discussão dos Trabalhos Relacionados

Em razão do crescimento da IoT, existe uma preocupação nestes trabalhos com padrões e tecnologias em uma perspectiva voltada para a Internet das Coisas. Percebe-se que em sua grande maioria, há um foco em questões de protocolos a serem utilizados para comunicação dos componentes a fim de tornar possível sua utilização na perspectiva da web.

Percebe-se que não há um foco no controle da entrada de recursos por parte desses trabalhos, enquanto o EXEHDA-RD tem como premissas principais a existência de componentes arquiteturais concebidos para realizar um controle melhor da presença de recursos no ambiente

computacional gerenciado pelo EXEHDA. Também é empregado processamento semântico na consulta de recursos a fim de prover uma melhor expressividade nos resultados.

Esta seção discute abordagens utilizadas pelos trabalhos relacionados envolvendo o mecanismo de consultas utilizado e a estratégia empregada para manutenção e controle da presença de recursos. Na discussão são priorizados os aspectos de expressividade e gerenciamento da presença de recursos, por estes serem aspectos centrais na proposta do EXEHDA-RD.

3.4.1 Expressividade de Resultados de Buscas

Uma arquitetura modelada visando uma maior expressividade na descrição e consulta de seus dados, torna possível que usuários tenham acessos a recursos com caraterísticas similares aos critérios principais de uma busca, e não apenas os que condizem diretamente com os critérios de busca.

Esta seção analisa a expressividade em cada um dos trabalhos relacionados, e como eles modelam sua arquitetura nessa perspectiva.

- **TRENDY:** realiza agrupamento de recursos baseado em localização, onde um dos recursos desenvolve o papel de gerente do grupo, realizando sub-consultas, onde a complexidade das consultas são afetadas diretamente pelo poder computacional do recurso que atua com gerente do grupo de recursos;

- **Scalable and Self Configuring Archicteture:** não há preocupação com expressividades, pois são realizadas apenas consultas do método GET para cada nó, retornando os serviços presentes e disponíveis em cada um destes nós;

- **Web of Things:** questões relacionadas à expressividade não são consideradas no presente trabalho relacionado;

- **EXEHDA-RD:** suas consultas são realizadas em SPARQL através do Apache Fuseki com raciocinadores ativados do módulo Processador Semântico, o que permite serem criadas regras dinâmicas de similaridade para buscas de recursos no ambiente computacional. Como o servidor está localizado em um nodo EXEHDABase, suas consultas podem ser complexas por não se tratar de um dispositivo embarcado, com limitações computacionais.

3.4.2 Gerenciamento da Presença de Recursos

Como a presença de um recurso em um ambiente computacional na perspectiva da Internet das Coisas possui um amplo dinamismo, toda e qualquer solução se faz de uma valia maior no momento em que a mesma possui um controle eficaz da entrada e saída de recursos em um ambiente computacional.

Nesta seção, vemos como é visto esse requisito essencial quando se tratando de soluções para Internet das Coisas, bem como cada trabalho aborda essa caraterística.

- **TRENDY:** cada recurso possui um leasing adaptativo que aumenta no momento em que este recurso vem se tornado menos utilizado. Desta forma, um recurso pode retornar em resultado de buscas e estar inativo, caso tenha um leasing muito grande;

- **Scalable and Self Configuring Archicteture:** há presença de Gateways a fim de capacitar recursos que operam com protocolos diferentes se comunicarem entre si de forma homogênea, onde o Gateway neste caso atua com uma ponto de comunicação entre os recursos. Os Gateways capturam informações dos recursos para armazenálas e tornar apto o acesso aos recursos pelas aplicações de diversas naturezas;

- **Web of Things:** neste trabalho relacionado, a descoberta de recursos se dá pelo conjunto de componentes utilizando funcionalidades Zeroconf com Bonjour, onde o recurso se conecta diretamente à arquitetura. Uma vez conectado e identificado o recurso, caso ele esteja presente no Servidor de Modelos, um serviço só se torna disponível caso existam modelos na arquitetura para trabalhar com o recurso conectado, e não existe uma forma de controle de leasing dos recursos;

- **EXEHDA-RD:** todo e qualquer recurso que se faz presente na arquitetura, ao anunciar sua presença, a arquitetura captura seu UUID, bem como as características presentes em seu XML, mantendo seu estado atualizado na ontologia. O paradigma da arquitetura que diz que todo e qualquer recurso só pode ser acessado quando não estiver negado, possibilita que recursos indesejáveis não estejam conectados e presentes no ambiente computacional gerenciado pelo EXEHDA, não sendo possível seu uso por aplicações de diversas naturezas.

3.5 Considerações Sobre o Capítulo

Este capítulo apresentou três trabalhos relacionados sobre descoberta de recursos para a Internet das Coisas. Todos eles utilizam padrões e protocolos adequados à provável limitação de hardware e software dos dispositivos conectados na IoT. Há também a preocupação com a detecção dos recursos com a mínima ou nenhuma intervenção humana. Estas características definidas nos trabalhos relacionados, bem como a dinamicidade em que os recursos entram e saem do ambiente computacional e a escalabilidade são algumas das premissas que estão sendo consideradas no modelo proposto.

Tabela 3.1: Comparação de Aspectos entre Propostas para IoT

	TRENDY	Scalable Archictecture	Web of Things	EXEHDA-RD
Expressividade	Agrupamento de Recursos	Não	Não	OWL
Controle de Presença	Leasing	Gateways	Bonjour	Leasing e Controle

A tabela 3.1 apresenta a comparação entre os trabalhos relacionados e o modelo proposto. Em resumo o modelo proposto oferece uma maior expressividade em consultas e maior efetividade em gerenciamento e manutenção de presença de recursos em um ambiente computacional, comparado aos trabalhos relacionados.

4 EXEHDA-RD: VISÃO GERAL E ARQUITETURA

A Descoberta de Recursos em infraestruturas modernas tem como requisito considerar que a presença de recursos se dá de uma forma dinâmica. Neste sentido, está sendo proposto um modelo semântico de descoberta de recursos capaz de localizar e disponibilizar acesso aos recursos às aplicações usuárias do *middleware* EXEHDA.

Em uma perspectiva mais ampla, este trabalho tem por meta contribuir com os trabalhos de pesquisa realizados pelo Grupo de Pesquisa em Processamento Paralelo e Distribuído (G3PD) referentes ao EXEHDA (DILLI, 2010), (DAVET, 2016). Particularmente, o EXEHDA-RD irá contribuir com a redefinição dos aspectos da arquitetura do *middleware* necessários ao tratamento da descoberta de recursos na IoT.

4.1 Características da Modelagem

- **Reorganização da arquitetura de software**: remodelação da presente arquitetura com o objetivo de capacitá-la para trabalhar com Protocolos e Padrões relacionados à Internet das Coisas;
- **Padronização de mensagens de comunicação de componentes**: usar padrões REST como a forma padrão de comunicação padrão entre os componentes e usar JSON como a forma padrão de descrever as mensagens trocadas entre os mesmos, para haver uma maior interoperabilidade entre todos os componentes da arquitetura;
- **Controle do estado de recursos de uma forma dinâmica**: o controle de estado se torna um requisito essencial, visto que em um ambiente computacional com a presença de sensores e atuadores, estes recursos tendem a entrar e sair do ambiente de forma dinâmica, a fim de que demais funcionalidades da arquitetura se tornam eficientes na perspectiva da Internet das Coisas;
- **Emprego de protocolos e padrões IoT**: o emprego de protocolos e padrões se faz necessário, levando em consideração esforços de pesquisas que corroboram e defendem um conjunto de padrões e tecnologias quando tratando de Internet das Coisas, a fim de tornar a arquitetura flexível, maleável e expansível;
- **Otimização do Processamento Semântico**: a adoção do Apache Fuseki como a ferramenta para tratamento de tecnologias, invés do uso do framework Apache Jena,

desacopla da arquitetura a necessidade de ter um componente específico na linguagem Java para tratamento de Ontologias, também pelo fato de que o Apache Fuseki se trata de um servidor SPARQL que já possui os requisitos para tratamento de Ontologias em conformidade do fato de fazer uso de padrões REST na troca de mensagens de seus componentes.

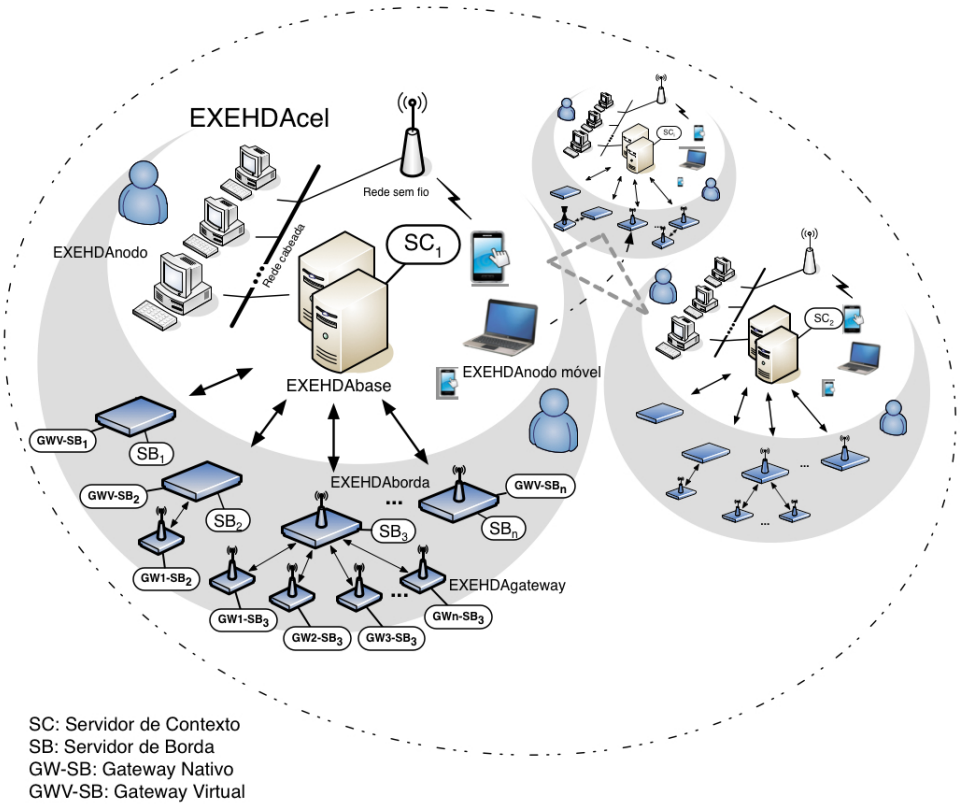

Figura 4.1: Organização Celular do Ambiente Computacional Gerenciado pelo EXEHDA-RD
Adaptado de: (DAVET, 2016)

Para que se torne possível a descoberta de recursos relacionados à Internet das Coisas de uma forma eficiente, se faz necessária a utilização de protocolos e tecnologias que estudos consideram essenciais para modelagem de arquiteturas voltadas para a Internet das Coisas. Além do uso destes protocolos necessários, é necessário validar quais destes protocolos e tecnologias são objetos de estudos do G3PD e já vêm sido utilizadas em estudos, a fim de contribuir de uma forma rica ao grupo, possibilitando trabalhos futuros em cima da arquitetura proposta para o gerenciamento e controle de recursos no ambiente computacional gerenciado pelo *middleware* EXEHDA, direcionado à Internet das Coisas.

A arquitetura de software do EXEHDA de acordo com a Figura 4.1 provê comunicação: (i) entre os Gateways e o Servidor de Borda; (ii) entre os Servidores de Borda e o Servidor de

Contexto; (iii) entre os Servidores de Contexto localizados em diferentes células do ambiente computacional; e (iv) com outros serviços do *middleware* ou aplicações. As linhas tracejadas definem a possibilidade de uso de Gateways físicos (Nativos ou Proprietários), somente Gateway virtual, ou ambos.

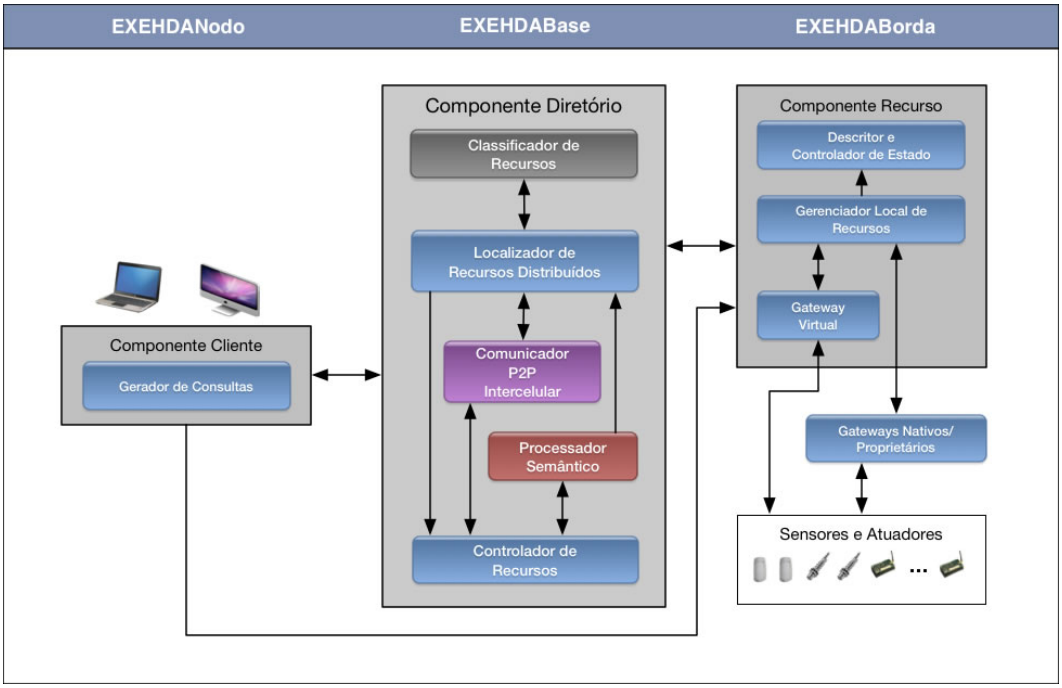

Figura 4.2: Arquitetura do Serviço de Descoberta Proposto

De acordo com a Figura 4.2, a arquitetura subdivide-se em três componentes distintos: (CD) Componente Diretório, (CR) Componente Recurso e (CC) Componente Cliente, cada qual possuindo características únicas e essenciais para a modelagem da arquitetura como um todo.

Para prototipação da arquitetura do EXEHDA-RD, foi empregada a linguagem Python, o framework Django-REST para interoperabilidade entre os componentes da arquitetura, a linguagem JSON para especificação de mensagens, o gerenciador de banco de dados PostgreSQL, o Apache Fuseki para realização do processamento semântico, para persistência de dados semânticos o banco de dados TDB, a linguagem SPARQL e a ferramenta Protégé para criação e edição da ontologia.

4.2 CC - Componente Cliente

Componente Cliente é o componente que se comunica diretamente com as aplicações que fazem operações de consultas a fim de localizar serviços oferecidos pelo recursos presentes no ambiente computacional gerenciado pelo *middleware* EXEHDA.

Este componente possui um módulo Gerador de Consultas, responsável por interpretar a descrição dos recursos necessários às aplicações no formato da linguagem SPARQL.

Esta comunicação se dá por meio de métodos que seguem o padrão arquitetural REST, que organizam as informações recebidas no formato JSON, incluindo as URIs necessárias para as aplicações acessarem os recursos.

Neste módulo concentram-se também os métodos responsáveis por disponibilizar os serviços dos recursos para as aplicações por meio de URIs.

4.3 CR - Componente Recurso

Neste componente da arquitetura encontram-se os métodos responsáveis pelo gerenciamento e controle de estado dos recursos presentes no ambiente computacional.

O componente é subdividido em 2 módulos: Descritor e Controlador de Estado e Gerenciador Local de Recursos.

4.3.1 Descritor e Controlador de Estado

Módulo que adiciona as seguintes funcionalidades à arquitetura:

- Extrair as informações dos arquivos de configurações enviados pelo Gerenciador Local de Recurso, para envio ao Componente Diretório;

- Envio de mensagens em intervalo de tempo pré-determinado ao Controlador de Recursos do Componente Diretório, a fim de assegurar que um determinado recurso realmente se encontra ativo no ambiente computacional.

4.3.2 Gerenciador Local de Recursos

O Módulo Gerenciador Local de Recursos se comunica diretamente com os Gateways Virtuais e com os Gateways Nativos/Proprietários obtendo informações atualizadas de quando um recurso se conecta ou desconecta em um EXEHDABorda e organiza estes recursos, onde este recursos fazem comunicação seguindo o padrão UPnP.

Gateway Virtual

Por meio do módulo de Gateway Virtual presente no EXEHDABorda, a arquitetura se torna apta a reconhecer quando um recurso se conecta em um determinado EXEHDABorda sem a dependência de Gateways de terceiros.

Gateways virtuais tratam-se de um módulo no Componente Recurso, que atua como Gateways, sem haver necessidade de um computador ou dispositivo embarcado agindo como Gateway no ambiente.

Gateways Proprietários

Gateways proprietários são equipamentos adquiridos comercialmente, onde o desenvolvedor não tem acesso à sua programação, para desenvolver novas funcionalidades, ou otimizar códigos. A Intel (INTEL, 2017) disponibiliza kits para configuração de Gateways e a Dell oferece servidores para gerenciamento de recursos em um ambiente computacional.

Gateways Nativos

Gateways nativos se tratam de dispositivos embarcados configurados para agirem como Gateways, construídos com placas de prototipação, tais como Arduino ou Raspberry.

4.4 CD - Componente Diretório

O Componente Diretório, localizado no EXEHDABase, se trata do componente com maior número de responsabilidades na concepção da modelagem arquitetural, sendo nele onde se encontram os módulos responsáveis por recebimento de mensagens dos outros componentes e gerenciamento, controle e organizações dos recursos presentes no ambiente computacional gerenciado pelo *middleware* EXEHDA.

Este componente é formado por cinco módulos: Controlador de Recursos, Processador Semântico, Comunicador P2P Intercelular, Localizador de Recursos Distribuídos e Classificador de Recursos, sendo estes módulos definidos a seguir, bem como métodos relevantes à arquitetura presente em cada um destes módulos.

4.4.1 Controlador de Recursos

O Controlador de Recursos provê as seguintes funcionalidades à arquitetura:

- Realizar manutenção dos estado dos recursos presentes no ambiente computacional na ontologia e na base de dados relacional;
- Disponibilizar interface de comunicação entre o Componente Cliente e Componente diretório, provendo informações de recursos ativos para aplicações gerais;
- Manter atualizado de uma forma eficiente os recursos anunciados pelo Componente Recurso e manter uma interface de troca de mensagens entre o Componente Recurso e Componente Diretório, a fim de estar com informações atualizadas acerca dos recursos presentes no ambiente.

Quando referenciando-se aos recursos presentes no ambiente computacional, a modelagem parte da premissa de que, todo e qualquer recurso quando descoberto pela primeira vez tem suas características extraídas de seu arquivo de configuração para armazenamento na ontologia. Uma vez presente no ambiente, há troca de mensagens entre o Componente Diretório e o Componente Recurso, garantido que todos recursos que não tenham seu *lease* renovado no período previsto sejam desativados.

A modelagem proposta defende o uso de três possíveis valores para o estado atual de um recurso presente no ambiente: "Inativo", "Ativo"e "Negado".

- **Inativo:** estado de um recurso que se encontra no ambiente mas não teve seu estado renovado no limite de tempo previsto para sua renovação na arquitetura;
- **Ativo:** estado de um recurso se fez descoberto recentemente pela arquitetura e que pode ser acessado por aplicações;
- **Negado:** estado de recurso definido de forma manual na arquitetura. Todo e qualquer recurso neste estado será ignorado pela aplicação.

Para auxiliar com o controle de estado de recursos, são mantidas informações do estado atual de cada recurso presente no ambiente, bem como informações referentes à última data que determinado recurso enviou uma mensagem se auto-afirmando ativo no ambiente computacional.

Por sua vez, o módulo Controlador de Recursos possui um serviço, denominado Controlador de Estado, que utiliza os seguintes repositórios: Repositório de Recursos Ativos(RRA) e o Repositório de Recursos Negados(RRN).

Controlador de Estado

O serviço Controlador de Estado é responsável por verificar periodicamente todos os recursos que não tiveram seu estado renovado, removendo-os do Repositório de Recursos Ativos.

Este serviço implementa uma funcionalidade de autoconfiguração, visto que não é necessária a iteração do usuário para realizar a manutenção do estado do recurso.

Repositório de Recursos Ativos

O RRA é um repositório em base de dados relacional Postgres que mantém armazenados todos os recursos que se encontram ativos no ambiente computacional gerenciado pelo *middleware* EXEHDA, seu identificador e data de última atualização, tempo este utilizado para validação de seu estado atual.

Repositório de Recursos Negados

O RRN é um repositório em base de dados relacional Postgres que mantém armazenados os recursos no primeiro momento em que eles forem anunciados pelo Componente Recurso, e que não podem ser retornados em consultas realizadas por aplicações externas, tão quanto tem seus estados validados pelo Controlador de Estado.

4.4.2 Processador Semântico

Por meio do processador semântico, são realizadas as consultas à ontologia. Neste processador são utilizados raciocinadores com o intuito de aumentar a expressividade das consultas, gerando por sua vez, melhores resultados.

O Processador Semântico atual foi aperfeiçoado e sua ontologia remodelada para fazer

uso do servidor Apache Fuseki, base de dados TDB e linguagem de consulta SPARQL. A Figura 4.3 demonstra a ontologia após as modificações realizadas.

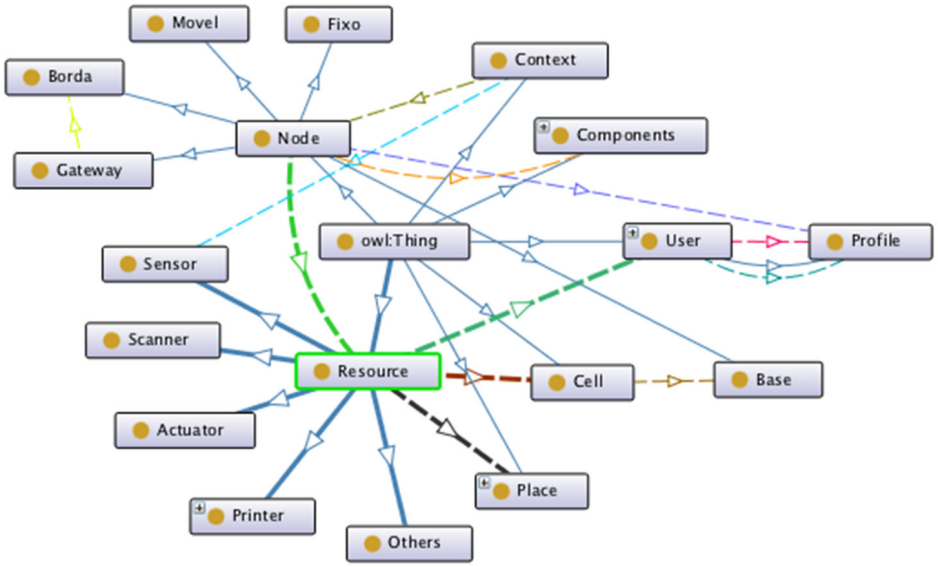

Figura 4.3: Ontologia Remodelada ao Cenário da IOT

Apache Fuseki

Apache Fuseki é um componente do Apache Jena que engloba vários componentes do Apache Jena a fim de disponibilizar um servidor SPARQL *standalone*, permitindo ser utilizada demais funcionalidades providas pelo Apache Jena (HOFEREK, 2012). A ferramenta Jena nos oferece as seguintes funcionalidades:

- API para leitura, processamento e escrita de dados RDF em XML, em formato de triplas;
- Uma API de ontologia para manipular OWL com uso de RDF;
- Motor de inferência de raciocinadores RDF sobre dados OWL;
- Armazenamento de grande quantidades de triplas RDF armazenadas em disco;
- Motor de consulta com base nas últimas especificações da linguagem SPARQL;
- Conjunto de configurações internas que permite que dados RDF seja publicado por outras aplicações, utilizando uma variedade de protocolos, incluindo SPARQL.

Em adição a todas estas características, o Fuseki permite que as consultas, atualizações e uploads de arquivos SPARQL sejam direcionadas a uma base de dados. Também conta com a presença de validadores para consultas e atualizações SPARQL, expandindo a comunicação para formatos além de RDF ou XML. (XIN et al., 2013)

Com base em sua utilização foi possível integrar a API REST da arquitetura com o servidor SPARQL sem haver necessidade de modificações específicas.

TDB

TDB se trata de uma base de dados que armazena informações em pares contendo chaves e valores, sendo seus dados armazenados em formato binário. No Apache Jena ele é o componente responsável por armazenar os dados RDF e realizar as consultas por meio da linguagem SQL. O Apache Fuseki permite que sejam realizados procedimentos com a linguagem SPARQL ao conjuntos de dados presentes no TDB baseado em padrões REST sobre o protocolo HTTP. (XIN et al., 2013)

SPARQL

De acordo com a W3C, SPARQL contém um conjunto de padrões de triplas chamado padrão básico de grafos. O resultado de dados de consulta SPARQL podem retornar conjuntos de grafos RDF. É considerada uma linguagem para trabalhar com dados RDF pela possibilidade de gerar consultas mais complexas, relacionadas diretamente com expressividade em resultados de buscas. (PRUD'HOMMEAUX; SEABORNE, 2008)

4.4.3 Comunicador P2P Intercelular

O Comunicador P2P Intercelular é o módulo que capacita a modelagem arquitetural distribuir a consulta realizada em uma única célula do ambiente computacional para as demais células do ambiente. Desta forma há a obtenção de um maior número de resultados sem a necessidade da intervenção do usuário para consulta nas demais células.

A intercomunicação entre células também garante com que não haja ambiguidade em resultados de buscas de recursos, bem como capacita a arquitetura retornar os recursos que melhor se adequem à necessidade da aplicação, independente de sua localização.

4.4.4 Localizador de Recursos Distribuídos

O Localizador de Recursos Distribuídos é o módulo responsável pela organização de resultados obtidos, notificando quando necessário o Comunicador P2P Intercelular, para que o mesmo interrompa a consulta intercelular.

4.4.5 Classificador de Recursos

O Classificador de Recursos tem por objetivo realizar o ranqueamento de recursos coletados pelo módulo Localizador de Recursos Distribuídos relacionando com variáveis de diferentes natureza, como por exemplo: localização, disponibilidade e frequência de uso.

4.5 EXEHDA-RD: Fluxo Operacional

Esta seção descreve as diversas situações previstas na arquitetura do EXEHDA-RD, referente ao controle dinâmico da presença dos recursos no ambiente computacional do *middleware* EXEHDA. As trocas de mensagens entre os componentes da arquitetura durante o processo de detecção da presença dos recursos estão representadas através de Diagramas de Sequência.

Recursos já descobertos ao menos uma vez pela arquitetura possuem suas características armazenadas na ontologia, enquanto recursos que se encontram ativos, além destas informações tem seu identificador e data de última atualização de estado do recurso a âmbito de disponibilizar uma forma de validar o tempo de troca de mensagens entre o Componente Recurso e o Componente Diretório.

A primeira vez que um recurso novo é descoberto pela arquitetura, as informações contidas em seu arquivo de configuração XML são extraídas e enviadas para ontologia. Também neste momento, é enviado o identificador único UUID de determinado recurso à base de dados relacionadas, em conjunto com suas características a fim de que possa ser gerenciado o seu estado pela arquitetura.

A comunicação entre todos os componentes presentes na arquitetura se dá seguindo o estilo arquitetural RESTful, com URIs para identificar os recursos e compartilhar informações entre todos componentes, enquanto as mensagens de troca entre os componentes são descritas no formato JSON, sendo legíveis para aplicações de diversas naturezas.

Há também uma comunicação P2P entre células para disponibilizar e expandir infor-

mações para todo o ambiente, não ficando centralizada em apenas uma célula do ambiente computacional gerenciado pelo *middleware* EXEHDA.

4.5.1 Requisições REST dos Componentes

A presente seção caracteriza a forma como o padrão REST foi aplicado na modelagem, destacando as características mais presentes neste padrão arquitetural e que foram utilizadas no trabalho proposto.

Disparo de uma Consulta pelo Componente Cliente

A Figura 4.4 ilustra por meio de um Diagrama de Sequência como a arquitetura se comporta ao ser realizada uma consulta de recursos disponíveis baseada em um conjunto de critérios. Quando a aplicação realiza uma consulta, o primeiro processo é executado pelo módulo Gerador de Consultas do Componente Cliente, gerando a partir daí um arquivo formatado em JSON contendo o tipo de recurso que se deseja pesquisar e a categoria de sensor pesquisado. Os atributos de consultas podem ser omitidos para realizar a consulta, não sendo obrigatórios quaisquer um deles.

Por sua vez, o componente Controlador de Recursos é responsável em enviar os dados em formato SPARQL ao componente Processador Semântico e uma cópia dos critérios utilizados ao componente Comunicador P2P Intercelular. O componente Comunicador P2P Intercelular envia os dados da consulta para demais células, a fim de permitir que a mesma consulta se expanda para demais células presentes no ambiente, acarretando uma maior expressividade dos dados. O Processador Semântico por sua fez realiza conexão ao servidor Apache Fuseki e interpreta a query de consulta no formato SPARQL.

O servidor Apache Fuseki retorna os resultados em formato JSON para a arquitetura e este resultado é enviado ao Localizador de Recursos Distribuídos. O Localizador de Recursos Distribuídos é responsável em validar se há uma quantidade de resultados considerados eficientes para o resultado da busca e, no caso de houverem menos resultados do que o mínimo, são realizadas solicitações de requisição de buscas às demais células pelo componente Comunicador P2P Intercelular até o momento que atingir o limite mínimo de resultados.

Após os resultados serem alcançados, é enviado ao componente Classificador de Recursos, a fim de ranquear os recursos que estejam mais de acordo com a consulta requirida,

retornando estes resultados classificados ao componente Localizador de Recursos Distribuídos. Os resultados são retornados ao componente Controlador de Recursos, que organiza os resultados em formato JSON e que seja legível para demais aplicações, retornando o resultado ao módulo Gerador de Consultas do Componente Cliente.

A Figura 4.5 apresenta o formato de dados que é produzido pelo componente Gerador de Consultas, a fim de disponibilizá-lo para as aplicações.

```
"data": [
    {
        "uuid": "eaed42ba-6a4c-47c8-8bd9-2eab0aa76f72",
        "gateway": "GW01",
        "resource": "Sensor",
        "category": "Temperatura",
        "uri": "http://190.12.34.32/recurso/3"
    },
    {
        "uuid": "eaed42ba-6a4c-47c8-8bd9-2eab0aa76f72",
        "gateway":"GW02",
        "resource": "Atuador",
        "category": "Lampada",
        "uri": " http://190.12.34.36/recurso/2"
    }
]
```

Figura 4.5: JSON formatado para as Aplicações

Anúncio de um Novo Recurso

Há pesquisas em andamento pelo G3PD no método que recursos são anunciados no EXEHDABorda, denominado EXEHDA-IOT (DAVET, 2016), onde existem Gateways com a responsabilidade de anunciar um novo recurso ao EXEHDABorda, ao momento que o mesmo estiver presente e conectado em algum deles. Quando um recurso é anunciado no ambiente, é enviado um arquivo no formato JSON ao componente recurso no formato da Figura 4.6:

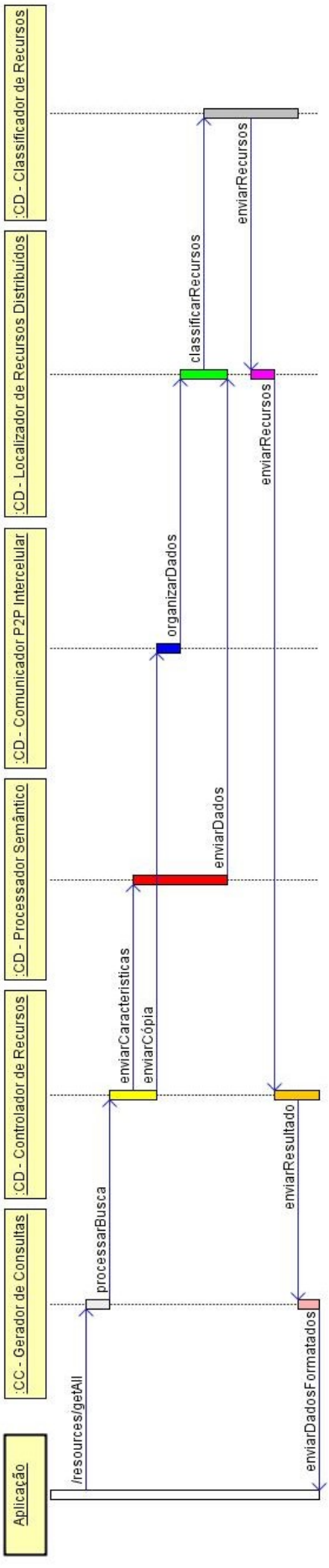

Figura 4.4: Diagrama de Sequência de Solicitação de um Recurso

```
"data": {
    "gateway": "GW01",
    "uuid": "ed6e0911-a7f7-4935-86f2-5364bb47df91",
    "resource": "Sensor",
    "category": "Temperatura"
}
```

Figura 4.6: JSON Anúncio de Recurso

O anúncio de um recurso em ambiente engloba as seguintes situações: recurso entrando pela primeira vez, entrada de um recurso negado, entrada de um recurso ativo.

Primeira Entrada de um Recurso

Conforme Figura 4.7, o diagrama de sequência representa os procedimentos realizados pela arquitetura quando um recurso está entrando pela primeira vez no ambiente computacional gerenciado pelo *middleware* EXEHDA. Quando conectado em um determinado Gateway, podendo o mesmo ser um Gateway Nativo, Proprietário ou Virtual, por meio dos protocolos que regem o padrão UPnP, são extraídas as informações do dispositivo conectado e enviadas ao módulo Gerenciador Local de Recursos presente no Componente Recurso. Este dados são então organizados para ser enviado ao módulo Descritor e Controlador de Estado.

O módulo Descritor e Controlador de Estado, é responsável em enviar o conjunto de características do recurso e notificar a arquitetura que ela precisa realizar os testes para realizar o cadastro do novo recurso corretamente na ontologia e no RRN.

A partir destas informações, o módulo Controlador de Recursos é responsável em gerar a inserção em SQL para RRN e o procedimento de atualização na ontologia na linguagem SPARQL.

O módulo Processador Semântico, é responsável em realizar a conexão com o servidor Fuseki e enviar a inserção para ser realizada na ontologia. O Processador Semântico, logo após verifica se o recurso foi inserido corretamente na ontologia, para que, caso esteja cadastrado corretamente, seja enviada ao módulo Controlador de Recursos, uma mensagem de sucesso, para ser executado o serviço responsável em registrar o recurso no Repositório de Recursos Negados.

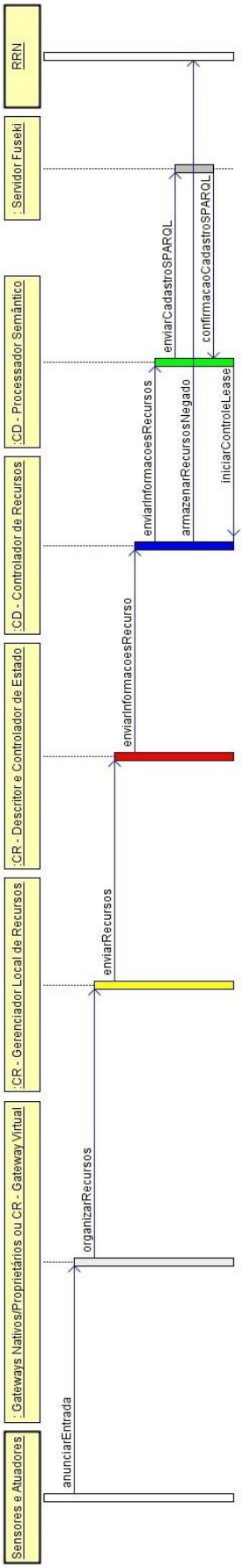

Figura 4.7: Primeira Entrada do Recurso no Ambiente

Entrada de Recurso Negado

Quando é anunciado um recurso negado no ambiente, são seguidos os mesmos procedimentos de uma Primeira Entrada de um Recurso, até chegar ao módulo Controlador de Recursos. Logo após ser realizado estes procedimentos, o módulo Controlador de Recursos gera um procedimento de atualização na ontologia em linguagem SPARQL ao servidor Fuseki.

Após retornada a mensagem de sucesso por parte do servidor Fuseki, o módulo Controlador de Recursos segue a confirmação de negação do recurso, para que não seja permitido o acesso aos serviços do recurso, nem mesmo seja executado nenhum método no mesmo conforme Figura 4.8

Entrada de Recurso Inativo

Quando um determinado recurso é descoberto no ambiente e o mesmo se encontra inativo na ontologia, o módulo Controlador de Recurso envia um procedimento para alterar o estado na ontologia no formato SPARQL. O Processador Semântico retorna uma confirmação de mudança de estado ao módulo Controlador de Recursos.

O módulo Controlador de Recursos se responsabiliza em gerar os procedimentos em SQL responsáveis por remover o recurso do RRN e adicionar ao RRA, conforme é demonstrado na Figura 4.9

4.6 Considerações Sobre o Capítulo

Este capítulo apresentou as características consideradas para modelagem do EXEHDA-RD, bem como sua arquitetura de software. Os Componentes Recurso, Diretório e Cliente foram descritos e apresentado o fluxo operacional dos componentes em cada um dos cenários operacionais.

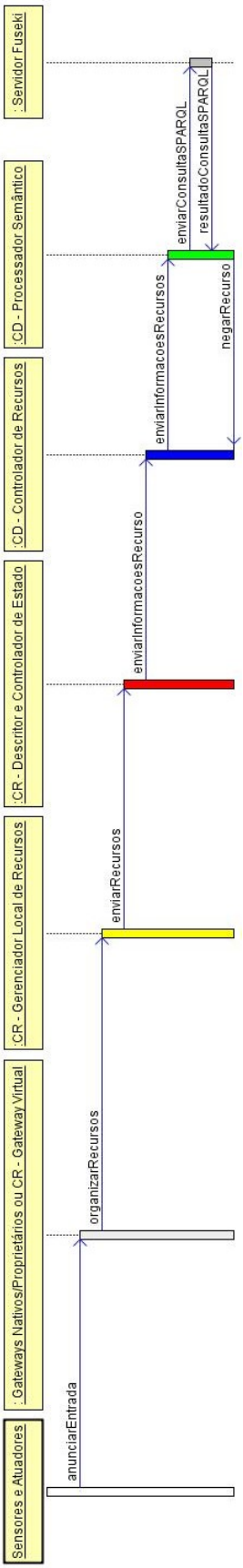

Figura 4.8: Entrada de Recurso Negado

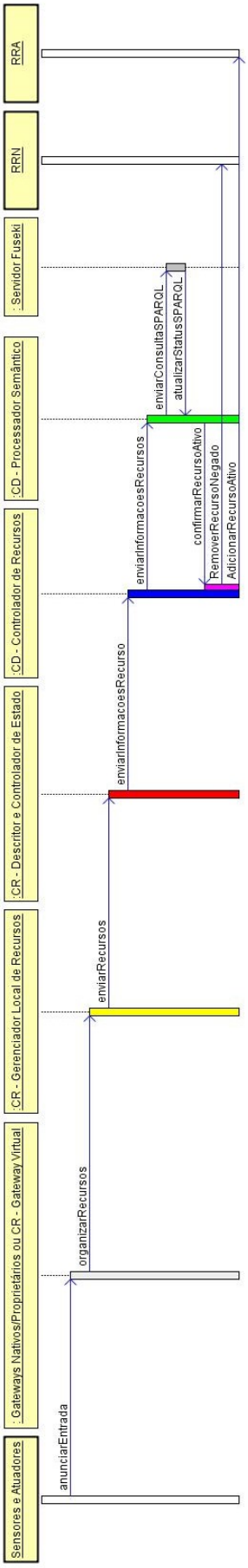

Figura 4.9: Entrada de Recurso Inativo

5 EXEHDA-RD: CENÁRIOS DE USO

Este capítulo descreve três cenários de uso que foram empregados para validar o Serviço de Descoberta de Recursos proposto. Sucintamente são caracterizados os elementos envolvidos e a problematização em cada cenário. As funcionalidades previstas no EXEHDA-RD serão exploradas através destes cenários.

5.1 Caracterização dos Cenários de Uso

Abaixo a descrição dos aspectos que serão considerados nos três cenários de uso previstos para avaliação do EXEHDA-RD:

- **Administrador:** pessoa autorizada para a realização das operações de manutenção da arquitetura, bem como a autorização de novos recursos auto-descobertos no ambiente computacional;
- **Recurso:** recurso vinculado a um Gateway Proprietário ou Nativo, ou Virtual de um EXEHDABorda, o qual deverá ter seu estado operacional atualizado na arquitetura, podendo assim ser descoberto. Este recurso pode ser um sensor ou outro tipo de dispositivo conectado no ambiente computacional a fim de prover serviços que possam ser utilizados por aplicações;
- **Componente Recurso:** componente responsável por realizar a intercomunicação do EXEHDABorda aos recursos, intermediados por Gateways, sendo eles Nativos, Proprietários ou Nativos;
- **Componente Diretório:** componente situado no EXEHDABase, sendo responsável pelo gerenciamento e armazenamento das características e recursos presente em uma célula de execução do EXEHDA, a URL de acesso definida para acesso aos serviços disponibilizados pelo componente foram acessados através do endereço IP ***http://172.16.1.254:8000***;
- **Componente Cliente:** componente que possui o módulo Gerador de Consultas, responsável em enviar a consulta em linguagem SPARQL ao Componente Diretório.Com o emprego da API disponibilizada pelo EXEHDA-RD foi possível a simulação de resultados de buscas baseados em consultas realizadas com a linguagem SPARQL;

- **Emulador CORE:** emulador de interfaces de sistemas distribuídos *Open Source*, e que para a simulação dos cenários foi configurado para se comunicar com o Componente Diretório por meio do endereço IP ***http://172.16.1.254***;

- **Linguagem Python:** linguagem de programação já utilizada amplamente pelo G3PD para desenvolvimento de componentes para o *middleware* EXEHDA e que foi adotada com padrão do EXEHDA-RD a fim de permitir uma melhor integração aos componentes já presentes no *middleware* direcionados para a Descoberta de Recursos;

- **Apache Fuseki:** servidor que faz uso da linguagem SPARQL para consultas em ontologias com o objetivo de obter expressividade em resultados de consultas. Na presente dissertação para construção dos cenários o servidor Apache Fuseki foi instalado em uma máquina virtual com sistema operacional Linux Mint;

- **VMWare:** estação de trabalho para emulação de sistemas operacionais. Foi criada uma imagem do Sistema Operacional Linux Mint, que se trata de uma distribuição Linux, a fim de garantir uma melhor eficiência nos testes com os cenários de uso em razão das ferramentas utilizadas serem Open Source.

5.2 Cenário 1 - Manutenção e Gerenciamento de Estado de Recursos pela Arquitetura

A realização da ativação de um recurso no emulador CORE se dá por meio do disparo de um conjunto de comandos dentro do próprio emulador que comunica ao serviço do Componente Diretório, enviando o UUID do recurso e todas suas características. Esta mensagem é sempre a mesma e enviada continuamente por parte do recurso.

Ao receber esta mensagem, o módulo de Controlador de Recursos do Componente Diretório realiza as seguintes operações para realizar o gerenciamento deste recurso:

- Na primeira vez que recebe a mensagem no Componente Diretório, o Processador Semântico dispara o método de inserção de um novo objeto na ontologia e o Controlador de Recursos insere na base de dados relacionais, no Repositório de Recursos Negados (RRN);

- Quando o Componente Diretório recebe a mensagem em um segundo momento, o mesmo se encontrando negado na Ontologia, ele ignora todos os métodos;

- Se o Componente Diretório recebeu a mensagem e o recurso já foi autorizado pelo administrador, o Processador Semântico realiza o disparo do método responsável em modificar o estado do recurso para Ativo na Ontologia em linguagem SPARQL;
- Confirmada a atualização de mudança de estado na ontologia, o módulo Controlador de Recursos do Componente Diretório executa o método responsável por inserir as informações do recurso na base de dados relacionais, no Repositório de Recursos Ativos (RRA);
- Ao receber uma mensagem, e o recurso em questão já estiver com seu estado configurado como ativo na ontologia e o mesmo já estiver presente no Repositório de Recursos Ativos (RRA), o módulo Controlador de Recursos se responsabiliza em disparar o método responsável em atualizar a data de última atualização deste recurso no Repositório de Recursos Ativos.

A Figura 5.1 apresenta o código em SPARQL utilizado na inserção de um novo recurso na ontologia pela arquitetura.

```
sparql = SPARQLWrapper("http://127.0.0.1:3030/inf/update")
sparql.setQuery("""
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

INSERT DATA {
        ont:""" + data['uuid'] + """ rdf:type ont:""" + data['resource'] +
            ↪ """, owl:NamedIndividual .
        ont:""" + data['uuid'] + """ ont:Resource_Status 'Negado' .
        ont:""" + data['uuid'] + """ ont:Resource_Type '""" + data['category
            ↪ '] + """' .
        ont:""" + data['uuid'] + """ ont:Resource_Node ont:""" + data['
            ↪ gateway'] + """.
}
""")

sparql.method = 'POST'
sparql.query()
```

Figura 5.1: Inserção de recurso em SPARQL

Para realização do controle de leasing dos recursos foi iniciado o serviço de **threads.py** no Componente Diretório, responsável por verificar de 30 em 30 segundos se existe recursos no

Repositório de Recursos Ativos (RRA) que não tenham sido atualizados a mais 3 minutos da data e hora atuais. Este intervalo de tempo, inicialmente configurado para 3 minutos pode ser modificado passando-se um novo valor para o atributo **lease**, em segundos, na inicialização do serviço. Ao ser executado o serviço, todos os recursos que não corresponderem aos requisitos possuem seu estado modificado para Inativo na Ontologia e são removidos do Repositório de Recursos Ativos (RRA), por meio de um método executado pelo módulo Processador Semântico do Componente Diretório em linguagem SPARQL.

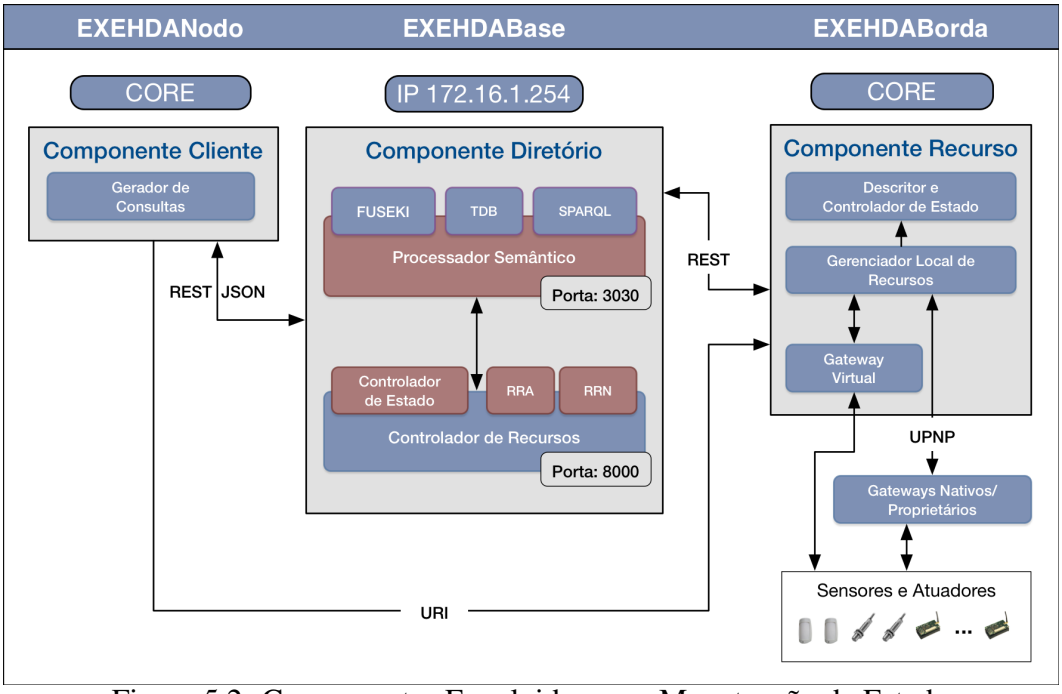

Figura 5.2: Componentes Envolvidos com Manutenção de Estado

A Figura 5.2 demonstra os componentes envolvidos com a manutenção de recursos pela arquitetura, que foram instanciados no emulador CORE.

Foram realizados os testes em uma máquina virtual Linux por meio do aplicativo de virtualização VMWare, estando instalado nesta máquina virtual o serviço de Controle de Recursos do Componente Diretório, o gerenciador banco de dados Postgres, e também o servidor Apache Fuseki. O emulador CORE foi empregado para construir uma infraestrutura distribuída contendo vários recursos.

O emulador CORE é ativado por meio do seguinte comando "**CORE-gui**" através do Prompt de comando. A Figura 5.3 representa a infraestrutura distribuída que é construída empregando os recursos do emulador CORE.

Uma vez ativo o CORE, foi inicializado o Servidor Apache Fuseki na porta 3030. Para concluir a ativação do servidor, foi necessário acessar o endereço IP **http://localhost:3030** e realizar o upload da ontologia ao Apache Fuseki.

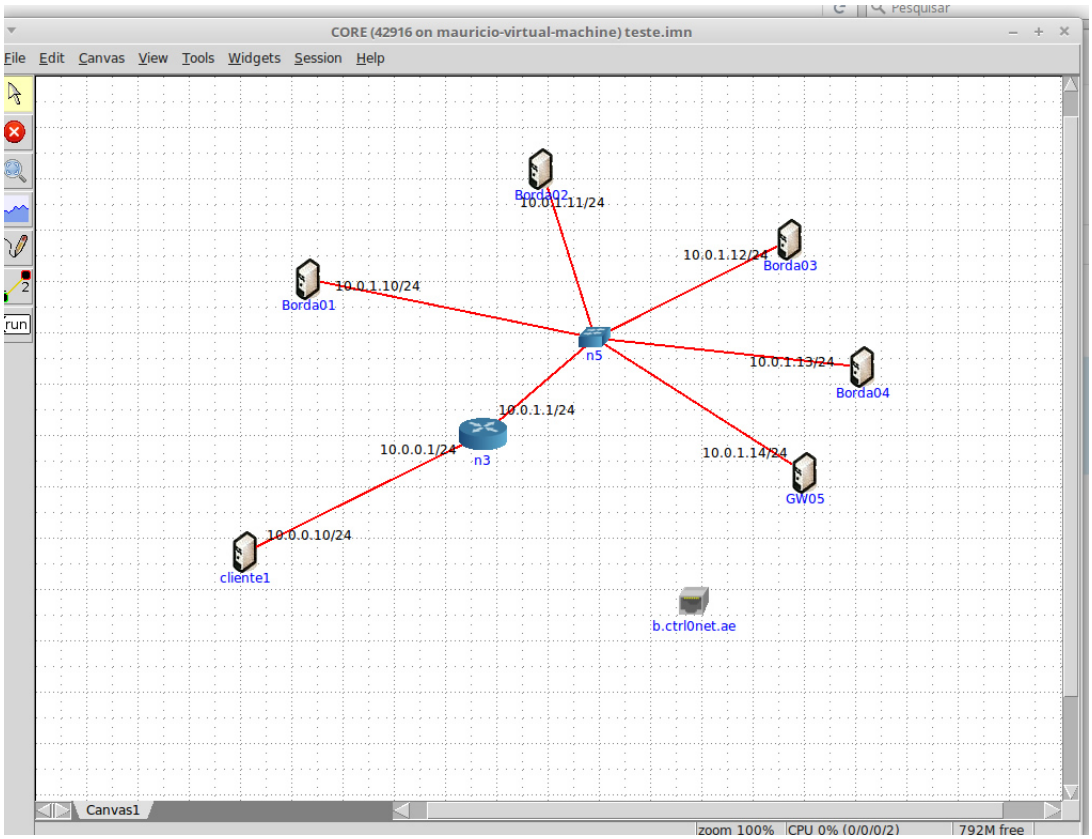

Figura 5.3: CORE Emulando uma Infraestrutura Virtual

Com o servidor Apache Fuseki ativo, foi inicializado o serviço de Controle e Manutenção da Presença de Recursos no componente diretório por meio do seguinte comando:

```
python manage.py runserver
```

Após realizado o comando de inicialização dos serviços foi executado um comando Shell Script a partir de um nodo do CORE enviando as seguintes informações para serviço do Componente Diretório, localizado no equipamento hospedeiro. Este procedimento tem por objetivo simular o Componente Recurso presente em um Gateway virtual através de mensagens de anúncio no ambiente emulado. Para ativar o Componente foi usado o seguinte comando:

```
sh base14.sh
```

PostgreSQL 9.6-exehda-rd-public.resources_denied

```
1  SELECT * FROM public.resources_denied
2  ORDER BY uuid
3  ASC
```

Data Output | Explain | Messages | History

added timesta...	uuid [PK] uuid	status characte...	resource text	gateway text	category text
2017-01-...	4cef01da-bfa4-11e6-a4a6-cec0c932ce12	1	Sensor	GW02	Temperatura
2017-01-...	4cef01da-bfa4-11e6-a4a6-cec0c932ce14	1	Sensor	GW02	Temperatura

Figura 5.4: Inserção do Recurso na Base de Dados

Por se tratar do primeiro momento que o recurso entrou no ambiente, o mesmo foi inserido na ontologia com seu estado Negado e inserido na base de dados relacionais no Repositório de Recursos Negados (RRN) conforme demonstrado na Figura 5.4.

Após ser liberado o acesso do recurso pelo administrador, no próximo anúncio realizado pelo recurso, o seu status é modificado para ativo na ontologia e o recurso é inserido na Repositório dos Recursos Ativos conforme Figura 5.5. Após este momento, toda vez que um recurso dispara uma mensagem de anúncio, é atualizada a data de último atualização do recurso no Repositório de Recursos Ativos, data esta que é utilizada pelo serviço de manutenção de estado dos recursos no módulo Controlar de Recursos do Componente Diretório.

	added timestamp with time zone	updated timestamp with time zone	uuid [PK] uuid	uri character varying(200)
1	2017-01-27 02:29:20.933345-02	2017-01-27 02:29:31.099201-02	4cef01da-bfa4-11e6-a4a6-cec0c932ce14	http://teste.com/sensor/sensor5
*				

Figura 5.5: Tabela de Recursos Ativos

A inicialização do serviço de verificação de leasing de cada um dos recursos foi disparado no Componente Diretório por meio do seguinte comando:

```
python threads.py
```

5.3 Cenário 2 - Gerência da Presença de Novos Recursos

Apesar dos recursos serem auto-descobertos na arquitetura, a presença de um gerenciamento de acesso aos recursos de forma manual é essencial, a fim de garantir a não presença de recursos indesejáveis no ambiente computacional gerenciado pelo *middleware* EXEHDA.

As mensagens enviadas do Componente Recurso ao Componente Diretório contém o UUID dos recursos. É verificada na ontologia a existência do UUID do recurso, caso seja um novo recurso, ele será inserido e seu estado definido como "negado". O Administrador deverá liberar o acesso através de interface própria para que o recurso fique disponível no ambiente. A partir desta interface o administrador tem a possibilidade de modificar o estado de negado para inativo, permitindo desta forma que os recursos se tornassem visíveis para os métodos presentes nos componentes da arquitetura.

Para testar o cenário, foi verificado se os recursos em sua primeira anunciação no ambiente foram inseridos corretamente no Repositório de Recursos Negados (RRN), e se as verificações de presença na ontologia foram respeitadas, a fim de saber se um determinado recurso deveria ter seu estado atual modificado para "negado"ou, se houve a necessidade de inserir um novo recurso na ontologia e no Repositório de Recursos Negados (RRN).

A Figura 5.6 apresenta a interface disponível ao administrador do ambiente que dá acesso a listagem de todos recursos negados presentes no Repositório de Recursos Negado (RRN). É mostrada uma lista com todos os recursos, e um campo de marcação onde o administrador poderá liberar a utilização de um recurso no ambiente. A interface pode ser acessada através da URL http://172.16.1.254/denieds/all/.

Figura 5.6: Lista de Recursos Negados

Ao clicar no botão “Liberar Recurso” é enviada uma requisição via POST para a URL http://172.16.1.254:8000/resources/renew/ com os dados de determinado recurso selecionado, e a lista é atualizada após mensagem de retorno de sucesso do método, sendo removido este

recurso do repositório de recursos negados, tendo seu status modificado para Inativo na ontologia, podendo o mesmo ser ativado na próxima vez que ele enviar uma mensagem de anúncio, ficando o Repositório de Recursos Negados sem aquele recurso, onde o bloco de código da Figura 5.7 caracteriza a troca do estado do recurso na ontologia.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

    DELETE {?resource ont:Resource_Status 'Negado'}
    INSERT {?resource ont:Resource_Status 'Inativo'}

    WHERE { ?resource owl:sameAs ont:%s . }
```

Figura 5.7: Atualização do Status do Recurso de Negado para Inativo

Uma vez que este recurso estiver liberado pelo administrador, o componente recurso poderá enviar novas mensagens ao Componente Diretório, tornando assim o recurso ativo.

5.4 Cenário 3 - Explorando o Processador Semântico

O presente cenário teve como objetivo caracterizar a operação do processador semântico quando do atendimento de solicitações de clientes. O emprego de uma abordagem semântica possibilita que sejam feitas inferências sobre os recursos existentes, também trazendo resultados que contemplam características semelhantes. Neste sentido, este cenário também apresenta o fluxo de troca de mensagens entre componentes.

O cenário tem como ponto de partida a solicitação do Componente Cliente por recursos que estão sendo gerenciados por Gateways que empregam o sistema operacional Unix. O formato desta solicitação está especificada na Figura 5.8.

O Componente Cliente, portanto, é responsável pela especificação e envio da consulta em SPARQL ao Componente Diretório. O Processador Semântico (servidor Fuseki) por sua vez retorna o resultado em formato JSON, que é devolvido à aplicação por meio do Componente Cliente.

As consultas foram feitas, primeiramente com o raciocinador desativado, logo após com o mesmo ativado, para demonstrar a expressividade obtida com uso de raciocinadores na ontologia, onde o bloco abaixo representa a consulta em SPARQL com raciocinadores.

Após inicializado o servidor Apache Fuseki no endereço http://127.0.0.1:3030 e inicializado o serviço de controle de recursos no componente diretório no endereço http://172.16.1.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT *
WHERE {
        ?resource rdf:type ont:Sensor .
        ?resource ont:Resource_Node ?node .
        ?node rdf:type ont:Gateway .
        ?node ont:Node_Components ?sysope .
        ?sysope rdf:type ont:Unix .
}
```

Figura 5.8: Especificação de Consulta em SPARQL

254:8000, foi acessada a interface de consulta de recursos por meio do endereço http://172.16.1.254:8000/resources/client/ no navegador, onde é acessada uma tela conforme a Figura 5.9.

Figura 5.9: Interface de Consulta de Recursos

Para validar a expressividade do uso de ontologias, foram realizadas duas consultas SPARQL buscando todos recursos que fossem classificadas como distribuições UNIX. Sem o uso de raciocinador a consulta trouxe apenas sistema gateway Solaris, enquanto ao ser utilizado o raciocinador, os resultados trouxeram também distribuições Linux, retornando gateways Ubuntu e Debian. As Figuras 5.10 e 5.11 apresentam os resultados obtidos diretamente da interface gráfica do servidor Fuseki.

A Figura 5.12 apresenta o resultado da consulta em JSON enviada ao Componente Cliente e a Figura 5.13 apresenta o resultado da consulta em JSON, com o raciocinador ativado.

	resource	type	node	sysope
1	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce02	"Umidade"	ont:GW02	ont:Solaris

Showing 1 to 1 of 1 entries

Figura 5.10: Resultado de Busca sem Raciocinador

	resource	type	node	sysope
1	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce01	"Temperatura"	ont:GW01	ont:Ubuntu
2	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce02	"Umidade"	ont:GW02	ont:Solaris
3	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce03	"Temperatura"	ont:GW03	ont:Debian

Showing 1 to 3 of 3 entries

Figura 5.11: Resultado de Busca com Raciocinador


```
{
  "head": {
    "vars": [ "resource" , "type" , "node" , "sysope" ]
  } ,
  "results": {
    "bindings": [
      {
        "resource": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#4cef01da-bfa4-11e6-a4a6-cec0c932ce02" } ,
        "type": { "type": "literal" , "value": "Umidade" } ,
        "node": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#GW02" } ,
        "sysope": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#Solaris" }
      }
    ]
  }
}
```


Figura 5.12: Resultado JSON s/ Raciocinador


```
{
  "head": {
    "vars": [ "resource" , "node" , "sysope" ]
  } ,
  "results": {
    "bindings": [
      {
        "resource": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#4cef01da-bfa4-11e6-a4a6-cec0c932ce01" } ,
        "node": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#GW01" } ,
        "sysope": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#Ubuntu" }
      } ,
      {
```

```
        "resource": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#4cef01da-bfa4-11e6-a4a6-cec0c932ce02" } ,
        "node": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#GW02" } ,
        "sysope": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#Solaris" }
      } ,
      {
        "resource": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#4cef01da-bfa4-11e6-a4a6-cec0c932ce03" } ,
        "node": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#GW03" } ,
        "sysope": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#Debian" }
      }
    ]
  }
}
```


Figura 5.13: Resultado JSON c/ Raciocinador

5.5 Considerações Sobre o Capítulo

Este capítulo apresentou três cenários de uso que serão utilizados para validar o modelo proposto. Os cenários consideram que o EXEHDA-RD é capaz de detectar a presença de dispositivos confinados e realizar o controle da presença dos recursos de forma dinâmica, realizando o controle de estado dos recursos de uma forma relevante.

6 CONSIDERAÇÕES FINAIS

Este capítulo resume as principais conclusões decorrentes do trabalho desenvolvido nesta dissertação de mestrado, bem como relaciona possíveis trabalhos futuros.

A presente dissertação apresentou uma proposta de Arquitetura para Descoberta de Recursos na perspectiva da Internet das Coisas denominada EXEHDA-RD.

Com o atendimento das demandas originadas pela IoT, foi possível capacitar o *middleware* EXEHDA para gerenciar ambientes providos pela infraestrutura da Internet das Coisas, incorporando padrões e tecnologias da área.

6.1 Principais Conclusões

A Internet das Coisas está cada vez mais presente em nosso meio, o que espelha um aumento exponencial de trabalhos acadêmicos direcionados para esta área. Protocolos e Tecnologias são estudadas, bem como a realização de modelagem de novas arquiteturas e *middlewares* cada vez mais eficazes e eficientes.

O Processo de Descoberta se torna um dos principais requisitos a serem considerados ao se modelar uma arquitetura na perspectiva da Internet das Coisas, garantindo um controle eficiente de sua presença em razão da sua dinamicidade. A não garantia deste estado pode acarretar em perdas de plantações na área da agricultura ou influenciar em conclusões médicas na área da saúde, por exemplo.

Foi proposta a modelagem de arquitetura do EXEHDA-RD para garantir a Descoberta de Recursos no cenário da IoT pelo *middleware* EXEHDA, visando garantir a descoberta destes recursos e segurar que as informações de estado se mantenham sempre atualizadas.

Com as simulações realizadas nos estudos de caso, constatou-se a conformidade da modelagem proposta com os objetivos propostos, e sua adequação com padrões e premissas que regem soluções direcionadas para a Internet das Coisas.

6.2 Publicações

Esta seção apresenta as publicações desenvolvidas durante o período do mestrado, bem como a previsão de publicações a serem efetivas em periódicos reconhecidos pela área de Engenharias IV.

6.2.1 Efetivadas

- **DE AZEVEDO**, M. M.; DILLI, R.; FILHO, H. K.; DAVET, P. T.; LOPEZ. J. L.; YAMIN, A. C. Autonomic Ranking Resources in IoT Exploring Multiple Criteria with Machine Learning. In: **IM 2017 - Special Track on Management of IoT - 2017 IFIP/IEEE International Symposium on Integrated Network Management (IM 2017)**. Lisbon, Portugal: IM, 2017 (aguardando resultado).
- **DE AZEVEDO**, M. M.; DILLI, R.; DAVET, P. T.; YAMIN, A. C. Descoberta de Recursos no Middleware EXEHDA para o Cenário da IoT. In: **ERRC 2016 - XIV Escola Regional de Redes de Computadores**. Porto Alegre -RS: ERRC, 2016.

6.2.2 A Serem Efetivadas

- Future Generation Computer Systems (Qualis A2, Eng4)
- IEEE Sensors Journal (Qualis A1, Eng4)

6.3 Trabalhos Futuros

O emprego de um ambiente de emulação possibilita a expansão dos cenários de uso, permitindo a avaliação da arquitetura do EXEHDA-RD em diferentes cenários, potencializando a avaliação de suas funcionalidades. Considerando isto surgem diferentes alternativas de trabalhos futuros, dentre as quais destacaríamos:

- Qualificar o módulo Classificador de Recursos introduzindo funcionalidades de ranqueamento, as quais dentre os recursos descobertos irão indicar o mais oportudo para o usuário, tendo por base suas preferências;
- Aprimorar as interfaces gráficas para gerenciamento e administração da arquitetura do EXEHDA-RD, bem como dos recursos envolvidos;
- Explorar o uso de ontologias probabilísticas quando da descoberta e priorização dos recursos a serem identificados.

REFERÊNCIAS

AHRENHOLZ, J. Comparison of core network emulation platforms. In: **2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE**. [S.l.: s.n.], 2010. p. 166–171. ISSN 2155-7578.

AHRENHOLZ, J. et al. Core: A real-time network emulator. In: **MILCOM 2008 - 2008 IEEE Military Communications Conference**. [S.l.: s.n.], 2008. p. 1–7. ISSN 2155-7578.

ANTONIOU, G.; HARMELEN, F. v. Web ontology language: Owl. In: ____. **Handbook on Ontologies**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 91–110.

BARTLETT, G.; HEIDEMANN, J.; PAPADOPOULOS, C. **Understanding Passive and Active Service Discovery (extended)**. [S.l.], 2007. To appear, ACM Internet Measurement Conference 2007.

BELQASMI, F.; GLITHO, R. H.; FU, C. Restful web services for service provisioning in next-generation networks: a survey. **IEEE Communications Magazine**, v. 49, n. 12, p. 66–73, 2011. Available from Internet: <http://dblp.uni-trier.de/db/journals/cm/cm49.html#BelqasmiGF11>.

BROWN, A. **Efficient Service Discovery in Wide Area Networks**. Dissertation (Master) — University of Stirling, 2008.

BUTT, T. A. et al. Internet of things, smart spaces, and next generation networking: 13th international conference, new2an 2013 and 6th conference, rusmart 2013, st. petersburg, russia, august 28-30, 2013. proceedings. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. chp. Adaptive and Context-Aware Service Discovery for the Internet of Things, p. 36–47.

CHRISTENSEN, E. et al. **Web Service Definition Language (WSDL)**. [S.l.], 2001. Available from Internet: <http://www.w3.org/TR/wsdl>.

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

CORE. **Open-Source Routing and Network Simulation**. 2015. Available from Internet: <http://www.brianlinkletter.com/open-source-network-simulators/>.

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

DILLI, R. M. **Uma Proposta para Descoberta de Recursos na Computação Ubíqua com Suporte Semântico**. Dissertation (Dissertação de Mestrado) — Programa de Pós-Graduação em Informática - Universidade Católica de Pelotas, 2010.

DOBREV, P. et al. **Device and service discovery in home networks with OSGi**. [S.l.], 2002. v. 40, n. 8, 86-92 p.

ECMA. **ECMA-404: The JSON Data Interchange Format**. Geneva, Switzerland: Ecma International (European Association for Standardizing Information and Communication Systems), 2013. Available from Internet: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Thesis (PhD), 2000. AAI9980887.

GRUBER, T. R. A translation approach to portable ontology specifications. **Knowl. Acquis.**, v. 5, n. 2, p. 199–220, jun. 1993. ISSN 1042-8143.

HOFEREK, O. **Ontological Reasoning with Taxonomies in RDF Database**. Dissertation (MASTER THESIS) — Department of Software Engineering - Charles University, 2012.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; HARMELEN, F. V. From shiq and rdf to owl: The making of a web ontology language. **Journal of Web Semantics**, v. 1, p. 2003, 2003.

INGMARSSON, M. **Creating & Enabling the Useful Service Discovery Experience: The Perfect Recommendation Does Not Exist**. Dissertation (Master) — Linköping University, 2013.

INTEL. **IOT Gateway Technology**. 2017. Disponível em: < https://software.intel.com/en-us/iot/hardware/gateways >. Acesso em janeiro de 2017.

JUNIOR, N. V. de A.; BASTOS, D. B.; PRAZERES, C. V. S. Web of things: Automatic publishing and configuration of devices. In: **Proceedings of the 20th Brazilian Symposium on Multimedia and the Web**. New York, NY, USA: [s.n.], 2014. (WebMedia '14), p. 67–74. ISBN 978-1-4503-3230-9.

KINDBERG, T.; FOX, A. System software for ubiquitous computing. **IEEE Pervasive Computing**, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 1, n. 1, p. 70–81, jan. 2002. ISSN 1536-1268. Available from Internet: <http://dx.doi.org/10.1109/MPRV.2002.993146>.

LOPES, J. et al. A middleware architecture for dynamic adaptation in ubiquitous computing. **J.UCS**, v. 20, n. 9, p. 1327–1351, sep 2014.

LOPES, J. L. et al. A model for context awareness in ubicomp. In: **Proceedings of the 18th Brazilian symposium on Multimedia and the web**. New York, NY, USA: ACM, 2012. (WebMedia '12), p. 161–168. ISBN 978-1-4503-1706-1. Available from Internet: <http://doi.acm.org/10.1145/2382636.2382672>.

LUETH, K. L. **Why the Internet of Things is called Internet of Things**. 2017. Disponível em: < https://iot-analytics.com/internet-of-things-definition/ >. Acesso em janeiro de 2017.

Marin Perianu, R.; Hartel, P.; Scholten, H. **A classification of service discovery protocols**. Enschede, the Netherlands: University of Twente, Centre for Telematica and Information Technology (CTIT), 2005. (CTIT technical report series, TR-CTIT-05-25). Available from Internet: <http://doc.utwente.nl/54527/>.

MATTERN, F.; FLOERKEMEIER, C. From active data management to event-based systems and more. In: SACHS, K.; PETROV, I.; GUERRERO, P. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2010. chp. From the Internet of Computers to the Internet of Things, p. 242–259. ISBN 3-642-17225-3, 978-3-642-17225-0.

MINERAUD, J. et al. A gap analysis of internet-of-things platforms. **CoRR**, abs/1502.01181, 2015. Available from Internet: <http://arxiv.org/abs/1502.01181>.

MUKHOPADHYAY, S. C.; SURYADEVARA, N. K. Internet of things: Challenges and opportunities. In: ____. **Internet of Things: Challenges and Opportunities**. Cham: Springer International Publishing, 2014. p. 1–17. ISBN 978-3-319-04223-7. Available from Internet: <http://dx.doi.org/10.1007/978-3-319-04223-7_1>.

OVIDIU, V. Internet of things strategic research roadmap. **Internet of Things - Global Technological and Societal Trends**, p. 9–52, 2011.

PERERA, C. et al. Context aware computing for the internet of things: A survey. **CoRR**, abs/1305.0982, 2013.

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

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. 2008. W3C Recommendation. Available from Internet: <http://www.w3.org/TR/rdf-sparql-query/>.

RICHARDSON, L.; RUBY, S. **Restful Web Services**. First. [S.l.]: O'Reilly, 2007. ISBN 9780596529260.

TEAM, N. S. D. **Technical Guidance to implement INSPIRE View Services**. 2009.

VERMESAN, O.; FRIESS, P. (Ed.). **Internet of Things: From Research and Innovation to Market Deployment**. Aalborg: River, 2014. (River Publishers Series in Communication).

VERVERIDIS, C. N.; POLYZOS, G. C. Service discovery for mobile Ad Hoc networks: a survey of issues and techniques. **IEEE Communications Surveys & Tutorials**, v. 10, n. 3, p. 30–45, rd 2008. Available from Internet: <http://dx.doi.org/10.1109/COMST.2008.4625803>.

WEISER, M. The computer for the 21st century. **SIGMOBILE Mob. Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 3, n. 3, p. 3–11, jul. 1999. ISSN 1559-1662. Available from Internet: <http://doi.acm.org/10.1145/329124.329126>.

XIN, R. S. et al. Shark: Sql and rich analytics at scale. In: **Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2013. (SIGMOD '13), p. 13–24. ISBN 978-1-4503-2037-5. Available from Internet: <http://doi.acm.org/10.1145/2463676.2465288>.

YAMIN, A. C. **Arquitetura para um Ambiente de Grade Computacional Direcionada às Aplicações Distribuídas, Móveis e Conscientes de Contexto da Computação Pervasiva**. Dissertation (Tese de Doutorado em Ciência da Computação) — Instituto de Informática/UFRGS, Porto Alegre-RS, 2004.

ZHU, F.; MUTKA, M. W.; NI, L. M. Service discovery in pervasive computing environments. **Pervasive Computing, IEEE**, v. 4, n. 4, p. 81–90, 2005.