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

ANDERSON AFONSO CARDOZO

Uma Abordagem de Fog Computing para o Subsistema de Reconhecimento de Contexto e Adaptação do Middleware EXEHDA

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

Orientador: Prof. Dr. Adenauer Correa Yamin

Pelotas
2017

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Cardozo, Anderson Afonso

Uma Abordagem de Fog Computing para o Subsistema de Reconhecimento de Contexto e Adaptação do Middleware EXEHDA / Anderson Afonso Cardozo. – Pelotas: 2017.

107 f.: il.

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

1. Fog. 2. Middleware. 3. IoT. 4. Arquitetura. 5. Sistemas Distribuídos. I. Yamin, Adenauer Correa. II. Título.

“All that is necessary for the triumph of evil is that good men do nothing.”

— Edmund Burke

AGRADECIMENTOS

Gostaria de agradecer a algumas pessoas e instituições que foram essenciais para o meu êxito.

Primeiramente gostaria de agradecer a minha família por oferecer todo o suporte emocional tão necessário durante esta caminhada. A minha namorada Jéssica pelo constante apoio e pela confiança na minha capacidade.

Agradeco ao meu orientador Adenauer Yamin, pelos seus ensinamentos e orientação. Agradeço também a sua disponibilidade, mesmo em finais de semana, sempre disposto a oferecer o suporte necessário a mim.

Agradeço ao Rodrigo Souza pelo apoio e pelo auxílio oferecido, principalmente durante as discussões arquiteturais do meu trabalho.

Aos meus colegas, e amigos, Bianca Cunha e Rafael Ferreira pela troca de experiências e pela amizade desenvolvida durante o mestrado.

A UCPEL, por oferecer toda a infraestrutura necessária para a realização de pesquisa e por me receber novamente como seu aluno. Aos professores do Curso de Engenharia Eletrônica e Computação por toda determinação e suporte que nunca deixaram de oferecer. E por fim a CAPES/PROSUP por me agraciar com uma bolsa de pesquisa que me possibilitou a oportunidade de investir na vida acadêmica.

RESUMO

Pesquisas recentes apontam que em um futuro próximo bilhões de dispositivos inteligentes estarão interconectados através da Internet, atraindo assim a atenção da Indústria e direcionando as pesquisas da comunidade acadêmica. Esta sinergia de investimentos vem contribuindo para a materialização do cenário conhecido como *Internet of Things* (IoT). Na perspectiva da IoT a computação provê informação de todas as "coisas", a todo o momento, independente de localização, constituindo um ambiente altamente distribuído, heterogêneo, dinâmico e com forte interação entre homem e máquina. Para tal, os dispositivos da IoT necessitam ter ciência dos dados contextuais que lhe interessam e quando for o caso reagirem aos mesmos, interoperando de forma autônoma e com o mínimo de intervenção humana possível nos aspectos de gerenciamento. Para o processamento de dados contextuais na IoT tem sido empregadas estratégias baseadas em *Cloud*, as quais tem se provado eficientes no tratamento de aspectos importantes para a IoT, como a facilidade de acesso e disponibilidade. Estas estratégias porém mostram-se vulneráveis para sistemas que possuem limitações nos seus canais com a Internet, assim como para sistemas que necessitam de baixa latência nas respostas ou ainda apresentem chances de desconexão elevadas. Considerando esta motivação, o objetivo central desta dissertação é a concepção de uma arquitetura capaz de prover a aquisição e o processamento de eventos contextuais distribuídos. Para tanto, esta arquitetura, denominada EXEHDA-FOG capacita o *middleware Execution Environment for Highly Distributed Applications* (EXEHDA) o suporte à *Fog Computing*, empregando o processamento distribuído de eventos nas bordas como estratégia de extensão da *Cloud Computing*. Os resultados obtidos com o estudo de caso desenvolvido se mostraram promissores, apontando para continuidade dos esforços de estudo e pesquisa.

Palavras-chave: Fog. Middleware. IoT. Arquitetura. Sistemas Distribuídos.

A Fog Computing approach to the Subsystem Context Recognition and Adaptation of Middleware EXEHDA

ABSTRACT

Recent surveys show that in the near future billions of smart devices will be interconnected via the Internet, thus attracting the attention of Industry and directing the research of the academic community, this synergy of investment has contributed to the materialization of the scenario known as IoT. From the perspective of IoT, computing provides information of all the "things" at all times, regardless of location, providing a highly distributed environment, heterogeneous, dynamic and strong interaction between man and machine. To this end, the IoT devices need to be aware of contextual data that interest you and where appropriate respond to, interoperating autonomously and with minimal human intervention possible in the aspects of management.

For the processing of contextual data in IoT has been used cloud-based strategies, which has proven effective in the treatment of important aspects for the IoT, such as ease of access and availability.

Howere, these strategies are vulnerable to systems that have limitations on their channels to the Internet, as well as for systems that require low latency in responses or present high disconnect chances.

Given this motivation, the central objective of this thesis is the design of an architecture capable of providing the acquisition and processing of distributed contextual events.

Therefore, the proposed architecture, called EXEHDA-FOG provides the middleware EXEHDA support to Fog Computing, using the distributed event processing at the edges as a cloud computing extension strategy. The results obtained with the case studies conducted have shown promising results, leading to the continuity of research efforts.

Keywords: Fog, Middleware, IoT, Architecture, Event-driven.

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
APScheduler	Advanced Python Scheduler
CEP	*Complex Event Processing*
CoAP	*Constrained Application Protocol*
CORE	Common Open Research Emulator
DSMS	*Data Stream Management Systems*
DAC	*Device Application Catalogue*
ECA	Evento-Condição-Ação
EDA	*Event Driven Architecture*
ED-SOA	*Event-Driven Service Oriented Architecture*
EPA	*Event Processing Agent*
EXEHDA	*Execution Environment for Highly Distributed Applications*
FIPA	*Foundation for Intelligent Physical Agents*
G3PD	Grupo de Pesquisa em Processamento Paralelo e Distribuído
GPS	*Global Positioning System*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
MQTT	*Message Queue Telemetry Transport*
MVC	Model-View-Controller
PaaS	*Platform as a Service*
PC	*Personal Computer*

QoS	*Quality of Service*
REST	*Representational State Transfer*
SB	Servidor de Contexto
SC	Servidor de Borda
SMS	*Short Message Service*
SRAC	Subsistema de Adaptação e Reconhecimento do Contexto
SOA	*Service-Oriented Architecture*
UbiComp	Computação Ubíqua
URI	*Uniform Resource Identifier*
XML	*Extensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Middlewares direcionados à sistemas distribuídos vem constituindo foco crescente de pesquisa nos últimos anos. A principal razão disto está associada aos avanços tecnológicos nas áreas de microeletrônica, comunicação de dados e dispositivos embarcados, os quais associados à disseminação da Internet potencializaram aspectos de mobilidade, conectividade e interoperação entre os dispositivos computacionais levando a computação a uma nova era, a da Computação Ubíqua (UbiComp). Na perspectiva da UbiComp a computação deve ser provida de forma transparente para o usuário, sendo incluída nas rotinas do seu dia a dia. A disseminação da UbiComp combinada com os avanços tecnológicos na miniaturização de dispositivos embarcados e reduções no seu consumo energético e nos custos de fabricação, estão mudando a forma como enxergamos o uso cotidiano da tecnologia, e vem criando novas formas de interação entre dispositivos e seres humanos, o conjunto destas novas formas de interoperar diferentes tecnologias através da Internet constitui uma frente conceituada como IoT.

A Internet das Coisas se caracteriza por um conjunto de objetos ou dispositivos interconectados à Internet, unicamente identificados, e com capacidade para coleta de informações e comunicação entre si e/ou com os usuários, independentemente de localização, rede, dispositivo ou tecnologias empregadas, o que o torna a IoT bastante associada aos preceitos da UbiComp. Esta associação, que aproveita a ubiquidade inerente à infraestrutura computacional distribuída provida pela Internet, faz com que a IoT esteja ganhando dimensão enquanto uma abordagem prática para a materialização da UbiComp (WANT; SCHILIT; JENSON, 2015).

As aplicações introduzidas pelo cenário da IoT, enquanto ubíquas são processadas em um ambiente computacional com organização largamente distribuída, e tem como requisito um comportamento autônomo minimizando o envolvimento dos seus usuários. Para atender este requisito estas aplicações necessitam ter ciência dos dados contextuais que lhe interessam e, quando for o caso, reagirem aos mesmos. Esta classe de sistemas computacionais, reativos ao contexto, abre perspectivas ao desenvolvimento de aplicações mais ricas, elaboradas e complexas, e que exploram a natureza dinâmica das modernas infraestruturas computacionais, bem como a mobilidade do usuário (BOTTA et al., 2016).

Neste cenário de milhares de dispositivos inteligentes produzindo dados contextuais, a *Cloud Computing* surge como uma viabilizadora da IoT, no sentido de prover poder de processamento e armazenamento sob demanda (HASHEM et al., 2015). Porém, alguns desafios de infraestrutura são originados desta comunicação de dados entre a *Cloud* e as bordas, onde os dados são coletados. Torna-se essencial a utilização de técnicas de filtragem e fusão de dados

antes de enviá-los para *Cloud*, visando otimizar o emprego dos recursos de rede, recursos de armazenamento, e em casos de conexões com canais de baixa velocidade minimizar sua chance de sobrecarga. Em alguns casos também é necessária uma política de atuação sobre o ambiente Ubíquo, baseada nos dados coletados, com atuação próxima ao tempo real. Para esta proposta, em que o esforço para ciência de contexto também acontece nas bordas computacionais dá-se o nome de *Edge-of-Cloud* ou, mais usualmente, *Fog Computing* (PRIVAT et al., 2016). Como premissa o Fog Computing prevê uma gerência autônoma dos recursos envolvidos, com regras disparadas a partir dos eventos que acontecem nos equipamentos de borda envolvidos.

1.1 Motivação e Objetivos

Com a disseminação da IoT dispositivos vem sendo interconectados à Internet em todo o mundo, a taxas cada vez maiores. Segundo (SRI Consulting Business Intelligence, 2008), em um futuro próximo, a expectativa é que qualquer objeto do dia a dia poderá ser integrado à IoT.

Desta forma, qualquer objeto poderá ser capaz de gerar dados e eventos dos mais diversos tipos na infraestrutura computacional distribuída provida pela IoT. Portanto, há uma necessidade crescente no gerenciamento da aquisição destes dados e seus eventos associados, identificando os mesmos com uma semântica adequada ao seu processamento (FONSECA; FERRAZ; GAMA, 2016).

Uma estratégia utilizada para gerenciar os dados gerados é a utilização middlewares baseados em *Cloud*, porém nem sempre todos os dados coletados são relevantes, podendo serem processados localmente, nas bordas computacionais, através de políticas de filtragem e/ou agregação visando poupar recursos de rede e armazenamento (BONOMI et al., 2014). Visando ainda uma melhor capacidade de agregação e abstração, dispositivos de borda podem requisitar dados de diferentes dispositivos em um mesmo espaço geográfico colaborando para o processamento contextual, materializando assim as contribuições da *Fog Computing* (PRIVAT et al., 2016).

Considerando esta motivação, o objetivo central desse trabalho, denominado EXEHDA-FOG, é a concepção de uma arquitetura capaz de lidar com a aquisição e processamento distribuídas de dados contextuais na IoT, explorando interoperabilidade entre as bordas computacionais.

A concepção desta arquitetura considerou o esforço de estudo e pesquisa em andamento no grupo, denominado CoIoT (Context + IoT) (SOUZA et al., 2016), o qual tem por objetivo prover Ciência de Contexto na IoT.

Neste sentido para o EXEHDA-FOG, foram desenvolvidos mecanismos considerando as demandas de distribuição e escalabilidade da IoT, bem como uma premissa colaborativa entre os agentes para coleta das informações contextuais. Para tanto, é empregada na concepção da abordagem colaborativa, uma estratégia para gerenciamento de eventos distribuídos.

A concepção destes mecanismos teve como premissa a sua integração à arquitetura de software do middleware EXEHDA (LOPES et al., 2014), contribuindo especificamente ao seu Subsistema de Adaptação e Reconhecimento do Contexto.

- sistematizar trabalhos científicos em relação ao tema de pesquisa proposto, caracterizando o estado da arte na área;
- prover suporte no EXEHDA a uma abordagem distribuída e colaborativa na ciência de situação orientada a eventos;
- capacitar a arquitetura de software do Subsistema de Reconhecimento de Contexto e Adaptação para detecção de eventos de forma distribuída mais próxima das bordas computacionais;
- realizar padronização da forma de interoperação entre os componentes da arquitetura de software do middleware EXEHDA, de forma a garantir as funcionalidades previstas, bem como considerando as tendências internacionais;
- conceber a infraestrutura de hardware e software necessária para aquisição e processamento dos eventos disponibilizando-os aos componentes do middleware;
- divulgar na comunidade científica os resultados atingidos pela pesquisa, por meio de publicações de artigos em eventos da área.

1.2 Estrutura do Texto

O texto é composto por 6 capítulos. No capítulo 1 são apresentados a motivação e objetivos que caracterizam a proposta desta dissertação de mestrado. Tópicos referentes as áreas de interesse como a Internet das Coisas, Computação Ciente de Contexto, *Fog Computing*, e sistemas distribuídos baseados em eventos são revisados no Capítulo 2 .

Com o intuito de caracterizar o estado da arte, o Capítulo 3 descreve e analisa alguns trabalhos relacionados. Uma visão geral e os principais aspectos quanto a concepção e mode-

lagem do EXEHDA-FOG , bem como uma discussão dos trabalhos relacionados em relação as estratégias concebidas são apresentados no Capítulo 4.

No Capítulo 5 é apresentado o estudo de caso para avaliação do EXEHDA-FOG, enquanto no Capítulo 6 são apresentadas as principais conclusões, as publicações realizadas e os trabalhos futuros.

No Anexo A são apresentados alguns dos hardwares utilizados na etapa de modelagem do EXEHDA-FOG. No Anexo B são apresentadas algumas interfaces gráficas desenvolvidas para a aplicação de gerenciamento do EXEHDA-FOG.

2 ESCOPO DO TRABALHO

Neste capítulo são apresentados conceitos identificados como importantes durante a revisão de literatura em relação ao tema proposto, e que constituem a fundamentação teórica desta dissertação de mestrado.

2.1 Internet das Coisas

Esta seção introduz uma visão a respeito da Internet das Coisas, do inglês *Internet of Things - IoT*, bem como resume seus principais desafios de pesquisa.

2.1.1 Visão Geral

O termo IoT foi cunhado por Kevin Ashton em 1998 para descrever a evolução tecnológica e conceitual da Internet a qual seria capaz de "capacitar os computadores com os seus próprios meios de coleta de informações, para que eles possam ver, ouvir e sentir o cheiro do mundo por si mesmos, em toda sua glória aleatória" (ASHTON, 2009). A IoT é um conceito e um paradigma onde um ambiente é composto por uma variedade de "coisas"/objetos inteligentes (*smart objects*) embarcados com dispositivos computacionais capazes de prover conexão à Internet, por meio de esquemas de endereçamentos únicos, bem como funcionalidades de sensoriamento e processamento de dados contextuais que os capacitam a interagir uns com os outros e cooperar de forma a criar novas aplicações e serviços para alcançar objetivos comuns entre eles.

Nesta perspectiva, o paradigma da IoT denota que qualquer "coisa" poderia estar interconectada e ser dotada de certa inteligência, tendo a Internet como principal meio de comunicação para as trocas de informações entre elas (VERMESAN; FRIESS, 2014). Desta forma pode-se dizer que o objetivo da Internet das Coisas é possibilitar que as "coisas"estejam conectadas a qualquer momento, em qualquer lugar, com qualquer coisa ou pessoa idealmente usando qualquer caminho ou rede ou serviço.

A IoT tem despertado o interesse mundial devido ao grande número de dispositivos capazes de se conectar a Internet que vem sendo introduzidos no mercado. É previsto que até 2020 haverão mais de 50 bilhões de "coisas" conectadas acumulando mais de 220 bilhões de conexões intermitentes, o que, por sua vez, fez surgir um novo termo conhecido como Internet

de Tudo, do inglês, *Internet of Everything* (SAVITZ, 2012).

Em relação as caracterísiticas que diferenciam a Internet das Coisas da Internet "normal", alguns atributos chaves foram discutidos pelo *Framework SENSE* (JANKOWSKI et al., 2014) e são apresentados na Tabela 2.1. As características de cada atributo constituem requisitos necessários para viabilizar o cenário da IoT e vem direcionando o rumo do desenvolvimento tecnológico de pesquisas na área.

Tabela 2.1 – *SENSE*: Fazendo Sentido (*SENSE*) da Internet das Coisas
Fonte: Adaptado de (JANKOWSKI et al., 2014)

S-E-N-S-E	O que a Internet das Coisas faz	Como se difere da Internet
***S**ensing*	Se utiliza de sensores conectados às 'coisas' (Ex. Temperatura, umidade, etc)	Mais dados são gerados pelas 'coisas' do que por pessoas
***E**fficient*	Adiciona inteligência a processos manuais (Ex. Desligar o regador quando for chover)	Estende o ganho de produtividade da Internet para as 'coisas'
***N**etworked*	Conecta objetos a rede (Ex. Carros, termostatos, etc.)	Parte da inteligência move-se da *Cloud* para as bordas computacionais (Computação em Névoa - *Fog Computing*)
***S**pecialized*	Personaliza tecnologias e processos para verticais específicas (Ex. Saúde, Agricultura, Etc.)	Ao contrário do amplo alcance horizontal de *Personal Computer* (PC)s e *smartphones*, a IoT é mais fragmentada
***E**verywhere*	Implantada pervasivamente (Ex. No corpo humano, nos carros, cafeteiras, etc.)	Presença ubíqua, resultando em uma ordem de magnitude de vários dispositivos e desafios de segurança

2.1.2 Desafios de Pesquisa em IoT

A materialização da IoT vem se tornando possível devido a avanços tecnológicos na miniaturização dos dispositivos embarcados, no desenvolvimento de novos sensores, atuadores e etiquetas inteligentes, bem como na infraestrutura da Internet que provê aos dispositivos da IoT sua gerência e interoperação independente de localização.

Porém, da mesma forma que as tecnologias possibilitadoras de um paradigma auxiliam na concepção desse, elas também introduzem diversos desafios a serem resolvidos.

Outrossim, devido a IoT ser uma área de pesquisa relativamente nova e bastante ampla, esta foi dividida em três diferentes visões de pesquisa direcionando seus focos de forma a endereçar desafios específicos. As três visões são relacionadas aos termos que dão o nome a este paradigma: Internet, Coisas ou Internet das Coisas (ATZORI; IERA; MORABITO, 2010), (GUBBI et al., 2013a).

- Visão orientada à **Internet**: possui foco de pesquisa do ponto de vista de redes. Pesquisas direcionadas a esta visão procuram criar modelos e técnicas para a interoperabilidade dos dispositivos em rede;

- Visão orientada às **Coisas**: objetiva a integração dos objetos (coisas) presentes em um cenário IoT. Pesquisas nesta visão procuram apresentar propostas que garantam o melhor aproveitamento dos recursos dos dispositivos e sua comunicação;

- Visão orientada à **Semântica**: esta visão parte de uma análise semântica da expressão composta "Internet das Coisas". Portanto, possui foco de pesquisa do ponto de vista da comunicação e troca de informações entre dispositivos distintos. Trabalhos nesta visão apresentam propostas que estão focadas na representação, armazenamento, interconexão, pesquisa e organização da informação gerada na IoT, buscando soluções para a modelagem das descrições que permitam um tratamento adequado para os dados gerados pelos objetos.

Estas visões de pesquisa, se desdobram em diferentes desafios, os quais estão correlacionados na Figura 2.1.

Dentre estes desafios, alguns são mais atinentes a pesquisa desenvolvida nesta dissertação e estão caracterizados nas seções a seguir, porém é importante notar que há uma sinergia entre eles. Na sistematização das suas principais características, foram empregados os trabalhos (VERMESAN; FRIESS, 2015)(PIRES et al., 2015a)(DELICATO; PIRES; BATISTA, 2013)(GUBBI et al., 2013b)(MIORANDI et al., 2012).

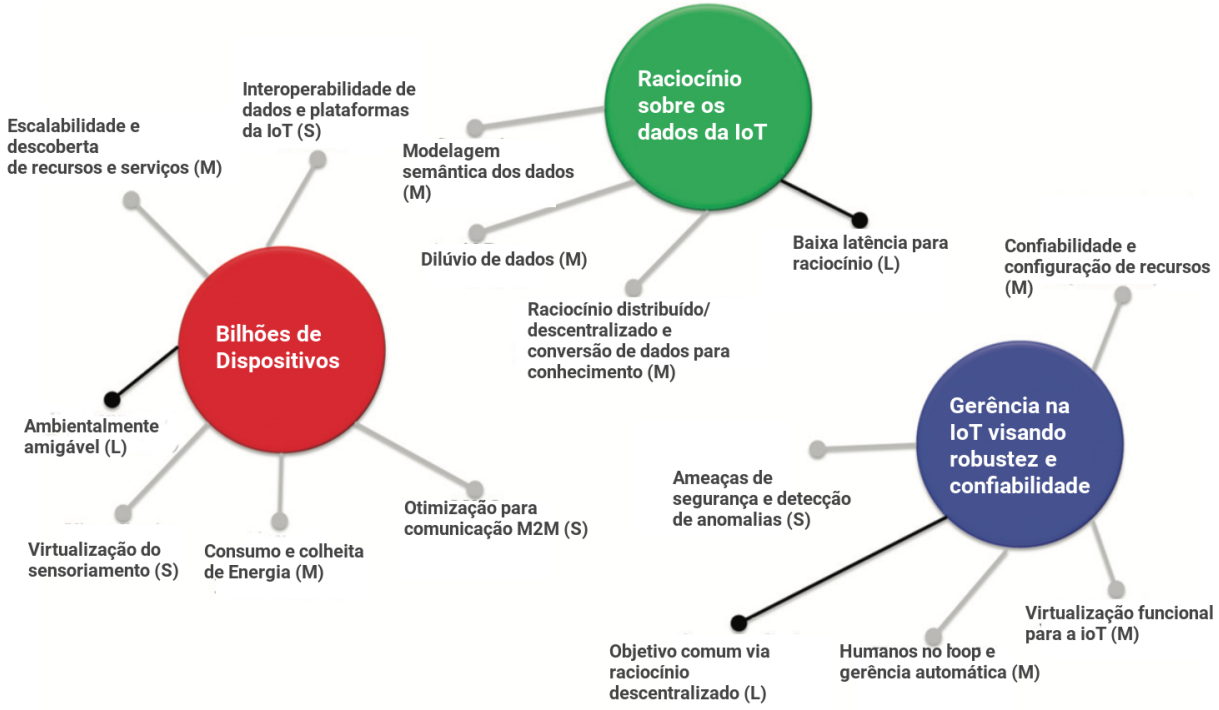


Figura 2.1 – Desafios da IoT
Fonte: Adaptado de IERC (VERMESAN; FRIESS, 2015)

Escalabilidade

Como consequência do crescente número de dispositivos sendo conectados, as plataformas de middleware para IoT devem atender ao requisito de escalabilidade, i.e., possuírem capacidade para suportar requisições provindas de inúmeros dispositivos, funcionando corretamente, mesmo em situações de uso intenso. Soluções de *Cloud Computing* vem sendo utilizadas para este fim, devido à sua facilidade de provisão e uso de recursos computacionais, que podem ser alocados e liberados sob demanda, proporcionando o surgimento da chamada "Nuvem das Coisas" (em inglês, *Cloud of Things*) (PARWEKAR, 2011).

Latência de raciocínio e Dilúvio de Dados

Estes dois desafios se interligam no que diz respeito ao tratamento de dados mais perto dos *sources*, origem dos dados sensoreados. A habilidade de processar os dados mais próximo do ponto de coleta provê uma capacidade de manter a inteligência do sistema ativa mesmo em momentos de desconexões da Internet e, também, provê uma latência menor no processamento das informações contextuais.

O Dilúvio de dados pode ser evitado de forma que as informações vão sendo processadas

perto do ponto de coleta, facultando ao sistema tomar a decisão se a informação necessita ou não ser encaminhada para outro nível do sistema, diminuindo assim o uso de recursos da rede. Entre possíveis soluções para estes desafios podem ser citados o uso de agregação de dados, processamento de fluxos de dados e processamento de eventos complexos (CEP).

Raciocínio distribuído/descentralizado e Conversão de Dados para conhecimento

Este desafio diz respeito ao emprego de técnicas para se distribuir a capacidade do ambiente ubíquo no reconhecimento do contexto e nas atuações sob o mesmo. A IoT tem sido apontada como a viabilizadora de diversas aplicações inteligentes no futuro, sendo assim são necessárias técnicas que possibilitem transformar os dados coletados em conhecimento sob os ambientes, comprimindo estes dados de forma a reduzir latência e aumentando a qualidade da informação.

Objetivo comum via Raciocínio distribuído/descentralizado

Complementando o desafio anterior, as bordas computacionais devem ser capazes de se utilizar do seu raciocínio distribuído visando a coordenação e colaboração nas tomadas de decisões locais em busca de um "objetivo comum".

2.2 Ciência de Contexto

A Ciência de Contexto refere-se a um modelo de computação no qual o sistema computacional é capaz de verificar as características do meio no qual está inserido e que sejam do seu contexto de interesse e, quando necessário, reagir as suas alterações. No cenário da IoT a Ciência de Contexto possui grande destaque, uma vez que existe a real necessidade do sistema tomar decisões autônomas nos mais diversos tipos de dispositivos conectados, via especificações do usuário ou do próprio sistema.

Porém, para a construção de sistemas cientes de contexto em ambientes altamente distribuídos, como o da IoT, alguns desafios devem ser tratados (BELLAVISTA et al., 2012): (i) aquisição do contexto a partir de fontes heterogêneas e distribuídas; (ii) processamento dos dados contextuais adquiridos e a respectiva atuação sobre o meio físico; e (iii) disponibilização distribuída dos dados contextuais processados.

Um sistema ciente de contexto é usualmente organizado em três componentes: aquisição, modelagem e processamento. Uma descrição resumida destes componentes é apresentada

a seguir.

2.2.1 Aquisição do Contexto

A aquisição das informações contextuais é usualmente realizada por meio de um conjunto de sensores heterogêneos, não limitando-se apenas aos dispositivos que coletam variáveis físicas do ambiente, podendo também ocorrer a partir de qualquer fonte de dados capaz de fornecer informações contextuais utilizáveis. Em relação aos tipos de sensores, estes podem ser classificados em três grupos (BALDAUF; DUSTDAR; ROSENBERG, 2007):

- **Sensores físicos**: é o tipo de sensor mais utilizado. Trata-se de dispositivos capazes de capturar grandezas físicas diretamente do ambiente. Exemplos deste tipo de sensores são: Sensores de movimento, temperatura, localização, entre outros;

- **Sensores Virtuais**: neste tipo de sensor as informações são provenientes de softwares ou serviços. Como exemplo pode-se citar uma identificação de atividade de usuário através de um software de monitoramento dos movimentos do mouse ou das entradas do teclado;

- **Sensores lógicos**: este tipo de sensor disponibiliza informações contextuais geradas através da combinação de dados provenientes de diversas fontes, como sensores físicos, virtuais ou informações armazenadas em bancos de dados. Por exemplo, um sensor lógico poderia fornecer a localização de um usuário utilizando a posição do *Global Positioning System* (GPS) do seu *smartphone* combinado com informações de login em suas contas de e-mail armazenadas em banco de dados.

Em ambientes IoT a quantidade de dispositivos pode atingir números elevados (WANT et al., 2010). Portanto, o desenvolvimento de sistemas computacionais para esses cenários deve adotar estratégias que promovam a escalabilidade. Por isso o momento correto da coleta das informações contextuais deve considerar a natureza do elemento monitorado para que não se façam requisições desnecessárias. Para apoiar essa decisão, algumas técnicas de coleta são apontadas por (PERERA et al., 2014a), as quais são classificadas tanto quanto a responsabilidade, quanto a frequência.

Quanto à responsabilidade, o processo de coleta das informações de contexto pode ocorrer de duas formas.

- **Pull**: nessa situação, o procedimento de coleta da informação contextual é disparado a partir do middleware, o qual faz uma consulta diretamente ao sensor. Nesta estratégia de coleta, o software que gerencia o comportamento do sensor não precisa realizar computações elaboradas, pois não cabe ao mesmo avaliar o momento oportuno de realizar a coleta;

- **Push**: nesse caso o procedimento de coleta é disparado a partir do sensor, o qual tem a iniciativa de enviar os dados capturados para o middleware através de um processo de publicação. Esta estratégia proporciona um processo de aquisição de maneira reativa, podendo ser disparada através de eventos do ambiente.

A coleta das informações contextuais também pode ser classificada em dois tipos, quanto à frequência de aquisição.

- **Instantânea**: esse tipo de frequência de coleta é disparada através de eventos do ambiente e acontece instantaneamente. Como exemplos de eventos que podem disparar procedimentos de coleta de informações de contexto, tem-se: a abertura de uma porta, o acender de uma luz, uma temperatura atingindo um determinado valor, entre outros. Essa frequência de coleta associada ao método *push* promove uma reação rápida e mostra-se conveniente em eventos críticos onde se necessita uma tomada de decisão rápida;

- **Periódica**: esse tipo de frequência de coleta obedece a períodos temporais especificados de acordo com as necessidades da aplicação do usuário. O período de tempo pode estar associado a um horário e dia específico, por exemplo, uma informação contextual sendo coletada de segunda a sexta às 19:00hs. A coleta também pode obedecer a intervalos periódicos de tempo, por exemplo, uma informação contextual sendo coletada a cada 30 segundos. As duas diferentes frequências de coleta (instantânea e periódica) podem ser implantadas tanto através do método *pull* quanto do método *push*.

2.2.2 Modelagem do Contexto

Devido a significativa quantidade de informação capturada e acessada em um modelo de representação de contexto há um crescente estudo em técnicas de representação destas informações contextuais.

Na modelagem de contexto deve-se conceber um modelo de entidades do mundo real, suas propriedades, estado de seu ambiente e situações que podem ser usados como referência para a aquisição, interpretação e raciocínio de informações contextuais (KNAPPMEYER et al., 2013). A modelagem das informações de contexto reduz a complexidade das aplicações cientes do contexto, facilita o acesso às informações realizando buscas de forma eficiente, além de melhorar a capacidade de manutenção e de evolução da aplicação.

Existem diversos modelos para representação do contexto (PERERA et al., 2013), tais como: chave-valor, linguagem de marcação, gráfico, orientado a objetos, lógico, ontológico. Modelos híbridos de modelagem de contexto são considerados os mais promissores, pois combinam diferentes técnicas de modelagem, com diferentes níveis de interpretação, para diferentes aspectos.

2.2.3 Processamento do Contexto

O processamento do contexto abrange prismas com relação à interpretação, agregação, armazenamento, consulta e inferência das informações contextuais que foram obtidas a partir de uma etapa de aquisição de contexto, visando posibilitar a compreensão dos contextos de interesse através da geração de informações mais significativas. De modo mais específico, a interpretação de contexto consiste em um conjunto de métodos e processos que realizam a abstração, o mapeamento, a manipulação, a agregação, a derivação, a inferência e demais ações sobre as informações contextuais, com o propósito de facilitar o entendimento de um determinado contexto pelas aplicações e auxiliá-las na tomada de decisão. O processo de interpretação de contexto consiste na manipulação e refinamento das informações contextuais de um ambiente.

Este processo de interpretação de contexto pode ser complexo, como realizar a inferência de humor de um usuário baseado no perfil desse e na ativdade sendo realizada ou então uma interpretação direta como localizar uma cidade através de suas coordenadas geográficas. Além disso, o ambiente da Computação da IoT é extremamente dinâmico e as informações contextuais podem estar distribuídas em diferentes locais ou produzidas por dispositivos com alto grau de mobilidade. Essa complexidade faz com que exista a necessidade de um suporte computacional às aplicações, de maneira a auxiliá-las na realização de interpretações de contextos.

Desta forma, as atividades de processamento de contexto necessitam de abstração das aplicações e, portanto, um módulo interpretador torna-se um componente essencial em uma plataforma de suporte a tais aplicações. Esse deve ser capaz de obter e prover informação

contextual em diferentes níveis de abstração, conforme o desejo do usuário e de suas aplicações. Uma aplicação pode necessitar tanto de informações brutas, de mais baixo nível como de informações mais abstratas e elaboradas, de mais alto nível, provenientes de um processo de refinamento e interpretação (COSTA; YAMIN; GEYER, 2008).

Um dos grandes desafios na utilização de informações contextuais consiste na obtenção de contextos significativos, a partir de um conjunto de informações dispersas e desconexas, obtidas por mecanismos heterogêneos de aquisição. Para isso, devem ser disponibilizadas funcionalidades de processamento e raciocínio sobre a informação contextual.

Os termos raciocínio e inferência são geralmente utilizados para indicar processos pelos quais deduções a respeito dos dados contextuais são alcançadas. O projeto e implementação de um mecanismo para raciocínio de contextos pode variar bastante a depender do tipo do conhecimento contextual envolvido. A priori, o processamento do contexto deve ser implementado separadamente do comportamento da aplicação do usuário e não embutido no código da mesma (KNAPPMEYER et al., 2013).

O raciocínio é utilizado para verificar a consistência do contexto e para inferir contexto implícito (alto nível), a partir de contextos explícitos (baixo nível) (Xiao Hang Wang et al., 2004). A consistência é necessária, pois, muitas vezes, a informação contextual adquirida automaticamente pode apresentar ambiguidades.

A necessidade de manter o histórico de dados de contexto é um requisito ligado à aquisição de informações contextuais, bem como à disponibilidade contínua dos componentes de captura destas informações. Um histórico do contexto pode ser utilizado para estabelecer tendências e predizer valores futuros de informações contextuais. Sem o armazenamento dessas informações, análises desse tipo não podem ser realizadas.

2.2.4 Ciência de Situação

A Ciência de Situação é tida como uma particularização da Ciência de Contexto, onde situações são vistas como contextos logicamente ligados (FLEISCHMANN, 2012). De forma geral, as situações fornecem significado aos dados coletados pelos sensores oferecendo maiores informações para as aplicações e usuários na tomada de decisões ou ações.

Desta forma, entende-se por Ciência de Situação a percepção de uma ou mais situações com relação ao tempo ou espaço, a compreensão do seu significado e a projeção do seu estado depois da mudança de alguma variável, como, por exemplo, um evento pré-determinado (ENDSLEY, 1995) (DUTT; GONZALEZ, 2012). Um melhor detalhamento sobre estas etapas é reali-

zado a seguir.

- **Percepção**: relaciona-se ao monitoramento, detecção e reconhecimento de um ou mais elementos do ambiente, como alertas ou leituras de sensores e seus estados atuais (locais, condições, ações, quando ocorreram).
- **Compreensão**: realiza a síntese e correlação dos eventos desconexos identificados na fase de percepção através de diferentes técnicas;
- **Projeção**: a fase de projeção pode ser descrita como a habilidade do sistema de antecipar eventos ou situações futuras a partir da compreensão da situação do ambiente atual. Desta forma, após a percepção e compreensão do ambiente o sistema pode determinar se estas situação afetarão o futuro do sistema.

Existem diversas técnicas utilizadas na fase de compreensão e projeção do ambiente, uma das mais utilizadas é o processamento de eventos, o qual possui a capacidade de agregar diversos eventos de forma a inferir que um evento complexo está acontecendo, oferecendo assim a capacidade de agir de acordo com isto ou antecipar algum evento futuro (PRIVAT et al., 2016).

2.3 Fog Computing

A *Cloud Computing* é uma das tecnologias mais promissoras para a materialização da IoT (VERMESAN; FRIESS, 2014) por viabilizar sistemas escaláveis ao fornecer uma grande capacidade de armazenamento de dados e processamento. Apesar destas características, soluções baseadas exclusivamente na *Cloud* necessitam de uma conexão ativa a internet visando o processamento dos dados contextuais e atuações sob o meio, o que torna necessário uma abordagem que garanta a confiabilidade operacional do sistema mesmo em eventuais desconexões. Com esta motivação surge o conceito de *Fog Computing* (STOJMENOVIC; WEN, 2014).

2.3.1 Principais Características

Com os dispositivos da IoT cada vez mais presentes no dia a dias das pessoas, dificilmente um paradigma baseado puramente em *Cloud* poderá satisfazer as necessidades de baixa latência, ciência de localização e confiabilidade (STOJMENOVIC; WEN, 2014). *Fog Computing*, ou computação em nevoeiro, surgiu de maneira a enfrentar esses desafios (BONOMI,

2011). A *Fog Computing* é implementada nas bordas computacionais, de certa forma considerada uma *Cloud* mais próxima do chão, provendo assim uma baixa latência, ciência de localização e aprimorando a *Quality of Service* (QoS). Além disto, esta provê armazenamento e uma capacidade de "raciocínio" para as bordas computacionais, a fim de auxiliar na tomada de decisões diretamente onde os dados são coletados. Este paradigma também é capacitado a lidar com coleta de dados altamente distribuída, uma importante característica da IoT (AAZAM; HUH, 2014).

Os nodos da *Fog Computing* devem possuir um poder computacional e capacidade de armazenamento de forma a lidar com as requisições dos usuários locais ou externos. A seguir são apresentadas algumas características inerentes da *Fog Computing* (BONOMI et al., 2012) (HONG et al., 2013):

- Proximidade da borda: possui a localização dos serviços mais próximos dos usuários finais, consumidores, auxiliando na diminuição de latência e acesso a dados. Isto é devido ao acesso ser local, não havendo necessidade de acessar um servidor centralizado;
- Organização hierárquica: a organização na *Fog* para suportar uma baixa latência e alta escalabilidade deve ser organizada de forma hierárquica do topo, *Cloud*, para baixo, *Fog*, de forma que o controle e gerenciamento possa ser controlado pelo topo da hierarquia;
- Consciência de localização;
- Acesso predominantemente *Wireless* aos dispositivos da IoT;
- Distribuição geográfica densa, a distribuição é interessante de forma a descentralizar os serviços que normalmente estariam disponíveis apenas na *Cloud*. Desta forma os nodos *Fog* encontram-se distribuídos mais próximos da aplicação;
- Escalabilidade, devido a distribuição geográfica pode-se lidar com uma grande escala de dispositivos;
- Suporte a mobilidade, deve oferecer suporte a entrada e saída de usuários na rede;
- Aplicações em tempo real, com uma latência baixa estas aplicações podem se beneficiar deste modelo computacional com serviços próximos sendo mais vantajosos do que uma abordagem centralizada;

- Heterogeneidade;
- Interoperabilidade, os componentes de uma arquitetura *Fog* devem ser capazes de interoperar e compartilhar recursos e serviços com dispositivos heterogêneos;
- Dinâmica de otimização por usuário, na *Fog* diferentemente do que na *Cloud* os usuários estão na mesma rede, desta forma é possível conhecer a dinâmica local em que o usuário está inserido, podendo assim, ter conhecimento específico sobre qualquer usuário podendo se adaptar a estes;
- Comunicação com a *Cloud*, os nodos devem ser capazes de se comunicar com a *Cloud*, caso necessário.
- Economia de recursos da rede, com a tomada de decisão local, os nodos podem decidir ou não repassar seu conteúdo para *Cloud*, poupando assim recursos de rede e, potencialmente, energia dos dispositivos.

2.3.2 Visão da Fog Computing

A *Fog Computing* não visa substituir a *Cloud Computing*, e sim, extendê-la para que sejam utilizadas em conjunto para as aplicações de IoT. A Figura 2.2 exemplifica esta distribuição da arquitetura na qual as responsabilidade são compartilhadas, assim como os dados e eventos gerados.

Na perspectiva da *Fog Computing* a computação deve ser realizada de maneira distribuída, visando-se obter uma otimização no resultado da análise dos dados contextuais e, também, de se utilizar os recursos do ambiente da melhor maneira possível. Desta forma, deve-se planejar a arquitetura e as aplicações para que estas possam enviar os dados contextuais aos locais adequados, de acordo com a sua necessidade.

Os dados mais críticos, que não possuem a necessidade de uma análise histórica, porém necessitam serem tratados quando da sua ocorrência deverão serem analisados nas bordas, podendo esta análise ocorrer por dados oriundos de diferentes nodos *Fog*. Posteriormente a esta análise e a uma possível tomada de decisão que pode gerar uma atuação imediata, os dados são enviados para a *Cloud* a fim de serem armazenados.

Os dados que necessitam de uma análise histórica ou de grande poder de processamento para inferência de situações e contextos devem ser processados em nodos centralizados,

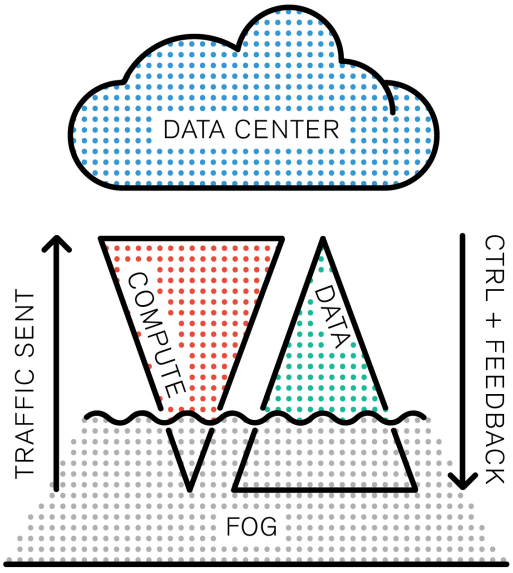


Figura 2.2 – Visão da Fog Computing
Fonte: (LANGTON, 2015)

na *Cloud*. A *Cloud* também pode alterar as regras dos nodos *Fog* sob demanda, ou mesmo, utilizando-se de métodos de aprendizagem em algumas situações.

2.4 Sistemas Distribuídos Baseados em Eventos

Os sistemas distribuídos baseados em eventos tem desempenhado um importante papel na computação distribuída e, com a popularidade da IoT, sua influência vem crescendo continuamente (HELMER; POULOVASSILIS; XHAFA, 2011). Esses sistemas tem sido utilizados em uma grande variedade de aplicações como monitoramento de ambientes, disseminação de informações, finanças, jogos onlines, ambientes assistidos de vivência, e etc.

Além disso, a utilização de um modelo de processamento distribuído de eventos possibilita um ganho significativo em escalabilidade e em confiabilidade. Os eventos que ocorrem em um área específica, ou mais ampla, podem ser filtrados e agregados localmente antes de serem enviados para o seu destino, por exemplo para a *Cloud* ou para outros agentes responsáveis pelo processamento de informações sobre os eventos e reagir conforme necessário. Desta forma, o processamento de eventos pode ser visto como um meio da materialização da *Fog Computing*.

Nestes sistemas um evento pode assumir diversas formas, mas todos estes eventos podem ser definidos como uma abstração de observações realizadas em um determinado ambiente,

ou uma observação que poderá ocorrer em um momento (ETZION; NIBLETT, 2010), ou então, alguma alteração de forma significativa para este ambiente (MACEDO; MARINHO; SANTOS, 2015). Como exemplo de um evento complexo pode-se citar o pouso de um avião, o ato de pousar em si pode ser considerado um evento complexo, gerado e detectado através de eventos mais simples como a redução de altitude e velocidade, o baixar do trem de pouso e etc.

Na área de processamento de eventos há duas grandes vertentes, sistemas que gerenciam fluxos de dados, *Data Stream Management Systems* (DSMS), e sistemas de processamento de eventos complexos, *Complex Event Processing* (CEP) (CUGOLA; MARGARA, 2012). Os DSMS são baseado no conceito de queries ativas e são evoluções dos Bancos de Dados Ativos, as queries permanecem produzindo resultados sobre um banco de dados na medida em que este é atualizado, e geralmente não são atribuídas semântica aos dados processados e quem as define são os clientes destes eventos. O CEP (LUCKHAM, 2008) é baseado no modelo publicador/subscritor, onde eventos são uma unidade de informação e processamento possuindo uma semântica bem definida. Neste tipo de sistema é possível especificar o relacionamento entre diferentes eventos ao se aproveitar de conceitos como casualidade, temporalidade, sequências, agregações e composições.

Podemos assim caracterizar duas categorias de eventos observados em um ambiente IoT.

- **Eventos Simples**: eventos que ocorrem em um momento específico no tempo, não representando uma abstração de alto nível ou composição de eventos. São eventos simples: Leitura de temperatura, soar um alarme, etc;
- **Eventos Complexos**: eventos que são gerados por um padrão de outros eventos, simples ou complexos, representando uma abstração de mais alto nível. Por exemplo, um evento complexo que indica fogo no prédio pode ser ativado quando ocorrer simultaneamente eventos simples de leitura de uma temperatura acima de um determinado valor e um evento de ativação de um detector de fumaça.

Em um ambiente de IoT os eventos são produzidos sempre que alguma alteração ocorre em um contexto de interesse, como no caso de um sensor atingir um valor especificado ou quando um alarme soar na sala. Estes eventos devem ser processados pelo sistema, podendo também serem utilizados na composição de eventos mais complexos, assim como, notificar as aplicações ou usuários interessados em recebê-los.

Os eventos simples agregados e combinados oferecerem um maior nível semântico, sendo considerados eventos complexos, pois estes eventos simples possuem significados implícitos, os quais via regras complexas podem ser explorados.

2.4.1 Arquiteturas Direcionadas a Eventos

Uma arquitetura direcionada a eventos, *Event Driven Architecture* (EDA), possui grande valia para um ambiente altamente escalável e heterogêneo como os ambientes da IoT. É uma abordagem capaz de oferecer proatividade e inteligência necessária ao ambiente para lidar com qualquer situação que possa ocorrer. Nos ambientes distribuídos os dispositivos capazes de realizar o processamento de eventos contextuais são chamados de *Event Processing Agent* (EPA).

De forma geral pode-se citar dois tipos de estratégias para gerência de eventos em sistemas direcionados a eventos (ETZION; NIBLETT, 2010):

- **Baseados em Regras ou Especificação**: sistemas baseados em regras são os tipos mais comuns. Eles apresentam uma grande capacidade de lidar com eventos e, por seus eventos serem explícitos via regras, possuem um requisito de poder computacional menor do que sistemas baseados em aprendizado, sendo ideais para os sistemas embarcados da atualidade;

- **Baseados em Aprendizado**: os sistemas baseados em aprendizado necessitam de um poder computacional maior para operarem, desta forma há uma necessidade de hardware para detecção de eventos mais complexos. Normalmente sistemas deste tipo são aplicados em um nodo específico da rede.

O modelo mais utilizado no tratamento distribuído de eventos são os modelos baseados em regras (PERERA et al., 2014b). Este modelo oferece uma alta capacidade de raciocínio ao ambiente da IoT aliada a simplicidade de especificação e processamento, capacitando o ambiente de geração de informações contextuais de alto nível a partir de dados contextuais de baixo nível. As regras Evento-Condição-Ação (ECA) são avaliadas no momento em que algum evento é detectado e sua sintaxe básica é dada por: `ON <evento> IF <condição> THEN <ação>`.

No modelo de interação tradicionalmente utilizado de requisição-resposta, como exemplificado na Figura 2.3, o publicador precisa enviar os eventos detectados ou leituras de sensores, para todos os interessados neste evento, ou então, estes interessados devem requisitar esta informação quando achar necessário.

Estas interações síncronas de requisição-resposta são muito utilizadas em arquiteturas de sistemas distribuídos orientados a serviços, porém neste modelo é necessário aguardar uma resposta do serviço para que o produtor do evento tenha seu sistema desocupado.

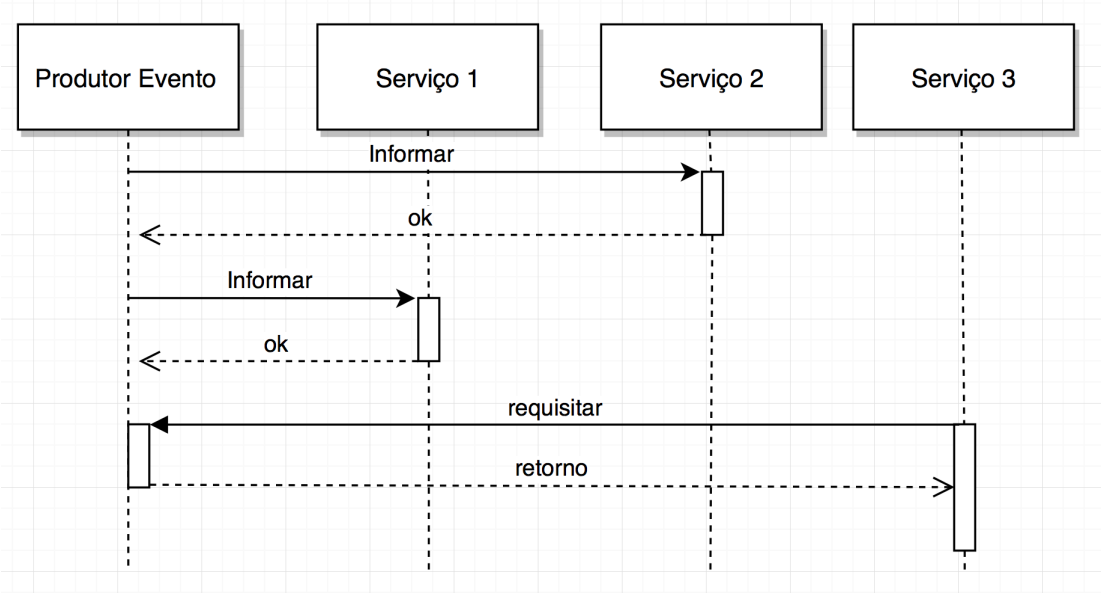


Figura 2.3 – Fluxo do Evento Requisição-Resposta
Fonte: O Autor

Se utilizando de uma abordagem direcionada a eventos pode-se obter um ganho de escalabilidade onde o processamento pode ser realizado de forma assíncrona, além disso a aplicação pode necessitar realizar uma análise de uma grande quantidade de dados para prover informações a outra aplicação e, através de um fluxo de eventos, pode-se distribuir a computação para que nodos realizem partes desta análise em paralelo. Neste princípio de desacoplamento o Produtor do Evento não precisa aguardar a resposta de um serviço que está consumindo um evento para continuar suas funções, na verdade, o Produtor do evento pode, inclusive, desconhecer a quantidade exata de consumidores para o evento que está sendo gerado.

Este tipo de interação é comumente referido como interação do tipo *Push*, neste tipo de arquitetura normalmente há uma relação pré-definida entre publicador/subscritor na qual o subscritor escolhe um tópico ou conteúdo para ser notificado.

Embora sejam tipos diferentes de arquitetura de interação, uma combinação dos dois tipos é muito utilizada para ambientes de IoT, de fato há vantagens em se utilizar uma arquitetura direcionada a eventos e orientada a serviços.

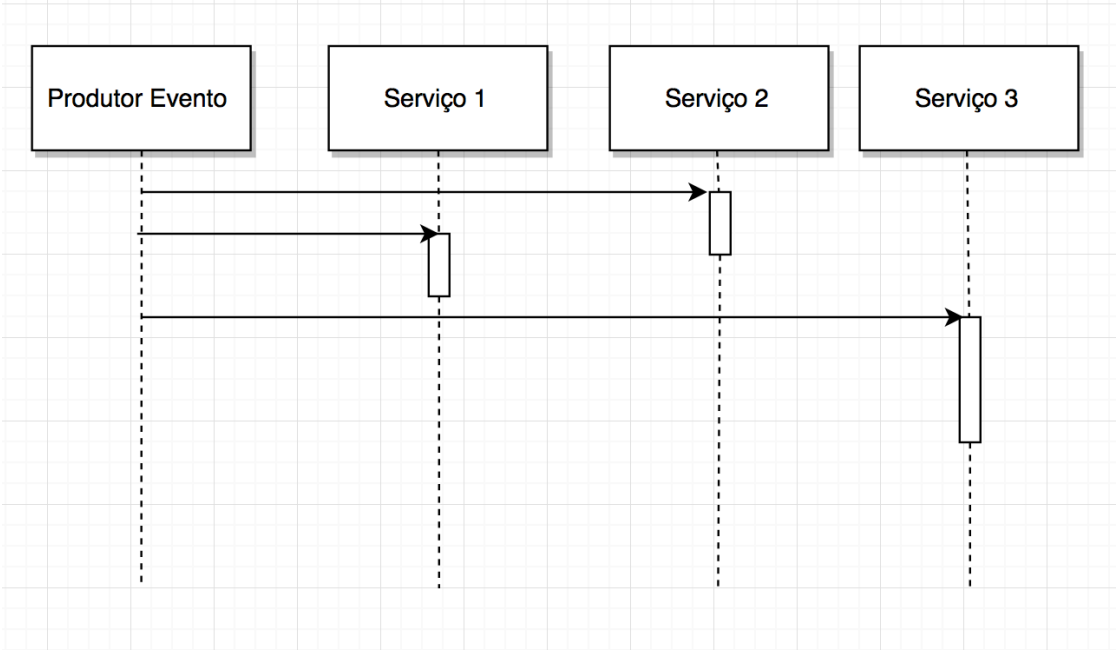


Figura 2.4 – Fluxo do Evento Push
Fonte: O Autor

2.4.2 Contexto Temporal

Por vezes para que determinado evento seja avaliado é necessário que ele ocorra dentro de um determinado período, sequência ou local específico. Com base nesse requisito torna-se necessário um contexto temporal e espacial para tratamento dos eventos.

O contexto temporal pode ser definido como um ou mais intervalos de tempo. Cada intervalo pode ser definido como uma partição de contexto que contêm eventos que ocorreram naquele intervalo (ETZION; NIBLETT, 2010). Desta forma os intervalos, ou janelas temporais, são responsáveis pela detecção e tratamento de eventos que ocorrem apenas dentro delas. Estas janelas temporais podem se sobrepor desde que as regras para detecção assim determinem, desta forma a instância de um evento pertence a janela na qual este foi detectado, porém pode-se compartilhar eventos com outros agentes ou janelas.

As janelas de tempo são definidas através de intervalos com início e fim, ou apenas um destes. Estes limites podem ser definidos através do tempo ou então através de uma detecção de algum evento previamente definido. Essa lógica usa intervalos para descrever eventos específicos e as relações entre os intervalos para descrever as interdependências entre os eventos de forma que dentro de uma janela de tempo podem ocorrer eventos de diferentes naturezas. Na figura 2.5 pode-se observar um exemplo de alguns destes tipos de janelas de tempo.

Após a janela de tempo ser definida pode-se haver partições de tempo dentro desta. Na figura 2.5 pode-se observar casos em que várias partições podem surgir, como no exemplo de

Intervalo de Evento.

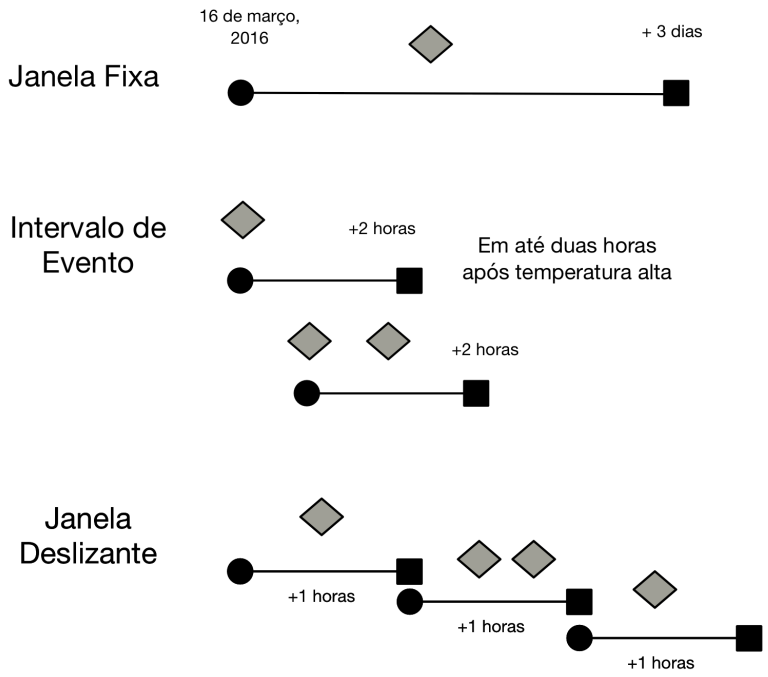


Figura 2.5 – Janelas de Tempo
Fonte: O Autor

Desta forma os tipos principais de janelas de tempo são:

- **Janela Fixa**: No modelo de janela fixa cada janela possui um período fixo de duração, podendo haver apenas uma janela aberta por vez, sem sobreposição. A janela pode se repetir em intervalos de tempo determinado. Exemplo: Monitoramento de um paciente no pós-operatório onde há uma data inicial e uma data final para observação;

- **Intervalo de Evento**: Este tipo de janela é aberto após o tratador de evento receber um determinado evento. Desta forma não há previsão de quando a janela pode ser aberta. Exemplo: Ao ser detectada a temperatura alta de um paciente abre-se uma janela e, em caso de detecção de arritmia dispara-se um alerta para o médico. Vale-se notar que caso não seja detectado o evento posterior dentro da mesma janela esta é descartada;

- **Janela Deslizante**: As janelas deslizantes podem ser de dois tipos, baseadas em tempo ou em eventos. As baseadas em tempo são abertas periodicamente em intervalos de tempo determinados, exemplo de 1 em 1 hora. Nas baseadas em evento as janelas ficam abertas de acordo com a contagem de eventos, por exemplo ao se detectar 3 vezes na sequência uma pressão arterial crescente.

2.4.3 Filtragem e Transformação

Na maioria das vezes os dados coletados em um ambiente Ubíquo possuem pouco valor semântico, sendo necessária se realizar operações sob estes dados para inferência de situações ou conhecimento. Entre os tipos mais importantes de técnicas utilizados nos EPA estão a Filtragem e a Transformação (ETZION; NIBLETT, 2010) (HELMER; POULOVASSILIS; XHAFA, 2011).

Filtragem

Os processos de filtragem visam qualificar os dados coletados, sendo analisados, ou propagados, apenas dados que atendam as especificações do filtro empregado. De maneira geral, de uma filtragem pode-se obter três resultados. Sendo que estes resultados podem ser submetidos a outras filtragens:

- ***Filtered-in***: O teste de avaliação dos dados resultou em sucesso;
- ***Filtered-Out***: O teste de avaliação dos dados resultou em falha;
- ***Non-filterable***: Estes dados não podem, ou não devem, ser filtrados;

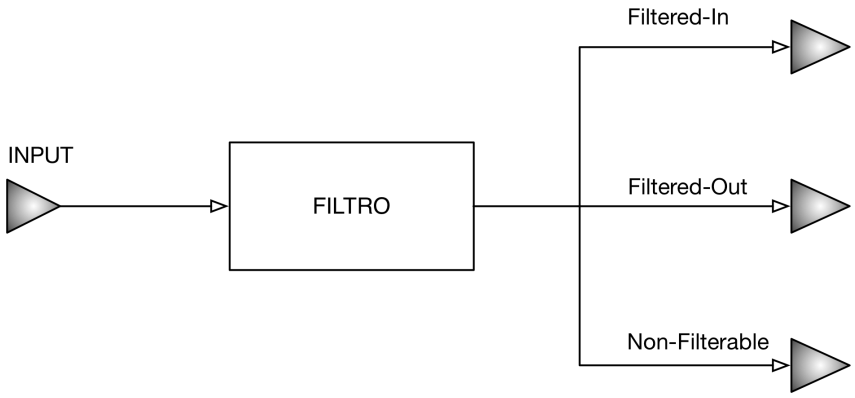


Figura 2.6 – Filtragem de dados
Fonte: Adaptado de (ETZION; NIBLETT, 2010)

Na Figura 2.6, pode-se observar que o bloco de filtragem recebe uma entrada, *Input*, e, com essa informação, poderá realizar as operações de filtragem. Após as operações o dados é encaminhando para o caminho definido pela operação. Este resultado poderá ser utilizado por outro bloco de filtra do EPA de forma a se agregar mais valor ao resultado obtido. Porém, caso

não seja necessária uma saída, ela pode ser simplesmente descartada, juntamente com os dados que a geraram.

De certa forma, as operações de filtragem podem ser utilizadas para redistribuir a informação para realizar uma distribuição no processamento (ETZION; NIBLETT, 2010). De maneira geral, pode-se definir alguns tipos de expressões de filtragem, como exemplificado na Tabela 2.2,

Tabela 2.2 – Tipos de filtragem

Tipo	Descrição
Inclusão de atributo	Verifica se o evento sendo processado possuí um atributo específico)
Valor de atributo	Verifica se um atributo possui valor (diferente de *null*)
Por tipo de dado	Verifica se o dado é do tipo esperado (data, hora, número, string)
Testes lógicos	Testa o valor de um atributo com um valor pré-determinado ou contra o valor de outro atributo. Operações: maior que, menor que, igual, diferente, maior ou igual e menor ou igual.
Conjunção	Expressão é a conjunção de duas, ou mais, expressões. (*AND*).
Disjunção	Expressão é a disjunção de uma ou outra expressão. (*OR*).
Negação	Expressão é a negação de outra expressão. (*NOT*).

Uma característica importante nos procedimentos de filtragem, é a capacidade de manter estados dos eventos que passaram anteriormente pelo EPA, tornando-o capaz de controlar o fluxo dos eventos. Esta característica é comumente chamada de Filtragem de Estado (*Stateful Filtering*, do inglês). Estes tipos de filtros são interessantes para controlar as janelas de tempo dos eventos e, também, podem ser utilizados para decisão de descartar determinados eventos, em sistemas onde isso é possível. A Filtragem de Estado, por seus aspectos, sempre são empregadas juntamente com um contexto temporal e podem ser divididas em três tipos básicos:

- Primeiro X: Passa a primeira instancia do evento X que for detectada na janela;
- Último X: Nesse caso apenas a última instância (mais recente) de um evento X dentro da janela é propagada;
- X aleatório: Este tipo de filtro deixa passar uma quantidade aleatória de instâncias do evento X;

Estas filtragens também podem possuir limites de quantos eventos podem ser propagados por elas.

Transformação

Transformação é a capacidade do EPA de receber um ou mais eventos e manipulá-los de forma a se criar um, ou mais, eventos de saída diferentes. Este tipo de transformação pode ser "sem estado", ou seja, cada evento entrando no EPA é processado individualmente de forma independente de quaisquer eventos anteriores, ou então "com estado", onde um evento de saída pode ser derivado de mais de um evento de entrada e pode haver a noção de sequência de eventos (ETZION; NIBLETT, 2010).

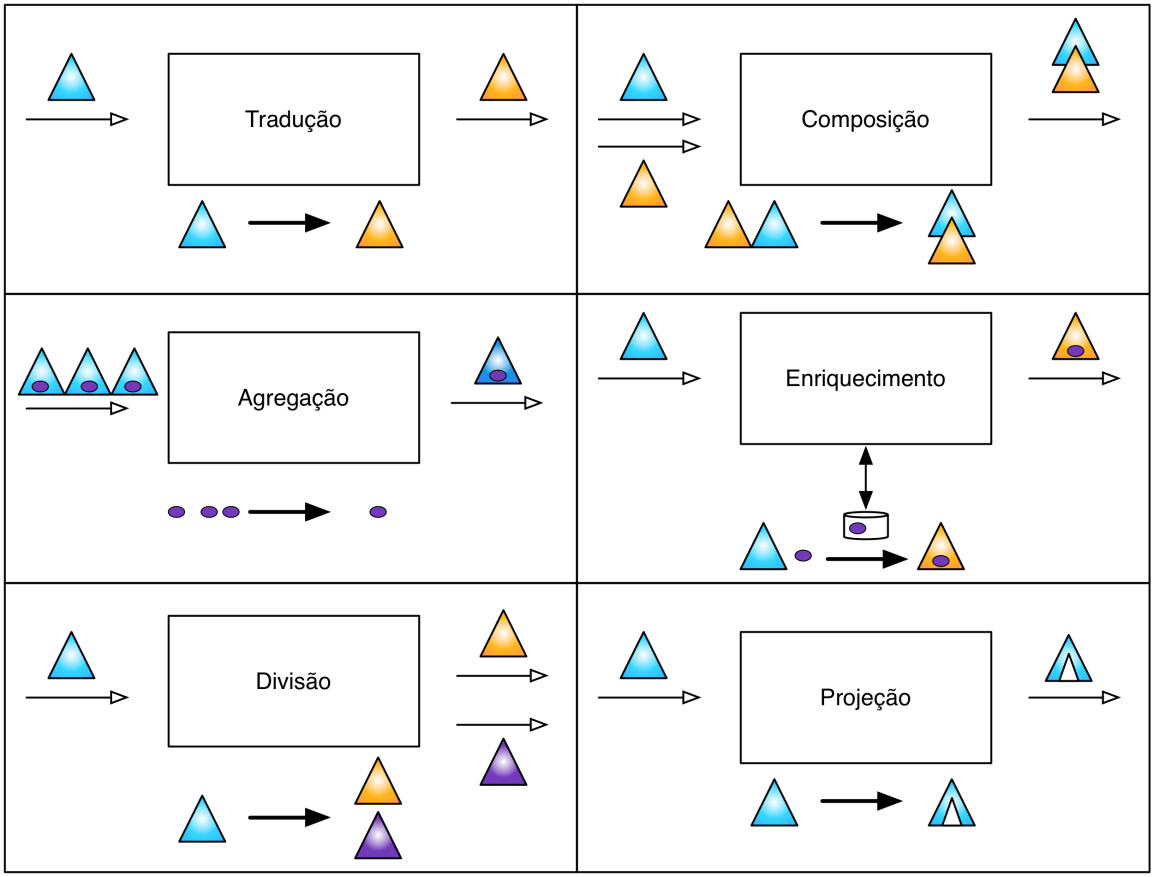


Figura 2.7 – Transformação de Eventos
Fonte: Adaptado de (ETZION; NIBLETT, 2010)

Na literatura encontra-se seis tipos de transformação que podem ser utilizados nos eventos, estes tipos são descritos abaixo e ilustrados na Figura 2.7:

Tradução

Uma transformação de Tradução é uma operação "sem estado"que recebe apenas um evento de entrada e gera um evento de saída baseado em regras de derivação do evento original.

Composição

Uma transformação de composição é uma operação que recebe grupos de eventos de dois tipos distintos e avalia estes com base em um critério de comparação bem definido e cria eventos derivados baseados nestas comparações.

Agregação

A transformação de agregação recebe uma coleção de eventos e cria um evento de saída baseada na coleção de eventos de entrada.

Enriquecimento

A transformação de Enriquecimento recebe a entrada de um evento e, utilizando a informação de uma base de estados, criar um evento derivado que inclui atributos do evento original, podendo conter atributos modificados ou então incluindo novos atributos e informações.

Divisão

Uma transformação de divisão recebe um evento de entrada e cria uma coleção de eventos. Cada evento gerado a partir do evento de entrada pode ser um clone do original ou uma projeção daquele evento contendo uma parte de seus atributos.

Projeção

Um transformação de projeção recebe um evento de entrada e cria um evento de saída derivado, contendo uma parte dos atributos do evento de entrada.

2.5 Considerações do Capítulo

Neste capítulo foram apresentados os conceitos e definições mais encontrados na literatura para a área de IoT e suas frentes de pesquisa. Notou-se diversos desafios ainda a serem enfrentados pela área, principalmente devido as inúmeras naturezas que podem assumir suas aplicações. Também foram sistematizados conceitos sobre Ciência de Contexto e Ciência de Situação, utilizados no tratamento das informações contextuais em ambientes autônomos, como os da IoT.

De forma a colaborar na solução de desafios da IoT foram estudadas a *Fog Computing* e a distribuição de eventos, e como as diferentes técnicas associadas a essas áreas podem auxiliar nas aplicações da IoT, formando assim a base conceitual para suporte a proposta arquitetural desta dissertação.

3 TRABALHOS RELACIONADOS

Considerando os objetivos para o desenvolvimento desta dissertação de mestrado, foi feita uma revisão de literatura, identificando os principais trabalhos na área.

Dentre os trabalhos identificados, foram selecionados quatro tendo em vista os seguintes critérios: (a) modernidade, (b) sua representatividade no cenário nacional e internacional (considerando suas citações) e se o grupo de pesquisa (c) mantém seu desenvolvimento ativo. Além destes aspectos, na seleção também foram considerados os requisitos específicos do EXEHDA-FOG atinentes ao tratamento de eventos em Fog Computing: distribuição, suporte a composição, escalabilidade e gerência dinâmica.

3.1 Projeto WSO2

O projeto WSO2[1] propõe uma série de soluções para a IoT, dentre elas uma arquitetura de referência com o objetivo de auxiliar desenvolvedores quanto a concepção de arquiteturas para o cenário da IoT. A arquitetura de referência é descrita em um *whitepaper* (FREMANTLE, 2015) e tem como objetivo fornecer um ponto de partida eficaz que contemple a maior parte dos requisitos de sistemas e projetos envolvendo IoT.

Em WSO2 são identificados alguns requisitos considerados necessários para soluções IoT, sendo estes classificados em cinco grandes grupos: (i) conectividade e comunicação; (ii) gerenciamento de dispositivos; (iii) obtenção de dados e análise; (iv) escalabilidade; e (v) segurança. Além desses, menciona ainda alta disponibilidade, predição e facilidade de integração.

Com o intuito de contemplar tais requisitos a arquitetura de referência proposta, ilustrada na Figura 3.1, consiste de um conjunto de camadas no qual cada camada executa uma determinada função claramente definida. As camadas previstas são: (i) Comunicações Externas; (ii) Processamento de Eventos e Análises; (iii) Camada de Agregação/Barramento; (iv) Comunicações entre Dispositivos, e; (v) Camada de dispositivos. Há também duas camadas transversais/verticais, que são (i) Gerenciador de Dispositivo e (ii) Gerenciamento de Identidade e Acesso.

Uma descrição das funcionalidades de cada camada é realizada a seguir.

[1] http://wso2.com/

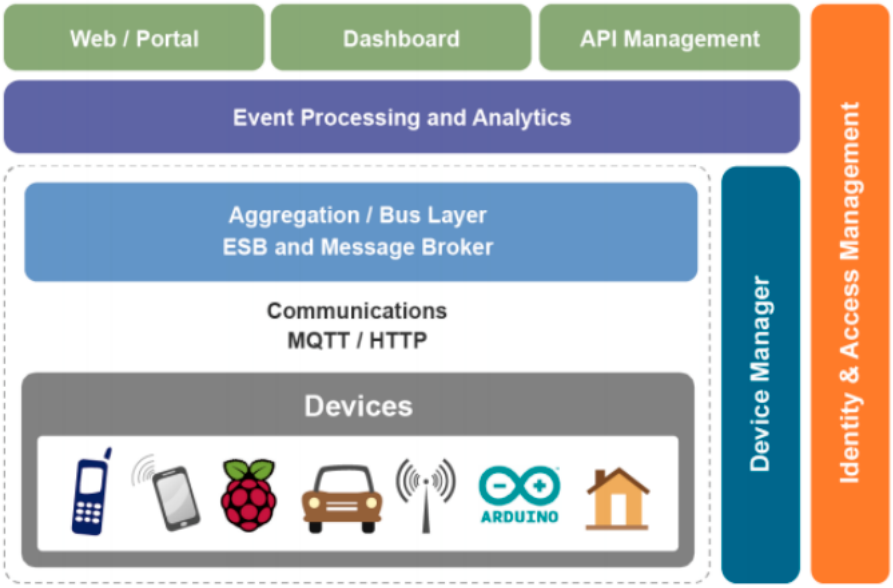


Figura 3.1 – Arquitetura de Referência WSO2.
Fonte: (FREMANTLE, 2015)

- **Camada de Dispositivos**: composta pelos objetos físicos (*devices*), que para serem considerados dispositivos IoT, devem possuir comunicação com a Internet de forma direta ou indireta, bem como um identificador único (local ou global). Além disso, é recomendado que cada dispositivo, possua um *token OAuth2 Refresh/Bearer* como identificador auxiliar, com o intuito de ser repassado a qualquer servidor, ou serviço, que exija um processo de autenticação;

- **Comunicações entre Dispositivos**: objetiva fornecer interoperabilidade via Internet para os dispositivos IoT. A proposta indica o uso de três protocolos de uso consolidado em ambientes IoT: (i) HTTP/HTTPS, com possibilidade de usar a abordagem *Representational State Transfer* (REST); (ii) *Message Queue Telemetry Transport* (MQTT) 3.1/3.1.1, e (iii) *Constrained Application Protocol* (CoAP);

- **Camada de Agregação/Barramento**: possui funções de implementar um servidor *Hypertext Transfer Protocol* (HTTP), ou um *broker* MQTT, para permitir a comunicação com dispositivos, bem como agregar e combinar informações de diferentes dispositivos e encaminhá-los a um determinado equipamento, como também realizar, se necessário, o papel de gateway convertendo mensagens MQTT para requisições HTTP, e vice-versa, servindo como um meio de ligação entre dispositivos que executam diferentes protocolos. Além disso, cabe a camada de agregação atuar como um servidor OAuth2, realizando requisições para a camada de identidade e gerenciamento de controle de acesso a fim de permitir ou negar o acesso aos dispositivos e

seus recursos;

- **Processamento de Eventos e Análise**: provê capacidades para processar e agir sobre os eventos obtidos da camada de agregação/barramento. Um dos pontos chave dessa camada é o armazenamento de informações em um banco de dados que pode ser do tipo relacional, com acesso cliente-servidor, ou orientado a tecnologias *Bigdata*, como o *Hadoop* ou, ainda, de sistemas de análise em tempo real como o *Storm*;

- **Comunicações Externas**: permite que o sistema seja acessado por equipamentos e serviços externos a solução, por meio de três abordagens principais. A primeira é por painéis de visualização e monitoramento (*dashboards*), que disponibilizam gráficos e técnicas de zoom, além de diferentes visões para análise de eventos e tomada de decisões. A segunda emprega interfaces de programação para aplicações (*Application Programming Interface* (API)), permitindo que se faça uma comunicação direta através da rede. A última abordagem se utiliza de páginas ou portais Web para interagir com dispositivos e com a camada de processamento de eventos;

- **Gerenciador de Dispositivo**: objetiva prover mecanismos para permitir a configuração e gerência remota desses dispositivos, como a instalação de imagens de sistemas e parametrização; obtenção de informações do funcionamento, disponibilidade e localização do dispositivo, entre outros. Esses mecanismos são implantados com o emprego de dois componentes, um servidor e um agente que executa no próprio dispositivo;

- **Gerenciamento de Identidade e Acesso**: fornece mecanismos de autenticação e autorização para que os recursos sejam acessados apenas por quem tiver esse direito. Para isso, a proposta WSO2 recomenda o emprego de sistemas de diretório, o uso de sistemas como *OpenID connect*, SAML2, entre outros.

3.2 Projeto Xively

O projeto Xively (XIVELY, 2016) disponibiliza um middleware comercial e de código fechado que se utiliza de uma abordagem baseada em *Cloud Computing* para tratar e armazenar os dados providos pelos dispositivos. O Xively se basea no padrão arquitetural REST, disponibilizando os sensores e atuadores como recursos Web e disponibilizando interfaces padroni-

zadas para a troca de dados, através de uma estratégia de Sensoriamento como Serviço (ZASLAVSKY; PERERA; GEORGAKOPOULOS, 2013).

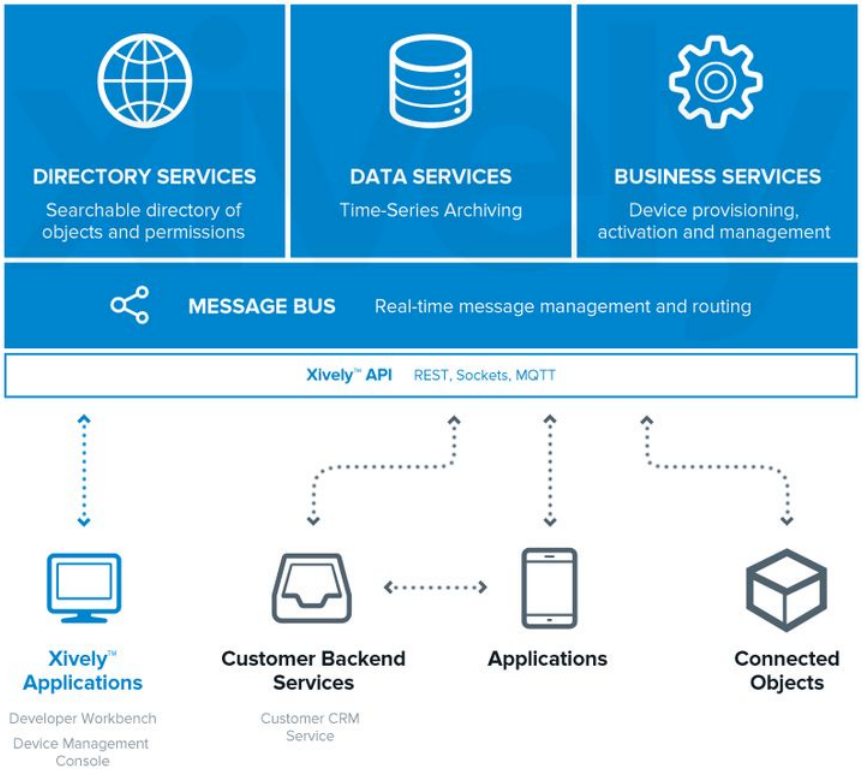


Figura 3.2 – Arquitetura do Middleware Xively
Fonte: (XIVELY, 2016)

Pelas suas características comerciais alguns detalhes sobre a sua arquitetura não são discutidos a fundo na literatura, sendo disponíveis apenas no site oficial da plataforma. A arquitetura simplificada do Xively é apresentada na Figura 3.2 e os componentes caracterizados a seguir:

- **Directory Services**: Este modulo é responsável por manter um diretório dos objetos que compões o ambiente e as permissões de acesso desses;
- **Data Services**: Este módulo é responsável por realizar o armazenado dos dados recebidos e também fornecer os dados históricos requisitados pela API;
- **Business Services**: Este módulo é responsável pela ativação e gerenciamento dos dispositivos. Nesta camada também é realizado o gerenciamento das *triggers* que devem atuar sobre os dados recebidos de maneira reativa;

- **Message Bus**: Este modulo é responsável por realizar a comunicação entre os módulos da plataforma, realizar o gerenciamento em tempo real dos eventos recebidos através de *triggers* e de realizar o roteamento das mensagens para os subscritores. As mensagens são enviadas através da API para uma URL pré-determinada pelos subscritores;

- **Xively API**: A API do Xively é utilizada para o recebimento de dados dos dispositivos, assim como para acesso dos serviços providos pela plataforma. Todo acesso é realizado através da API REST, trabalhando com Sockets, MQTT e protocolo HTTP. Os dados enviados através da solicitação POST são estruturados em formato *JavaScript Object Notation* (JSON) ou *Extensible Markup Language* (XML).

A plataforma suporta diversos tipos de formatos de dados, porém os dados não são homogeneizados automaticamente, devendo assim ser realizado um pré-processamento dos dados a fim de combiná-los. Os dados são organizados através de *feeds*, *datapoints* e *datastreams*. Um *feed* representa os dados de um ambiente (uma sala, por exemplo) com seus respectivos *datastreams*, que representam dados enviados por um determinado sensor desse ambiente (por exemplo, temperaturas do ambiente monitorado), enquanto um *datapoint* representa um único valor de um *datastream* em um determinado instante de tempo. Xively não possui nenhum mecanismo de descoberta, sendo necessário adicionar os dispositivos manualmente pela interface web.

3.3 Projeto LinkSmart

O projeto LinkSmart (anteriormente chamado Hydra) é uma plataforma de middleware baseada em *Service-Oriented Architecture* (SOA) e desenvolvida na linguagem Java, que oferece suporte ao desenvolvimento de aplicações com base em informações fornecidas por dispositivos físicos heterogêneos, disponibilizando interfaces de serviços Web para controle destes dispositivos (LINKSMART, 2015).

Os dispositivos físicos integrados a plataforma são classificados como: dispositivos nativos (com capacidade computacional para hospedar a plataforma) ou dispositivos não nativos (dispositivos restritos, sem capacidade para hospedar a plataforma), no entanto podem ser integrados a ela através de gateways. Os gateways contêm uma *proxy* para cada dispositivo ligado a eles, e por meio desta *proxy* fica capacitado a trocar dados com o dispositivo, colocando-o na rede LinkSmart.

Sua arquitetura possui três camadas principais: (i) camada de rede, responsável pela comunicação com os dispositivos; (ii) camada de serviço, responsável pelo gerenciamento de eventos, dispositivos, escalonamento de recursos, dentre outros, e; (iii) camada semântica.

A arquitetura LinkSmart é apresentada na Figura 3.3 e seus componentes caracterizados a seguir.

- **Gateway**: é representado por dispositivos computacionais com conexão à Internet contendo *proxies* que traduzem o tipo de tecnologia e formatos de dados empregados pelos dispositivos físicos ligados a ele, possibilitando a integração de dispositivos não nativos a rede LinkSmart. Possui suporte a conexão com estes dispositivos através de USB, *ZigBee e Bluetooth*;

- ***Network Manager***: provê o registro de clientes ou serviços, bem como a busca de serviços dentro do LinkSmart. Possibilita a comunicação entre dois nós da rede LinkSmart, para o envio de mensagens e abstração dos métodos de comunicação e rede;

- ***Event Manager***: gerencia a troca de informações utilizando o modelo de comunicação Publicador-Subscritor, sendo a subscrição realizada a partir de tópicos de interesse;

- ***Crypto and Trust***: realiza operações criptográficas, a avaliação de confiança e a execução de políticas de segurança de controle de acesso;

- ***Device Application Catalogue* (DAC)**: responsável por manter o controle de dispositivos disponíveis na rede LinkSmart e acesso destes pelas aplicações.

A LinkSmart também contempla uma descrição semântica dos dispositivos através do uso de ontologias de dispositivos, capazes de representar todas as meta-informações sobre os dispositivos. A ontologia utilizada é baseada na ontologia de dispositivos *Foundation for Intelligent Physical Agents* (FIPA) que permite a parametrização semântica para incluir informações dos dispositivos, como seus recursos de segurança.

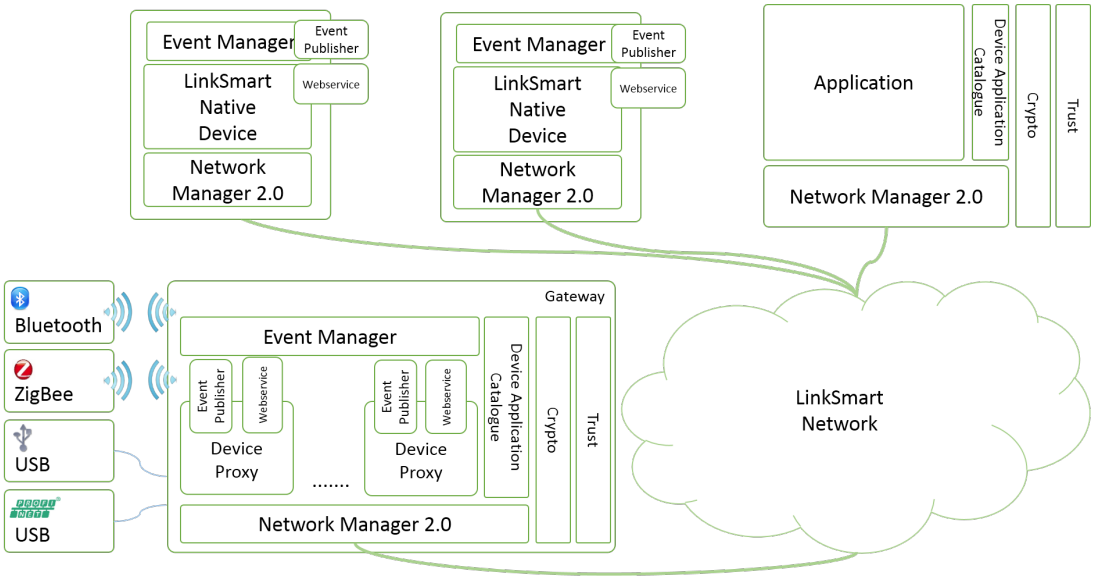


Figura 3.3 – Arquitetura do Middleware LinkSmart
Fonte: (LINKSMART, 2015)

3.4 Projeto Carriots

O Projeto Carriots propõe uma plataforma para aplicações em IoT que utiliza serviços de *Cloud Computing* para gerenciar dados providos por qualquer tipo de dispositivo, além de conectar dispositivos a dispositivos e a outros sistemas, o que a faz se alinhar ao conceito de *Platform as a Service* (PaaS) (CARRIOTS, 2015). Portanto, se um sistema for conectado à plataforma, ele também pode ser modelado como um dispositivo. A partir de sua API *RESTful*, a Carriots tem por objetivo coletar e armazenar qualquer dado originado dos mais diversos dispositivos, utilizá-lo em seu motor de aplicações e disponibilizá-lo a seus usuários não importando o volume de dispositivos conectados.

A plataforma Carriots foi projetada tendo como base dois blocos principais: (i) organização lógica de entidades e (ii) arquitetura de componentes de software.

No bloco de organização lógica de entidades, as entidades são definidas de forma hierárquica. Dessa forma, os dispositivos são organizados primeiramente em grupos, de acordo com a sua localização geográfica. Por sua vez estes grupos são associados a um serviço que pertence a um projeto específico desenvolvido pelo usuário. Por conta dessa hierarquia, a primeira atividade de um usuário ao interagir com a plataforma é a criação de um projeto e por consequência caso um projeto seja desabilitado, isso impactará na desativação de todas as entidades (serviços, grupos e dispositivos) vinculados a ele.

Outro componente que pode ser inserido nesta hierarquia de entidades é o *Listener*, responsável por realizar o monitoramento de eventos, como o recebimento ou persistência de

dados, e caso um evento monitorado atenda alguma condição pré-configurada executar a análise e processamento dos dados relacionados a este evento.

A arquitetura de software do Carriots (vide Figura 3.4) é formada pelos seguintes módulos:

- **API REST**: seguindo padrões da Internet esta API REST é implementada sobre HTTPS, sendo responsável pela forma como os dispositivos e outros sistemas interagem com a plataforma. A API REST utiliza os formatos JSON ou XML (em um formato particular da plataforma) para representação dos dados. Possui plena capacidade de interação entre a plataforma e os painéis de controle personalizados, *dashboards*, etc;

- ***Big Data***: provê as aplicações flexibilidade para gerenciar os dados dos mais diversos dispositivos, de modo que a massa de dados seja armazenada em uma arquitetura de *Big Data* em formato *schemaless*, permitindo assim armazenar dados sem que seja necessário normalizá-los;

- **Gerenciamento de Projetos e Dispositivos**: possui como funcionalidade armazenar e gerenciar os projetos criados pelos usuários (dos mais simples aos mais complexos), podendo estes serem organizados para atenderem a quaisquer requisitos. Este módulo é também responsável por realizar a configuração dos dispositivos, como por exemplo o ajuste na taxa de amostragem de um sensor de estacionamento e atualizar os seus softwares embarcados, possuindo a capacidade de realizar o gerenciamento remoto dos dispositivos;

- **Processamento de Eventos e Regras de Negócio**: estes módulos são responsáveis por armazenar e executar eventos, segundo uma lógica de negócios, em forma de *scripts* criados utilizando a linguagem de programação *Groovy* e fazendo uso de regras do tipo *if-then-else*;

- **Segurança**: neste módulo questões de segurança podem ser tratadas de quatro formas: (i) através do uso de chaves (*APIkeys*) pré-compartilhadas para a definição de privilégios de acesso; (ii) através da utilização do protocolo HTTPS para a criptografia das requisições e respostas à API REST; (iii) pela utilização de Hash HMAC (*Keyed-Hash Message Authentication Code*) e senha para autenticação e verificação de conteúdo, e; (iv) por criptografia customizada a partir de *scripts* criados pelo usuário, permitindo a adição de soluções de segurança adicionais;

- ***Logs e Debug***: fornece um console de *debug* que facilita o desenvolvimento dos projetos hospedados na plataforma, através de ferramentas para depuração de erros e registro de mensagens, tornando tais dados (mensagens de *log*) acessíveis a partir do painel de controle;

- **Painel de Controle**: permite o gerenciamento pelo usuário de todos os outros módulos e recursos da plataforma a partir de uma interface Web;

- **Módulo de Comunicação Externa**: complementa os recursos de interação da plataforma ao permitir o envio de *e-mails*, *Short Message Service* (SMS) e a interação com outros sistemas (como a plataforma de compartilhamento de arquivos Dropbox ou a rede social Twitter).

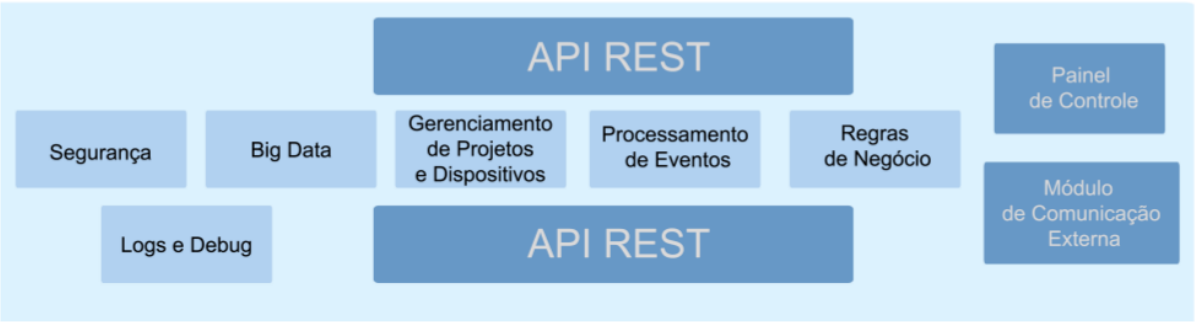


Figura 3.4 – Arquitetura do Middleware Carriots
Fonte: (PIRES et al., 2015b)

3.5 Discussão dos Trabalhos Relacionados

Nesta seção são discutidas as características e funcionalidades dos trabalhos relacionados considerando as principais premissas para concepção do EXEHDA-FOG, listadas a seguir:

- (a) suporte a distribuição na coleta e tratamento de eventos;

- (b) suporte a composição de eventos;

- (c) escalabilidade sob o prisma da IoT;

- (d) suporte a gerência dinâmica de eventos.

Na Tabela 3.1 é apresentada a comparação dos trabalhos, onde o símbolo “+” indica que o requisito é completamente atendido, o símbolo “-” denota que o requisito não é atendido e o símbolo “+/-” indica que o requisito é parcialmente atendido. A seguir são discutidas a classificação dos trabalhos em cada quesito.

Tabela 3.1 – Comparação dos trabalhos relacionados.

	WSO2	Linksmart	Xively	Carriots
a	+/-	+	+/-	+/-
b	+/-	+/-	-	-
c	+	+/-	+	+
d	+/-	+/-	+/-	+/-

Conforme os dados apresentados na Tabela 3.1, percebe-se que nenhum dos Middlewares apresentados atende completamente os requisitos de uma abordagem de Fog Computing para o processamento contextual. O Middleware que apresenta maior número de requisitos atendidos, ou parcialmente atendidos, é o WSO2, porém por se tratar de uma arquitetura baseada em hierarquia, apenas um nodo da rede é responsável pelo tratamento dos eventos, não havendo possibilidade de tomada de decisões locais diretamente onde os dados são coletados, na borda computacional. Outro ponto negativo encontrado em todo os Middlewares estudados é a necessidade de se manter uma conexão ativa com a *Cloud* de forma a se realizar as análises sobre os dados coletados e atuar sobre o ambiente. Os Middlewares dos trabalhos relacionados permitem alterar as regras de processamento contextual, mesmo que em um nodo específico, porém isso implica na necessidade de interrupção de sua execução ou que um usuário desenvolva novos códigos de processamento contextual, implicando diretamente na gerência dinâmica dos ambientes monitorados por eles.

Dentre os aspectos interessantes sistematizados nos trabalhos relacionados está o da utilização de modelos publicador/subscritor de forma a se distribuir os eventos detectados a todos dispositivos interessados. Além disso, os trabalhos se utilizam de APIs do tipo REST de forma a desacoplar os serviços providos, bem como o uso de padrões e protocolos abertos utilizados na web como o HTTP, JSON e XML.

3.6 Considerações Sobre o Capítulo

Neste capítulo foi realizada uma sistematização dos principais middlewares direcionados a tratar as demandas relacionados a IoT encontrados na literatura, visando identificar suas características e também avaliar as suas arquiteturas sob o prisma da *Fog Computing*.

A sistematização mostrou-se oportuna, permitindo identificar como a proposta de *Fog Computing* empregada no EXEHDA-FOG, pode contribuir para a solução de alguns dos desafios da IoT, principalmente no quesito de tratamento e coleta de dados contextuais de forma distribuída e na sua gerência dinâmica, assim como o suporte a composição de eventos e na colaboração entre nodos arquiteturais visando obter uma vantagem na tomada de decisão da arquitetura.

4 EXEHDA-FOG: ARQUITETURA E FUNCIONALIDADES

Este capítulo apresenta os aspectos de concepção da abordagem proposta denominada EXEHDA-FOG, a qual constitui a contribuição central desta dissertação de mestrado. Neste sentido são discutidas as premissas de concepção do EXEHDA-FOG, sua arquitetura e funcionalidades desenvolvidas.

4.1 Premissas de Concepção

As premissas de concepção do EXEHDA-FOG tem por finalidade qualificar o middleware EXEHDA, de forma a permitir o mesmo atender às demandas inerentes às modernas infraestruturas computacionais providas pela Internet das Coisas. A perspectiva é promover neste cenário da IoT suporte ao processamento distribuído dos dados contextuais obtidos do meio, através dos próprios equipamentos que gerenciam a coleta e atuação, caracterizando uma abordagem de *Fog Computing*.

Como principais aspectos a serem providos neste cenário destacaríamos: (i) suporte a perda de conexão, (ii) distribuição vertical do processamento contextual entre nodos EXEHDA de diferentes naturezas, (iii) distribuição horizontal do processamento contextual entre os nodos EXEHDA que tratam da captura das informações dos sensores e (iv) escalabilidade.

O EXEHDA (LOPES et al., 2014) é um middleware baseado em serviços que visa criar e gerenciar um ambiente para UbiComp, bem como promover a execução de aplicações sobre esse ambiente. A Internet das Coisas é vista nesta dissertação como a abordagem que vem contribuindo para a materialização dos pressupostos da UbiComp.

A arquitetura de software do EXEHDA, apresentada na Figura 4.1, é dividida em três camadas: camada inferior, composta pela infraestrutura para execução das outras camadas (máquina virtual, sistema operacional e hardware); camada intermediária, constituída pelo Middleware EXEHDA; e a camada superior, formada pelo *framework* para programação das aplicações ubíquas. O módulo que representa a Consciência ao Contexto tem destaque na arquitetura, uma vez que permeia os demais componentes. Esse módulo tem por finalidade gerenciar os contextos de interesse das aplicações e do próprio ambiente de execução.

Dos subsistemas que constituem o EXEHDA, mais especificamente esta dissertação contribui com o aquele responsável pela Adaptação e Reconhecimento do Contexto (vide Figura 4.1).

O EXEHDA-FOG herda as características do mecanismo de ciência de contexto pro-

posto para o EXEHDA, o qual emprega uma estratégia colaborativa entre aplicação e ambiente de execução, através da qual é facultado ao programador individualizar políticas para reger o comportamento de cada um dos componentes que constituem o software da aplicação. No EXEHDA-FOG, os eventos de contexto são pró-ativamente monitorados e o suporte à execução deve permitir que tanto as aplicações como ele próprio utilizem essas informações na gerência de seus aspectos funcionais e não-funcionais.

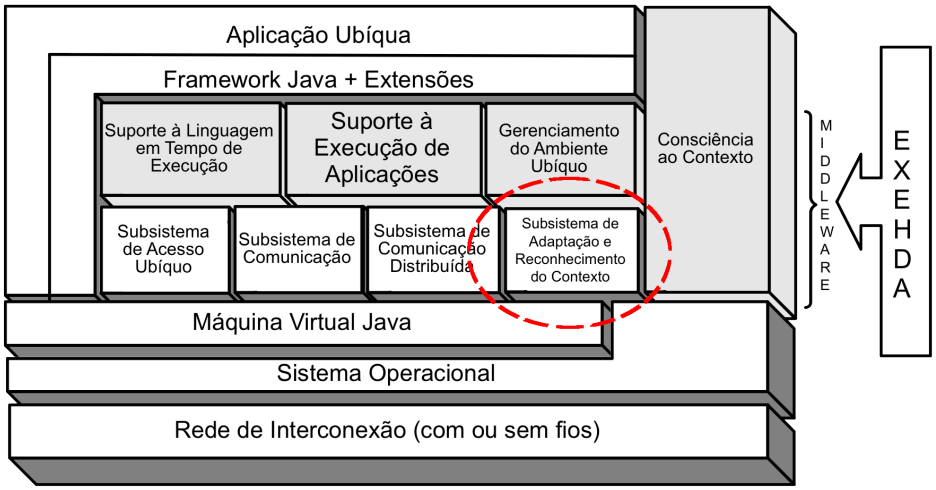


Figura 4.1 – Subsistemas do Middleware EXEHDA
Fonte: (LOPES et al., 2014)

Os esforços de concepção dos módulos que constituem o EXEHDA-FOG consideraram sua integração na arquitetura de software do middleware EXEHDA, bem como os seus princípios operacionais.

O fato do EXEHDA ter uma origem enquanto middleware para UbiComp, faz com que o tratamento da Ciência de Contexto seja inerente à arquitetura. O desafio central desta dissertação está em capacitar o middleware para que este continue provendo suporte a Ciência de Contexto, considerando o processamento das informações contextuais o mais próximo possível do local da coleta, interoperando os equipamentos responsáveis por tratar os sensores, equipamentos de borda, de forma mais autônoma possível, caracterizando assim a perspectiva operacional considerada em Fog Computing.

Considerando isso, o EXEHDA-FOG foi modelado empregando os seguintes pressupostos:

- Estratégia descentralizada: os dispositivos de borda do middleware deverão possuir capacidade de tomar decisões no ambiente que estão inseridos, de forma colaborativa, com intuito de reduzir a latência na tomada de decisões com base nas variações contextuais e aumentando a confiabilidade operacional do middleware;
- Processamento distribuído de eventos: através de políticas de publicação e subscrição os dispositivos de borda podem se comunicar e processar os eventos no momento em que eles são gerados, auxiliando na distribuição necessária ao *Fog Computing*;
- Serviço de diretório: tem por objetivo prover suporte as ações de subscrição e publicação dos equipamentos de borda. O diretório com as informações dos recursos de sensoriamento e atuação será atualizado na arquitetura a partir dos próprios equipamentos de borda a medida que novos sensores e atuadores são instanciados.
- Operação de borda desconectada: os equipamentos de borda devem manter os procedimentos de coleta de dados contextuais, bem como o processamento de regras de contingência locais, gerando os respectivos procedimentos de atuação emergenciais.
- Eventos da arquitetura tratados através de API: os diferentes eventos inerentes a operação do EXEHDA-FOG como: coleta de informações sensoriadas, subscrições, publicação, etc., deverão ser tratados por uma interface que permita uma manipulação em mais alto-nível dos mesmos.

Para atender estes pressupostos o EXEHDA-FOG introduz novos módulos arquiteturais no Subsistema de Adaptação e Reconhecimento do Contexto (SRAC) do Servidor de Contexto e nos Servidores de Borda que serão discutidos na próxima seção.

O EXEHDA-FOG capacita a arquitetura do middleware EXEHDA a combinar informações distribuídas de eventos, realizar associações e inferências semânticas sobre os mesmos, promovendo operações de filtragem e agregação.

A expectativa é obter uma nova organização arquitetural que expanda a atual visão do EXEHDA que é SOA, para uma arquitetura *Event-Driven Service Oriented Architecture* (ED-SOA) com inteligência descentralizada (LAN et al., 2015).

4.2 Modelo Arquitetural

O entendimento da atuação da arquitetura do EXEHDA-FOG passa pela análise de como é constituído o ambiente Ubíquo gerenciado pelo middleware EXEHDA. Uma visão dos prin-

cipais componentes deste ambiente é apresentada na Figura 4.2.

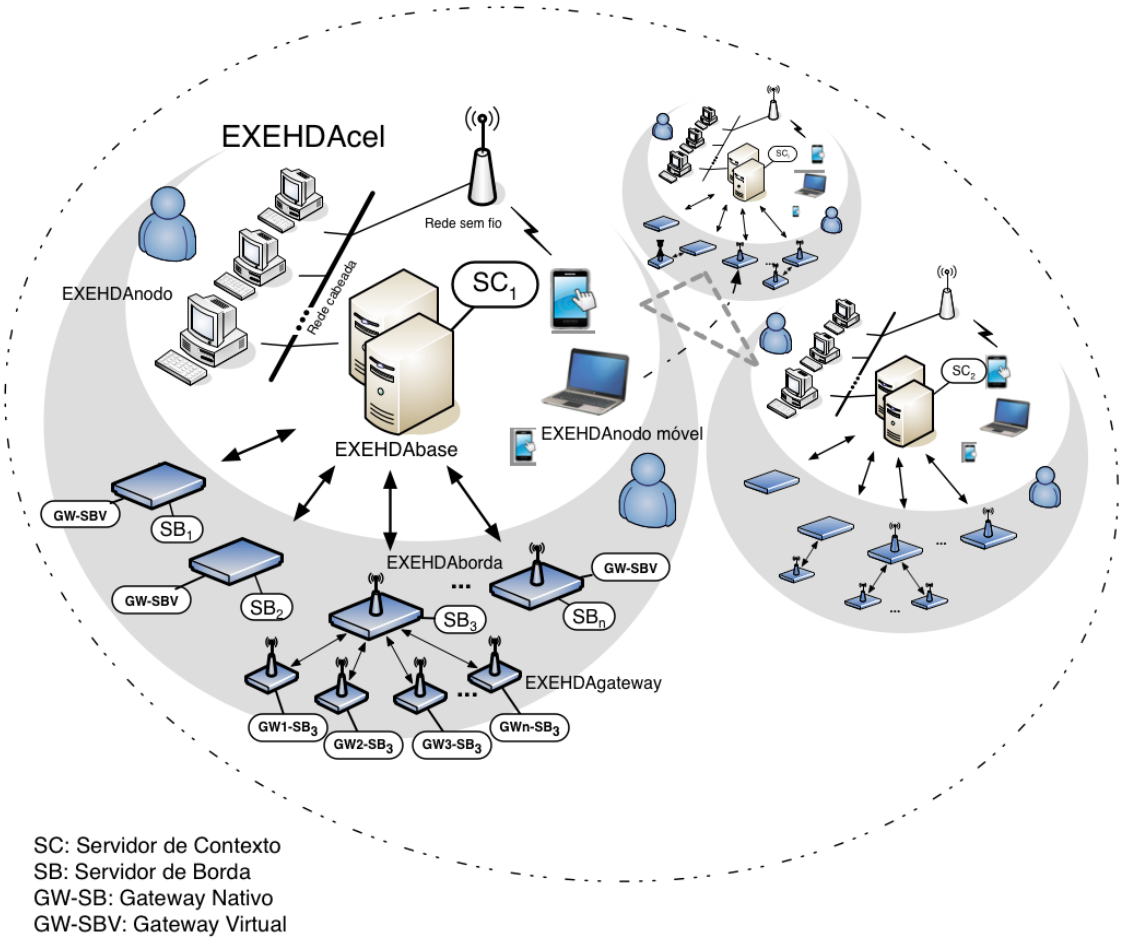


Figura 4.2 – Ambiente Ubíquo gerenciado pelo EXEHDA
Fonte: (LOPES et al., 2014)

O Servidor de Borda (SC) permanece em execução no dispositivo central da Célula de Execução do middleware EXEHDA (EXEHDAbase), sendo o SC responsável pela coordenação dos serviços referentes a ciência de contexto.

O Servidor de Contexto é disponibilizado na perspectiva de uma *Cloud* interna em cada uma das Células de execução do EXEHDA, utilizando assim uma abordagem chamada *cloud-centric* (MLADENOW; KRYVINSKA; STRAUSS, 2012), na qual o processamento dos eventos relacionados à aquisição das informaçõees contextuais, são realizados em um dos equipamento desta *Cloud*. Na arquitetura atual do ambiente ubíquo o Servidor de Contexto (SB) tem como objetivo principal repassar ao Servidor de Contexto as informações contextuais coletadas através dos Gateways. Os Gateways são dispositivos de baixo poder computacional dotados de capacidade de tratar os protocolos exigidos pelos diferentes sensores e atuadores, gerenciando a interação com o meio físico.

Por sua vez, no EXEHDA-FOG a premissa é que os Servidores de Borda possam se comunicar entre si, sendo dotados de recursos para coletar e processar eventos de contexto produzidos em outros Servidores de Borda da mesma Célula de Execução.

Esta estratégia computacional que permite que aos equipamentos de borda terem um perfil de operação distribuído e cooperativo, constitui a base para a abordagem de *Fog Computing* (PRIVAT et al., 2016).

A abordagem prevista para o EXEHDA-FOG, dentre outros aspectos, irá contribuir para diminuir a necessidade de comunicação entre os SBs e o SC, uma vez que os dados que serão enviados para o Servidor de Contexto já terão sido consolidados por uma estratégia de fusão e/ou filtragem considerando valores de diferentes sensores, os quais poderão estar em um mesmo Servidor de Borda, ou em Servidores de Borda Distintos.

Na Figura 4.3 é apresentada a visão mais alto nível da nova arquitetura do middleware EXEHDA para o Servidor de Contexto e Servidor de Borda, a qual é baseada nos trabalhos (DAVET et al., 2016) e (CARDOZO et al., 2016a).

De forma a viabilizar uma abordagem direcionada ao processamento de eventos de contexto distribuídos foram introduzidos dois novos módulos arquiteturais nos Servidores de Contexto e de Borda, o SC-FOG e o SB-FOG, respectivamente. Outros módulos arquiteturais já presentes na arquitetura também tiveram suas características alteradas visando oferecer suporte a proposta do EXEHDA-FOG, estes módulos estão em destaque na Figura 4.3, os módulos legados não sofreram alterações significativas.

4.2.1 Servidor de Contexto

O Servidor de Contexto no EXEHDA-FOG tem suas funcionalidades organizadas através dos seguintes módulos arquiteturais.

Módulo Repositório de Contextos e Configurações

O Repositório de Contextos foi reformulado para que as informações referentes as configurações da arquitetura fiquem em um Repositório específico para esta finalidade. Assim teremos dois Repositórios: um de Contextos e outro de Configurações.

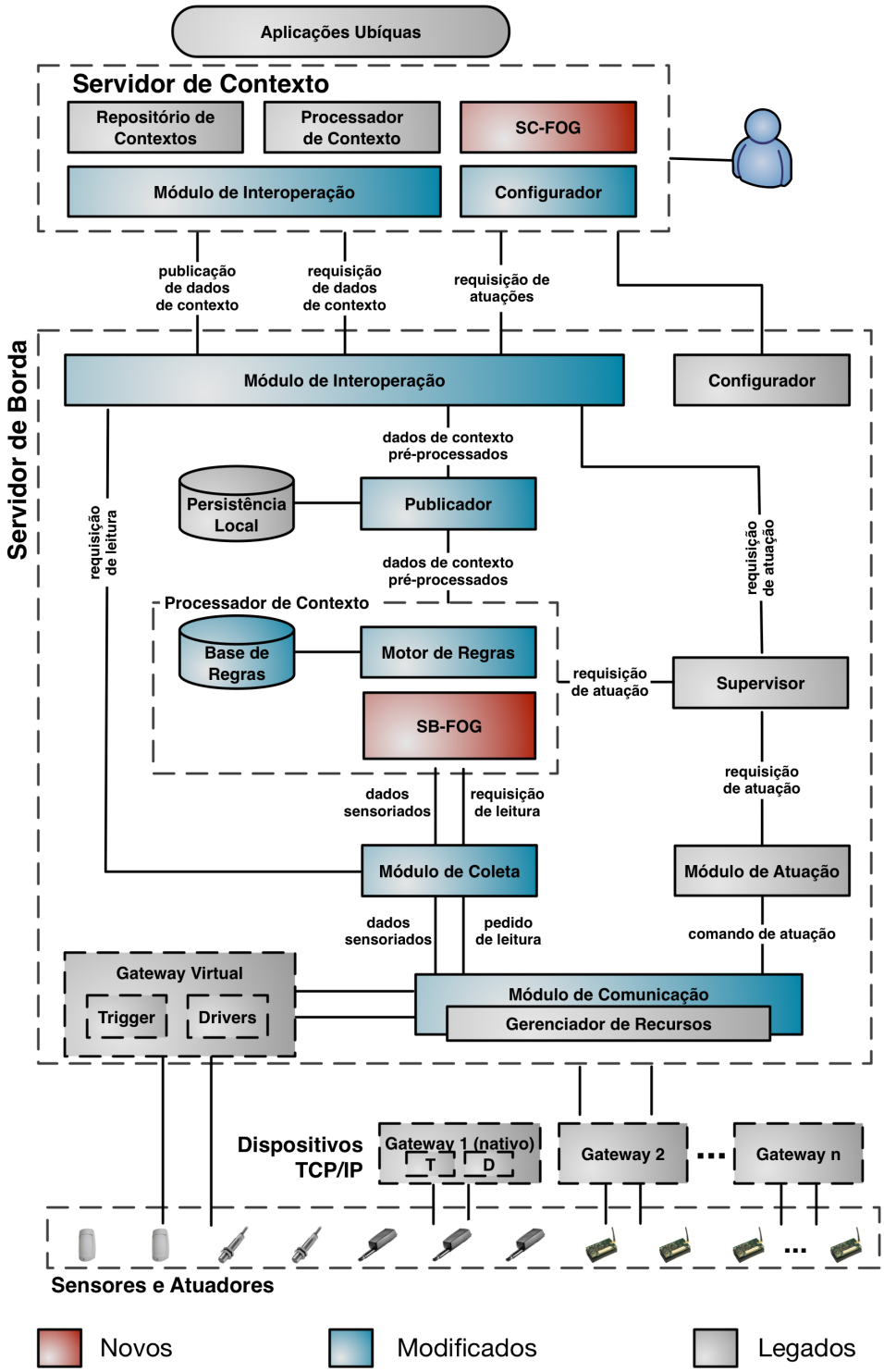


Figura 4.3 – EXEHDA-FOG: Visão Geral
Fonte: O Autor

No Repositório de Contextos são armazenados os valores sensoreados publicados pelos Servidores de Borda existentes na célula. Os dados de contexto armazenados nesse Repositório são usados pelo módulo Processador de Contexto. Este armazenamento ganha importância sempre que for necessária a utilização de valores históricos de contexto nas lógicas de processamento contextual.

Assim, no EXEHDA-FOG o processamento sobre os dados contextuais realizado no Servidor de Contexto é voltado para dados coletados durante um intervalo de tempo maior (dados históricos), enquanto que nas bordas este processamento é realizado em janelas de tempo menores. Ressalte-se que no Servidor de Contexto estarão disponíveis os dados contextuais provenientes de todos os Servidores de Borda da Célula. Também é facultado a um Servidor de Contexto acessar dados pertinentes a outra Célula de Execução, através da interoperação com o Servidor de Contexto da mesma.

Módulo Processador de Contexto

O Processador de Contexto realiza tarefas de manipulação das informações contextuais. Baseia-se em regras do tipo ECA, de forma a tratar eventos gerados a partir de mudanças nos estados dos contextos de interesse, que podem acarretar no disparo de ações pertinentes em função do estado contextual. A recepção de uma publicação de dados contextuais pelo Servidor de Borda constituem o gatilho para o disparo de regras. O processador de contexto recebe os eventos detectados pelo servidor de borda, que sejam do seu interesse, via o módulo SC-FOG.

Módulo Configurador

O módulo Configurador concentra os procedimentos de configuração necessários para o funcionamento dos demais módulos. Através deste módulo é disponibilizada uma aplicação web através da qual é possível gerenciar dispositivos que atuam como Servidores de Borda, como Gateways, as regras para processamento contextual, bem como realizar todo tipo de cadastro necessário ao funcionamento de uma aplicação IoT ciente de contexto, uma dessas interfaces pode ser visualizada na Figura 4.4. Outras funcionalidades providas por essa aplicação web estão disponíveis no Anexo B.

O módulo configurador também é responsável pela comunicação com o módulo SC-FOG, com o intuito de configurar as regras para processamento dos eventos de contexto distribuídos configuradas através da interface web do Servidor de Contexto.

A comunicação com os demais módulos é realizada através de uma APIRestful desenvolvida para esta interoperacão e uma padronização de mensagens entre Servidor de Contexto e Servidor de Borda utilizando-se de requisições POST, PUT, DELETE.

As informações pertinentes a configuração dos diferentes recursos de hardware empre-

gados no Subsistema de Reconhecimento de Contexto e Adaptação, são gravadas pelo Configurador no Repositório de Configurações.

Figura 4.4 – EXEHDA-FOG: Aplicação Web Configuração
Fonte: O Autor

Módulo de Interoperação

O Módulo de Interoperação foi concebido de forma a gerenciar a comunicação com Servidores de Borda e demais serviços do middleware EXEHDA, oferecendo uma interface padronizada e bem definida. Este modulo implementa o estilo arquitetura REST sob o protocolo HTTP.

Este Módulo provê o suporte necessário para disponibilização de uma API REST para o acesso aos recursos oferecidos pelo Servidor de Contexto. Com isso, a interoperação ocorre por meio *Uniform Resource Identifier* (URI), viabilizando o acesso a todos os recursos disponibilizados pela arquitetura.

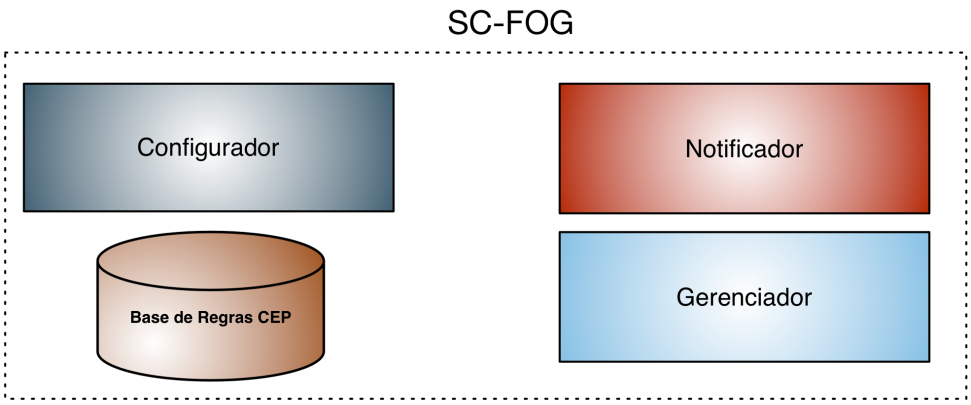


Figura 4.5 – SC-FOG: Módulos do Servidor de Contexto
Fonte: O Autor

SC-FOG

O módulo SC-FOG é responsável pela criação, edição e distribuição aos Servidores de Borda, das regras que irão tratar os eventos de contexto nos Servidores de Borda.

Os módulos do Servidor de Contexto do EXEHDA-FOG, descritos na Figura 4.5, são apresentados a seguir, assim como as suas funcionalidades específicas.

Módulo Configurador

O Configurador do SC-FOG é o módulo da arquitetura responsável por receber regras e dados de configuração provenientes do Configurador do Servidor de Contexto.

As regras são codificadas em JSON, e a medida que são recebidas são transpostas para a Base de Regras do SC-FOG, um exemplo dessa regra é apresentado na Figura 4.12 na Secção 4.3.

Módulo Base de Regras CEP

A Base de Regras CEP armazena as regras para processamento dos eventos de contexto distribuído. Esta tabela mantém a associação das regras com os Servidores de Borda que as empregam. A definição das Regras e quais Servidores de Borda estão envolvidos é obtida através do Módulo Configurador.

As informações da Base de Regras são disponibilizada ao administrador do middleware sempre que novas definições para tratamento de contexto nos Servidores de Bordas tiverem de ser feitas.

Esta base também se mostra importante, para os procedimentos de reconstrução de um Servidor de Borda, que venha sair de uso por defeito.

Módulo Notificador

O Notificador tem a função de gerar as notificações de acordo com as detecções de situações no EXEHDA-FOG. Essas notificações são geradas tanto pelo módulo Processador do SC, como pelo processamento contextual realizado de forma distribuída pelos Servidores de Borda.

Este módulo trabalha com o modelo publicador/subscritor, recebendo subscrições de todos os serviços e/ou aplicações que desejarem ser informados das mudanças nos estados contextuais e/ou seus eventos de interesse. De acordo com as regras previamente especificadas, o Notificador pode enviar comandos de atuação aos Servidores de Borda.

Módulo Gerenciador

O Gerenciador do SC-FOG é o módulo responsável por gerenciar os Servidores de Borda. Sua principal função é caracterizar o status de um Servidor de Borda junto a Célula de Execução, assim como manter um registro de todas operações e serviços que podem ser utilizados para processamento contextual nos Servidores de Borda.

O status de um SB é decorrência da operação do mesmo, ou de uma ação deliberada do administrador do middleware através do módulo de Configuração.

4.2.2 Servidor de Borda

Os Servidores de Borda têm suas funcionalidades sistematizadas através dos seguintes módulos arquiteturais.

SB-FOG

O módulo SB-FOG foi concebido de forma a implementar as premissas operacionais de *Fog Computing* no processamento distribuído de eventos de contexto na arquitetura. Neste módulo é realizada a gerência dos eventos produzidos por um Servidor de Borda em particular, bem como a interoperação deste com outros SBs existentes na mesma Célula de Execução.

Os módulos do Servidor de Borda do EXEHDA-FOG estão apresentados na Figura 4.6, e sua dinâmica de interoperação quando da aquisição e processamento de contexto está caracterizada na Figura 4.7. Uma descrição destes módulos e suas funcionalidades está feita a seguir.

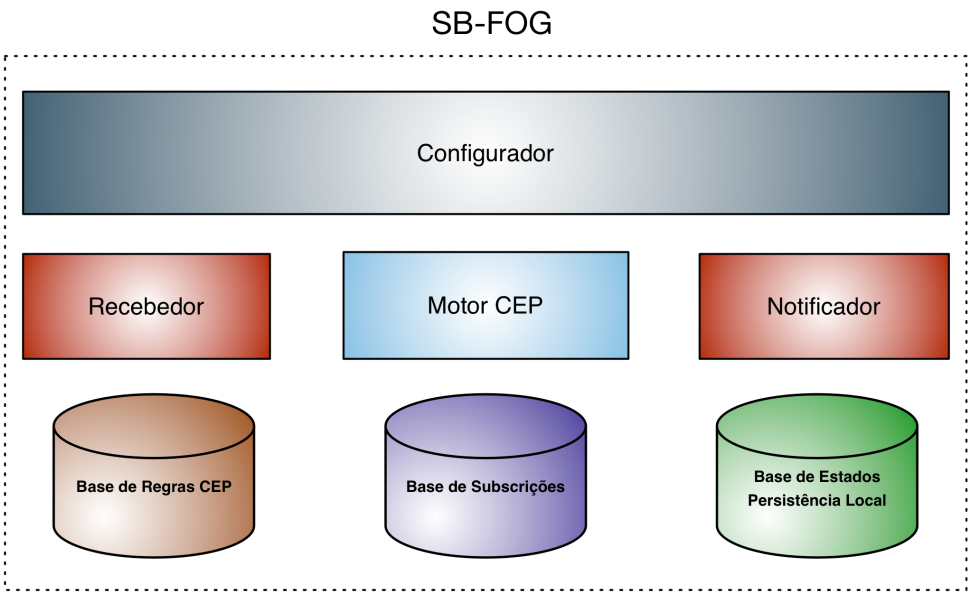


Figura 4.6 – SB-FOG: Módulos do Servidor de Borda
Fonte: O Autor

Módulo Configurador

Analogamente ao Configurador presente no SC-FOG, este é o módulo arquitetural que recebe os dados de configuração a partir do SC-FOG e transpõem as regras para o Servidor de Borda, realizando o seu armazenamento local.

O Módulo Configurador também gerencia as requisições de subscrição aos eventos de contexto gerados pelo Servidor de Borda. Uma visão do procedimento para gerência das regras no EXEHDA-FOG pode ser vista na Figura 4.8.

Módulo Recebedor

O módulo Recebedor do Servidor de Borda tem como principal função identificar se existe interesse por parte de alguma Regra de Processamento Contextual Distribuído, de valores de contexto recebidos pelo próprio Servidor de Borda, como visto na Figura 4.7 o módulo recebe mensagens do módulo de interoperação e do módulo de comunicação e, em caso de um evento ou dado contextual, este é formatado de forma que o Motor CEP possa interpretá-lo.

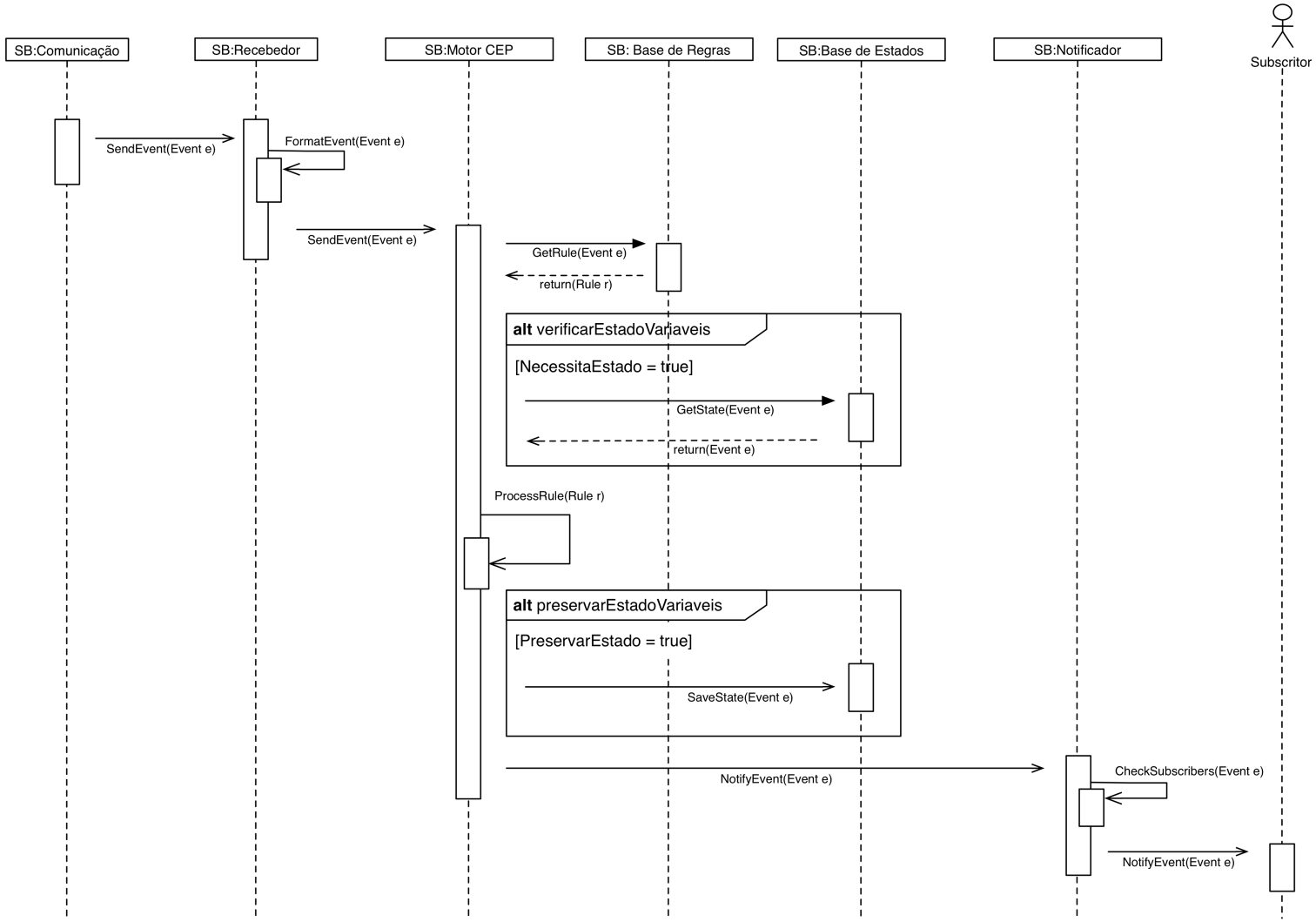


Figura 4.7 – Fluxo de Processamento Contextual
Fonte: O Autor

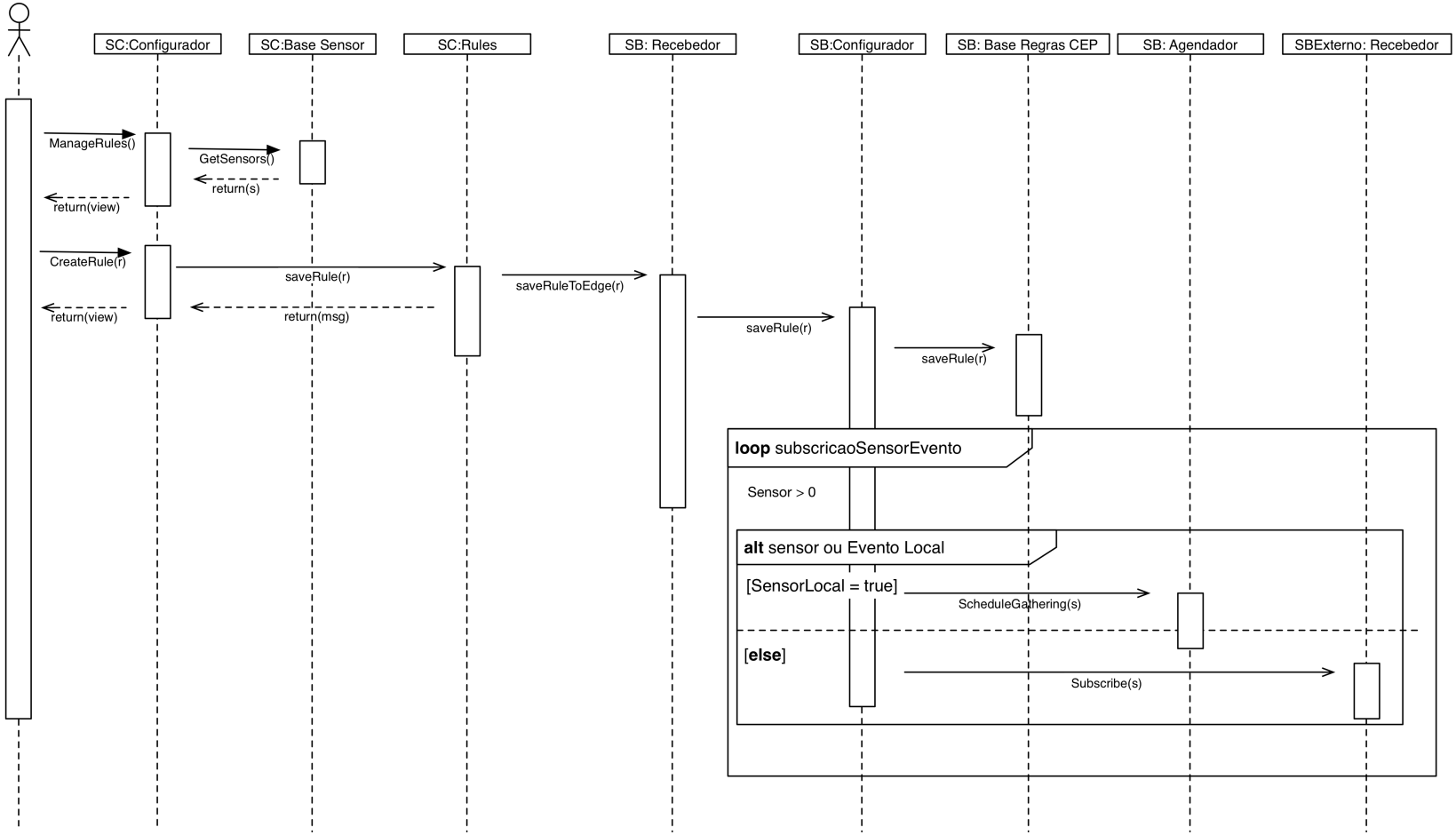


Figura 4.8 – Fluxo de Criação de Regra
Fonte: O Autor

O módulo Recebedor também trata as informações provenientes de outros Servidores de Borda, implementando uma fila de eventos de contexto com o intuito de repassá-la ao Motor de CEP e é responsável por receber as mensagens de configuração das regras CEP e enviá-las para o configurador.

Uma vez recebido os eventos e traduzidos para a linguagem interna do SB-FOG, estes eventos são instanciados nas regras armazenadas na Base de Regras CEP local do Servidor de Borda, sendo chamado o Motor CEP para efetuar o seu processamento.

Módulo Motor CEP

O Motor CEP tem como principal função na arquitetura a manipulação das regras estabelecidas para detecção de eventos de contexto e o processamento distribuído desses através dos Serviços de Processamento Distribuído que serão discutidos na secção 4.3.

Quando este módulo é ativado, ou seja, é recebido um evento de contexto o Motor CEP verifica a partir da especificação das regras quais os outros eventos de contexto deverão ser aguardados. Estes eventos poderão ser locais ou remotos, isto é, produzidos em outros Servidores de Borda, esta subscrição de eventos remotos é realizada no momento de registro da regra de processamento, conforme descrito na Figura 4.8 e é realizada automaticamente a partir da regra definida pelo usuário.

O módulo também tem responsabilidade de lidar com os estados dos eventos e armazená-los localmente quando necessário, de acordo com as especificações das regras, possibilitando uma avaliação das mesmo ao longo de um intervalo de tempo.

Após o processamento de um evento de contexto, distribuído ou não, o Motor CEP ativa o módulo de notificação, disponibilizando aos interessados a informação de que um determinado evento foi detectado e processado.

Este fluxo de processamento realizado pelo Motor CEP também está descrito na Figura 4.7.

Módulo Notificador

O modulo Notificador recebe dados diretamente do Motor CEP sob os eventos que foram detectados. O módulo notificador então verifica as subscrições existentes e notifica todos os subscritos a respeito deste evento.

Em função do resultado do processamento contextual, o Servidor de Borda irá decidir se avança ou não com a submissão no Servidor de Contexto das informações contextuais, ou

parte delas, envolvidas no processamento, nesse caso informando o módulo Publicador com as informações necessárias para a publicação.

Estas subscrições são realizadas previamente através de uma API pelos Servidores de Borda e são armazenadas na Base de Subscrições. Estes dados podem ser atualizados a qualquer momento, removendo ou adicionando subscritores aos eventos providos. As subscrições são realizadas pelos Servidores de Borda a partir das especificações das regras de processamento contextual distribuído.

Módulo Base de Regras CEP

A Base de Regras CEP é responsável por armazenar todas as regras para processamento dos eventos de contexto, distribuídos ou não, pelo Servidor de Borda.

Estas regras podem ser alteradas a qualquer momento pelos usuários do ambiente gerenciado pelo EXEHDA-FOG, sem a necessidade de recompilação de código ou interrupção de sua execução. A Base de Regras CEP é consultada pelo Motor CEP sempre que um dado contextual é recebido pelo módulo Recebedor.

Módulo Base de Subscrições

A Base de Subscrições dos Servidores de Borda é responsável por manter a informação de todos os Subscritos nos eventos providos por aquele. Uma vez que um evento de interesse seja detectado a Base é acessada para que o evento seja notificado o mais rápido possível dentro do ambiente. A Tabela de Roteamento também informa o método de notificação que deve ser utilizado de maneira que a arquitetura possa comunicar-se com outros dispositivos que não fazem parte do EXEHDA-FOG.

A Tabela de Roteamento também informa o método de notificação, por padrão o EXEHDA-FOG utiliza-se de um *broker* sob o protocolo MQTT para publicação dos eventos processados, porém a arquitetura permite que seja definido outro método de maneira a interoperar com dispositivos que não fazem parte nativamente no EXEHDA-FOG, como por exemplo uma API web sob o protocolo HTTP.

Módulo Base de Estados, Persistência Local

Este módulo da arquitetura é responsável pelo armazenamento de dados e estados para uma determinada Regra de Processamento. Por exemplo, considerando uma regras para detecção de uma situação de incêndio quando temperatura alta e o sensor de fumaça ativado sejam

detectados em uma mesma área. Quando a informação de temperatura alta é detectada o evento "Incêndio" ainda não pode ser detectado até que o sensor de fumaça ativo seja também detectado, dessa forma o Motor CEP armazena a informação do primeiro estado detectado na Base de Estados até que o sensor de fumaça ativo seja detectado ou que uma situação se altere no ambiente.

Módulo Publicador, Coleta, Interoperação, Comunicação e Demais Módulos

Os demais módulos da arquitetura, que constituem a parte legada do EXEHDA-FOG, receberam atualizações tecnológicas visando a integração com os novos módulos afim de dar pleno suporte a operação em FOG e as funcionalidades introduzidas pelo EXEHDA-FOG. Uma visão geral da interoperação realizadas por estes módulos na operação de aquisição síncrona de dados contextuais, quando a aquisição do contexto se dá devido ao um agendamento de leitura, pode ser visualizado na Figura 4.9.

No fluxo descrito na Figura 4.9 pode-se observar o **loop processaRegrasDeNegócio** onde para cada sensor que possue uma regra de processamento contextual é enviado como um evento para o **rf ProcessamentoRegra**, com seu fluxo descrito na Figura 4.7.

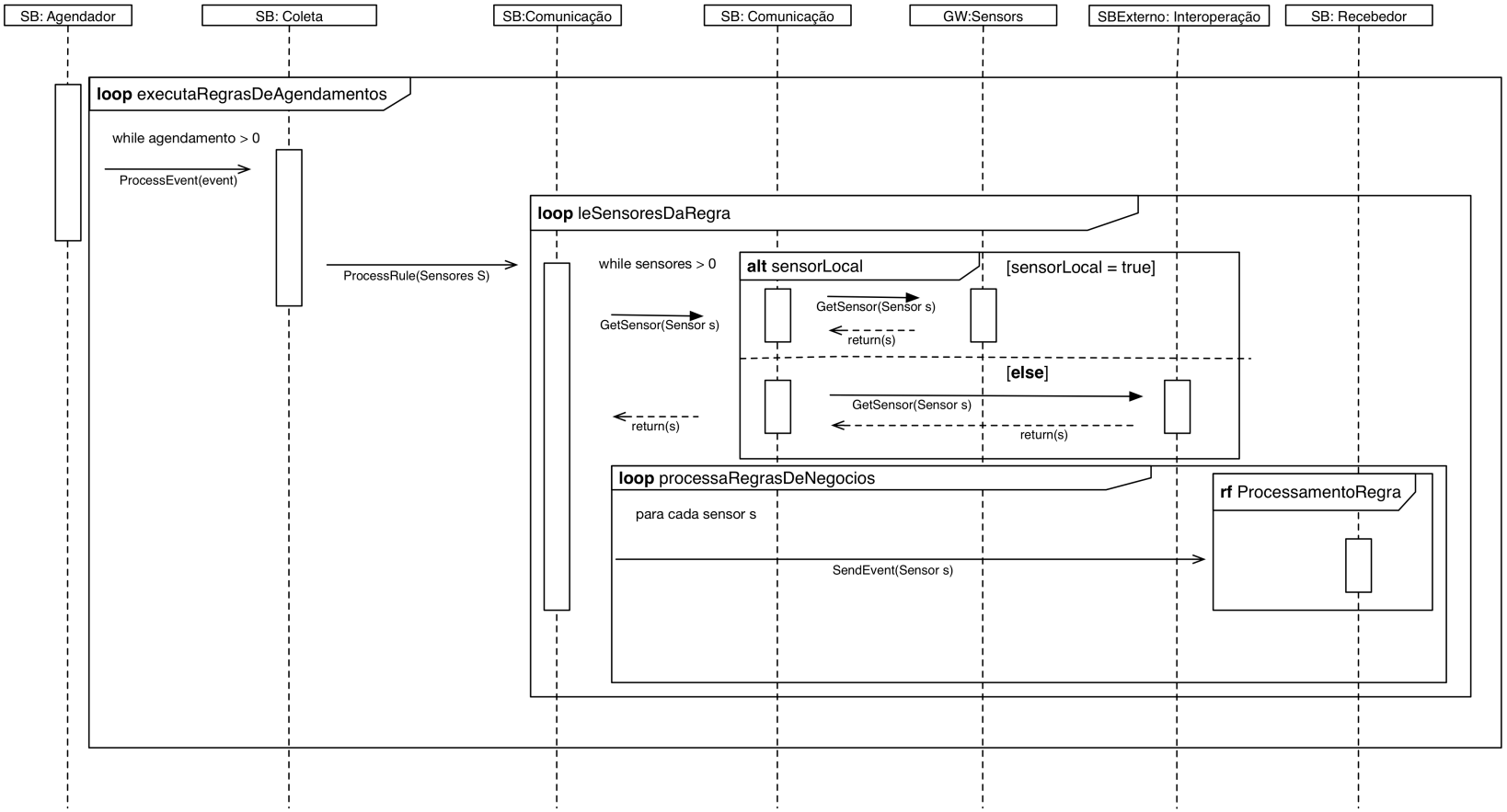


Figura 4.9 – Fluxo de Aquisição Contextual Síncrona
Fonte: O Autor

4.3 Serviços de Processamento de Eventos

Conforme discutido na seção 2.4, os sistemas distribuídos baseados em eventos necessitam prover alguns métodos para realizar o processamento de dados contextuais. De maneira a qualificar o EXEHDA-FOG a realizar este processamento foram projetados serviços capazes de realizar as tarefas mais comumente encontradas no processamento de eventos complexos com foco ao Contexto Temporal e as suas funcionalidades de Filtragem e Transformação.

Este serviços fazem parte do Motor CEP e também foram concebidos de forma a trabalhar em conjunto com os módulos Recebedor e Notificador de maneira a implementar a funcionalidade de subscrição e publicação de Eventos.

4.3.1 Serviço de Filtragem

Sistemas direcionados para IoT são capazes de gerar uma a grande quantidade de dados contextuais e podem ser necessários mecanismos para qualificação dessa informação contextual. Uma característica essencial nessa qualificação é a capacidade de realizar-se filtragem sob os dados contextuais coletados pelos sensores.

Foi desenvolvido um Serviço de Filtragem para os Servidores de Borda, onde estes podem receber regras de filtragem através de operadores relacionais(menor que, menor ou igual a, igual, diferente de, maior que, maior ou igual a). Estes operadores também podem ser combinados em regras mais complexas utilizando-se operadores lógicos **E** e **OU**, ou então a negação destes (**E não**, **OU não**) destes. A Figura 4.10 mostra um exemplo do fluxo realizado para avaliação de uma regra pelo Serviço de Filtragem configurada para execução a cada 5 minutos, por exemplo.

Foi desenvolvida uma interface web de forma a oferecer gerência dinâmica sobre a criação de regras de processamento contextual, um exemplo dessa interface com a regra previamente discutida por ser visualizada na Figura 4.11. A regra foi construída para monitorar um ambiente e informar sempre que a temperatura esteve fora da faixa de execução, para isso foi introduzida a possibilidade de combinar negação na regra de processamento contextual onde ações são realizadas apenas **se não** forem detectadas as condições especificadas na regra.

Após essa configuração de regra ela é armazenada no banco de dados do Servidor de Contexto e transmitida em formato JSON para o Servidor de Borda conforme exemplificado no

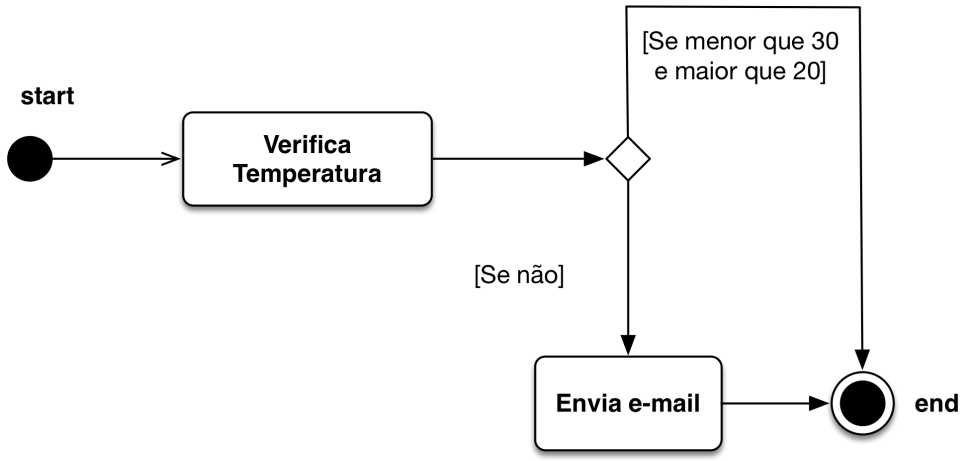


Figura 4.10 – Exemplo de Serviço de Filtragem
Fonte: O Autor

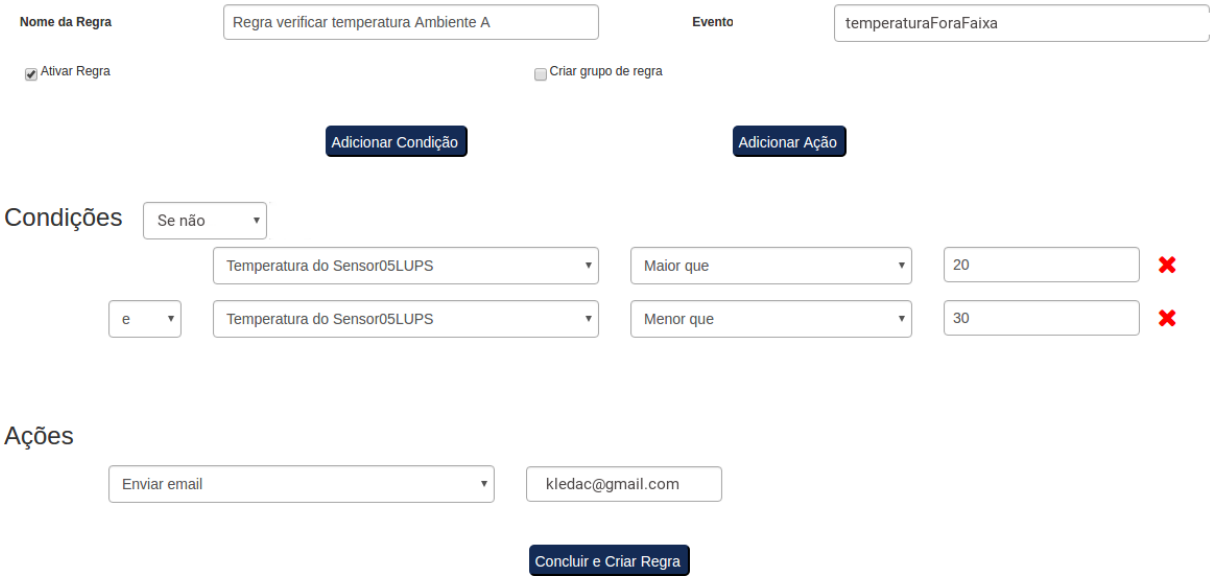


Figura 4.11 – Exemplo de Regra de Filtragem
Fonte: O Autor

fluxo da Figura 4.8. Parte do pacote de dados JSON utilizado para comunicação com o Servidor de Borda pode ser visualizado na Figura 4.12,

4.3.2 Serviço de Agregação

O Serviço de Agregação visa prover capacidade de consolidação de dados contextuais coletados ao longo de uma janela temporal, implementando assim o conceito de Contexto Temporal conforme discutido na seção 2.4. Este Serviço permite que o inicio e o final da janela temporal sejam definidos através de eventos gerados na arquitetura através das regras de

```
"rule":
{
    "contextServerId": 12,
    "edgeServerId": 3,
    "ruleName": "Regra Verificar Temperatura Ambiente A",
    "eventName": "temperaturaForaFaixa",
    "type": "boolean",
    "conditions":
    [
        {
            "ifType": "not",
            "type": "and",
            "params": [
                {
                    "processingService":"filter",
                    "data":
                    {
                        "type":"sensor",
                        "params":
                        {
                            "id":1,
                            "edgeServerId": 3,
                            "url": "local",
                            "method": ">",
                            "value": 20
                        }
                    }
                },
                {
                    "processingService":"filter",
                    "data":
                    {
                        "type":"sensor",
                        "params":
                        {
                            "id":1,
                            "edgeServerId": 3,
                            "url": "local",
                            "method": "<",
                            "value": 30
                        }
                    }
                }
            ]

        }
    ],
    "actions":[
        {
        "name":"sendMail",
            "params":{
                "0": "kledac@gmail.com"
            }
        }]
}
```


Figura 4.12 – Regra em Formato JSON
Fonte: O Autor

processamento contextual. Esta abordagem permite que sejam combinados eventos para monitoramento do ambiente gerenciado pelo EXEHDA-FOG ou mesmo alteração das regras de operação que devem ser executadas nessa janela.

Para melhor compreender o Serviço de Agregação vejamos que a Regra de filtragem

especificada na Figura 4.10 define o evento de nome **"temperaturaForaFaixa"**, sendo assim, sempre que ela for executada será gerado um status de **True**(Verdadeiro) ou **False**(Falso) como saída de sua execução a ser armazenado no banco de dados do Servidor de Borda e do Servidor de Contexto e também transmitida através do módulo Notificador para todas os interessados subscritos no resultado de sua execução. Como a regra definida na Figura 4.11 utiliza-se de uma negação das condições ela terá uma saída **True** sempre que as condições especificadas **não** forem verificadas, esse comportamento é exemplificado na Figura 4.13.

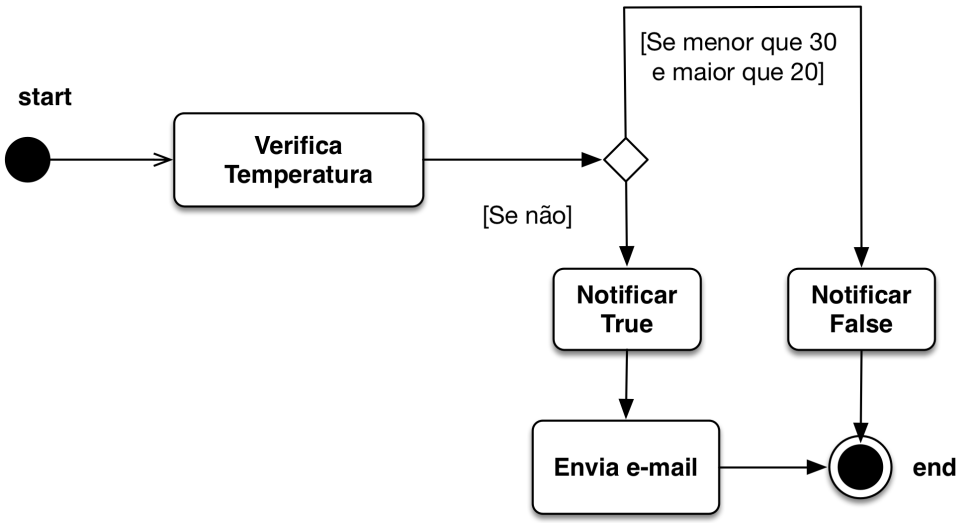


Figura 4.13 – Exemplo de Regra de Filtragem em Evento
Fonte: O Autor

A partir de uma Regra de Filtragem pode-se definir uma regra de Agregação como na Figura 4.14 capaz de modificar as Regras de operação e, também, realizar uma notificação agregada de todas vezes que a Temperatura esteve fora da Faixa esperada. Este tipo de serviço poderia ser utilizado também para realizar uma média de valores durante um determinado período ou outro tipo de operação desenvolvida para o EXEHDA-FOG.

No exemplo da Figura 4.14 sempre que a Temperatura está fora da Faixa é feito um registro de seu Estado e uma ativação de um grupo de regras de contingência, no momento em que a Temperatura retorna a seu valor esperado é realizada uma notificação por e-mail, por exemplo, com a quantidade de vezes que a Temperatura esteve fora da faixa de operação dentro desta janela temporal e também são desativadas as regras de contingência.

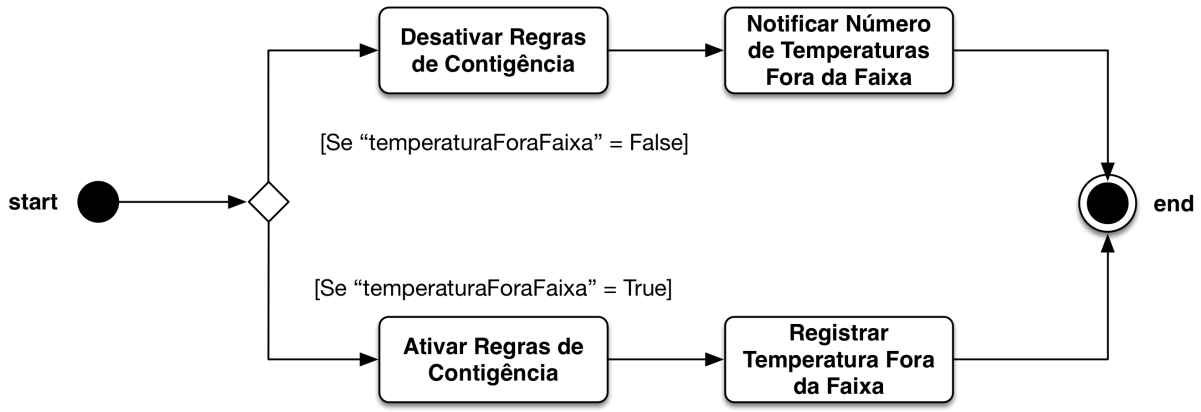


Figura 4.14 – Exemplo de Regra de Agregação
Fonte: O Autor

4.3.3 Serviço de Encadeamento

Uma característica presente nos processadores de eventos está na capacidade de observar-se encadeamento de eventos, ou seja, uma determinada sequência que necessita ser obedecida para a realização uma determinada ação. Essa funcionalidade de combinação de eventos é possibilitada no EXEHDA-FOG através do Serviço de Encadeamento, que possibilita o controle de uma sucessão de eventos. Além de permitir que seja realizada uma verificação na cumplicidade de um encadeamento de eventos, também torna-se possível verificar a situação negativa, ou seja, quando um processo específico não foi seguido. Essa característica possibilita uma detecção de falhas em processos ou operações, onde os passos necessários não foram seguidos.

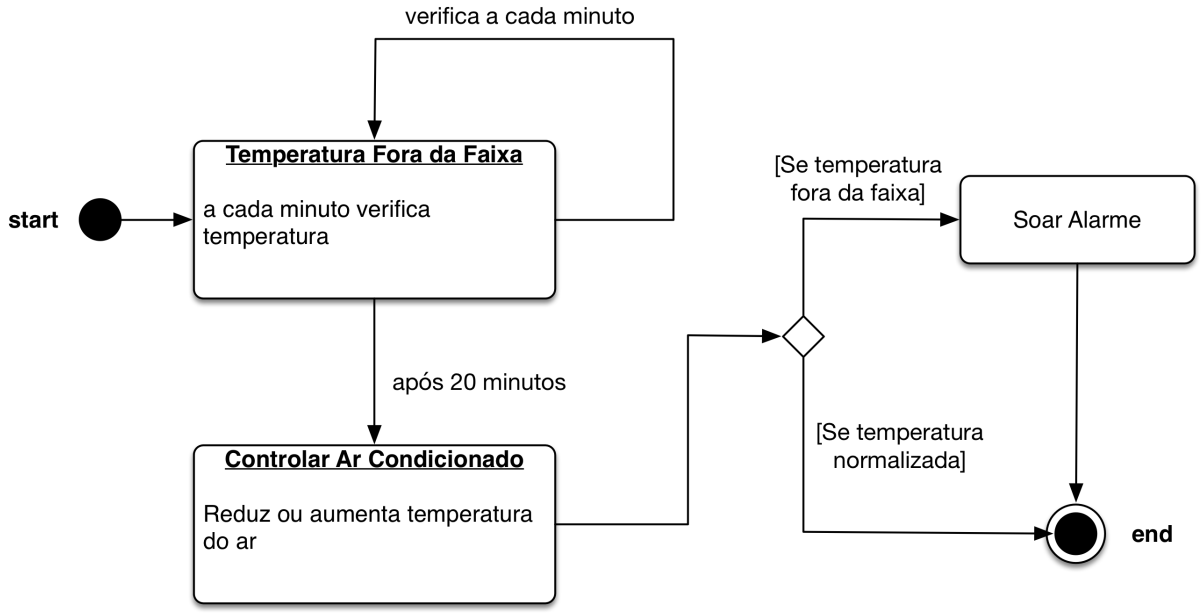


Figura 4.15 – Exemplo de Regra de Encadeamento
Fonte: O Autor

Na Figura 4.15 utiliza-se o exemplo da regra de Filtragem para o processamento de um encadeamento de eventos que deve ser seguido para controle da temperatura. Nesse caso, após ser detectada que a temperatura está fora da faixa de operação são realizadas verificações mais frequentes, passados 20 minutos em que a temperatura ainda está fora da faixa é realizada uma ação de controle da temperatura, caso essa não surja efeito na temperatura é então soado um alarme sonoro, pois a sequência não teve sucesso por alguma falha nos sensores de temperatura ou do atuador, nesse caso o ar condicionado.

4.4 Tecnologias Utilizadas

Nesta secção serão abordadas as tecnologias utilizadas no desenvolvimento dos módulos da arquitetura do EXEHDA-FOG, visando um maior detalhamento no funcionamento da arquitetura e a sua contribuição ao Subsistema de Reconhecimento de Contexto e Adaptação.

4.4.1 Agendador

O agendador é um componente presente no **Processador de Contexto** dos Servidores de Borda que permite o agendamento de tarefas de aquisição e processamento de contexto a serem executados no ambiente gerenciado pelo EXEHDA-FOG, fornecendo também a capacidade de controlar as janelas temporais e a sua interoperação com o Serviço de Agregação. Estas tarefas podem serem agendadas como execução única ou periódica, dependendo da configuração da regras definida pelo usuário. O agendador foi implementado utilizando-se o Advanced Python Scheduler (APScheduler), uma biblioteca de agendamento desenvolvida em Python.

Os agendamentos são armazenados em um banco de dados centralizado, no Servidor de Contexto, e em um banco de dados local, no Servidor de Borda de destino dos agendamentos, de maneira que caso um Servidor de Borda falhe este possa ser reconfigurado. Também é possível alterar, adicionar ou remover regras de agendamento sem que a execução do EXEHDA-FOG seja interrompida, caso uma regra já esteja em execução ela termina seu ciclo de execução, sendo removida ou alterada após sua finalização.

O agendador trabalha em conjunto com o motor de regras CEP para implementar o conceito de contexto temporal, como visto na seção 2.4.2, no EXEHDA-FOG, possibilitando assim um controle sob os eventos gerados no ambiente monitorado. Estes eventos são de natureza síncrona, ou seja, eles resultam de um pedido expresso do agendador de leitura de algum dado

contextual do ambiente para os Gateways ou de uma atuação periódica, porém é possível que através desses eventos sejam disparados eventos assíncronos dependendo das regras especificadas para o ambiente. Eventos assíncronos também podem ser gerados pelos Gateways que possuem gatilhos de valores, ou seja, no caso de uma variação muito grande em um dado contextual o Gateway pode enviar automaticamente este dado para o seu Servidor de Borda e este dado será repassado diretamente pro motor de regras CEP, sem intervenção do Agendador.

4.4.2 Codeigniter

O **Codeigniter** é um framework PHP open-source, no modelo Model-View-Controller (MVC), desenvolvido com o intuito de facilitar o desenvolvimento de aplicações Web. Através deste framework foi desenvolvida a aplicação Web do Servidor de Contexto capaz de receber as configurações necessárias para o EXEHDA-FOG e suas regras de processamento contextua.

Através das interfaces é possível alterar agendamentos, regras de execução, configuração dos Servidores de Borda e sensores monitorados, etc. O framework também interage no armazenamento e distribuição das regras através de serviços RESTful, interagindo assim com o framework **Django** utilizado nos Servidores de Borda.

4.4.3 Django

Django é um Web framework, no modelo MVC, implementado na linguagem de programação Python que prove uma infra-estrutura para aplicações web, abstraindo características como a comunicação, gerenciamento de dados, entre outras funcionalidades. Este framework foi utilizado para construir as ferramentas necessárias da API desenvolvida para o Servidor de Borda de maneira a prover as funcionalidades introduzidas pelo EXEHDA-FOG.

Através da interface do EXEHDA-FOG é possível configurar o ambiente gerenciados e seus diversos dispositivos, assim como criar/alterar/remover as regras de eventos que monitoram o ambiente.

A estrutura basica do Django e composta por:

- **models.py** é o arquivo contem uma descricao da tabela do banco de dados,como uma classe Python. Isso e chamado de modelo. Usando Esta classe, voce pode criar, recuperar, atualizar e excluir registros em seu banco de dados usando o codigo Python simples do que escrever instrucoes SQL repetitivas;

- **views.py** contem a logica de funcionamento da pagina acessada;
- **urls.py** e o arquivo que especifica qual vista e chamada para um determinado padrao de URL.

As requisições solicitadas para o Web framework Dangjo, tem como formato arquitetural APIrestful.

4.4.4 Business Rules: Processador de eventos

Durante a fase de análise dos motores de regras CEP foram avaliadas algumas ferramentas, como por exemplo Drools Fusion e o Esper, porém estes exigem hardwares com alta capacidade de processamento e não se mostraram propícios para os dispositivos com capacidades restritas, típicos da IoT, como, por exemplo, o hardware de Servidor de Borda que vem sido utilizado no EXEHDA-FOG , como pode ser visto no Anexo A.

Através de pesquisas chegou-se em uma solução capaz de aliar performance com a funcionalidade desejada, o ***Business Rules***[1], um motor de regras implementado em Python que avalia condições através de variáveis estabelecidas e comparando-as a um valor estipulado pelo usuário. Diferentemente dos motores mencionados anteriomente, o ***Business Rules*** trabalha com Python e é de código aberto, possibilitando assim a sua alteração para fins específicos.

Para os serviços desenvolvidos para processamento contextual foram realizadas alterações no padrão de processamento contextual do motor CEP, na Figura 4.16 pode-se visualizar um exemplo de alteração realizada para processamento de condições de uma regra contextual.

4.4.5 Mosquitto

Eclipse Mosquitto (LIGHT, 2013) é um broker de mensagens open-source que trabalha sob o protocolo MQTT para distribuir mensagens leves utilizando-se do protocolo publicador/-subscritor. Devido a seu protocolo de baixo consumo e a utilização de um protocolo leve, Mosquitto é um ótimo broker para se utilizar em ambientes de IoT onde os componentes, normalmente, possuem baixa capacidade computacional.

O Mosquitto é capaz de lidar com mediana capacidade de suporte a conexões de subscritores, se comparado a algumas alternativas como **eMQTT**, porem ele apresenta um pequeno

[1] https://github.com/venmo/business-rules

```
def check_condition(condition, defined_variables):
    """ Checks a single rule condition - the condition
    will be made up of variables, values,
    and the comparison operator. The
    defined_variables object must have
    a variable defined for any variables
    in this condition. """
    name, op, value,parameters = condition['name'],
        condition['operator'], condition['value'],
        condition['parameters']
    parameters = json.dumps(parameters)
    operator_type = _get_variable_value(defined_variables, name,
        parameters)
    return _do_operator_comparison(operator_type, op, value)
def _get_variable_value(defined_variables, name,parameters):
    def fallback(*args, **kwargs):
        raise AssertionError('Variable {0} is not defined
        in class {1}'.format(
                name, defined_variables.__class__.__name__))

    method = getattr(defined_variables, name,fallback)
    val = method(parameters)
    ret = method.field_type(val)
    return ret
```

Figura 4.16 – Exemplo de método modificado no Business Rules
Fonte: O Autor

consumo de memória e CPU se comparado com outros brokers, portanto consumindo menos energia e sendo a ferramenta ideal para comunicação através do hardware utilizado para os Servidores de Borda (TORRES; ROCHA; SOUZA, 2016).

No EXEHDA-FOG o broker é instalado nos Servidores de Borda, habilitando-os a comunicarem-se com diversos dispositivos nativos da arquitetura e a dispositivos externos e está integrado ao módulo de Comunicação. As regras criadas através da interface para o ***Business Rules*** são consideradas eventos na arquitetura e, dependendo da vontade do usuário, podem ser transmitidos para os interessados através do broker. Efetivamente cada Servidor de Borda capacita-se como um agente processador de contexto, capaz de colaborar com o processamento contextual através de uma troca de mensagens e subscrição em eventos.

4.5 Considerações Sobre o Capítulo

Neste capítulo é apresentado o *middleware* EXEHDA e os módulos que fazem parte de sua arquitetura. Em seguida é apresentado o EXEHDA-FOG, uma proposta que capacita o EXEHDA a realizar o processamento contextual distribuído e colaborativo através dos conceitos de processamento de eventos utilizando-se de um modelo publicador/subscritor, o que também possibilita uma gerência dinâmica sobre estes eventos.

Os módulos do EXEHDA-FOG são apresentadas e seus procedimentos de interação entre eles, assim como, as tecnologias utilizadas na concepção das funcionalidades.

5 EXEHDA-FOG: ESTUDO DE CASO

Neste capítulo está apresentado o estudo de caso desenvolvido com o objetivo de explorar funcionalidades propostas para o EXEHDA-FOG. As funcionalidades exploradas no estudo de caso contemplam tarefas relacionadas a aquisição e processamento de dados contextuais distribuídos, bem como a gerência e distribuição de eventos no ambiente provido pelo EXEHDA-FOG.

O estudo de caso trata as contribuições possíveis com a utilização do EXEHDA-FOG para a área da Viticultura de Precisão (VP), uma área que possui um grande potencial econômico e social, em diversas regiões do Brasil, e em particular do Rio Grande do Sul, motivando um aporte significativo de pesquisas e investimentos em tecnologia direcionados a gerência dinâmica e autônoma dos seus vinhedos e/ou áreas de produção agrícola associadas (PINHO; BOAVENTURA-CUNHA; MORAIS, 2015).

5.1 Hardware e Software Utilizados

Na avaliação do EXEHDA-FOG, foram priorizadas tecnologias escaláveis, robustas e de código aberto.

O hardware utilizado no estudo de caso foi um equipamento tipo *desktop* com processador Intel Pentium i5-2310 2.9GHz de quatro núcleos, com 8Gb de memória RAM e com o Sistema Operacional Ubuntu Server. A escolha da distribuição Linux Ubuntu foi decorrente da sua ampla utilização da mesma pelo grupo de pesquisa em seus equipamentos servidores, o que potencializa a exploração de funcionalidades avançadas e o seu suporte continuado.

Os hardwares empregados nos Servidores de Borda, Gateways e Sensores são operacionalizados em ambiente de emulação. As especificações destes hardwares seguem os padrões que estão sendo utilizados pelo Grupo de Pesquisa G3PD. Uma caracterização dos mesmos, está disponível no Anexo A.

Os *drivers* para a captação de dados dos sensores podem ser acionados através de uma requisição `pull` de leitura pelo Servidor de Borda, ou por regras que exploram a variação das grandezas físicas medidas diretamente nos Gateways, e cujos valores serão propagados via `push` para o Servidor de Borda. O Servidor de Borda poderá então realizar o tratamento necessário para a informação contextual obtida através do meio físico. As trocas de mensagens ocorrem, principalmente, sob dois protocolos, o HTTP através de APIs REST, e o protocolo MQTT para publicação de eventos relativos as regras de controle do ambiente monitorado.

O ambiente de emulação empregado é denominado Common Open Research Emulator (CORE), e está descrito a seguir. Após os diferentes testes, foi possível constatar que o hardware empregado facultou a emulação dos diferentes dispositivos necessários para realizar o estudo de caso concebido.

Ambiente de emulação CORE

O ambiente computacional concebido para avaliação do EXEHDA-FOG foi emulado com o auxílio do emulador CORE (AHRENHOLZ et al., 2008). O CORE[1] é um framework *open-source* desenvolvido pela *Boeing Research and Development Division* (BR&T)[2] que visa a emulação de ambientes computacionais utilizando-se de uma virtualização de pilha de rede FreeBSD, provendo assim testes sob ambientes computacionais largamente distribuídos sem a necessidade de implantações de hardwares reais. Os ambientes computacionais providos pelo CORE são emulados dinamicamente a medida que a execução se desenvolve, possibilitando assim uma conexão entre ambientes reais e emulados, além disso o CORE possibilita uma emulação distribuída onde múltiplos computadores podem colaborar para a emulação de um ambiente mais complexo.

O CORE fornece uma interface de usuário provida através da Linguagem de Comandos de Ferramentas/Toolkit (Tcl/Tk) que permite customização e desenvolvimento de ferramentas de interface para suas diferentes funcionalidades. Através desta interface é possível definir a configuração do ambiente computacional posicionado os dispositivos e determinando suas interconexões. Também é possível emular diversas características operacionais de um ambiente computacional, como por exemplo: limite de largura de banda, latência, perda de pacotes e etc. A figura 5.1 exibe uma visão geral do ambiente de emulação provido pelo CORE.

Através de virtualização o CORE possibilita que múltiplas instâncias virtuais sejam executadas simultaneamente, fazendo assim com que cada dispositivo emulado possuam sua própria instância privada. Estas instâncias recebem o nome de **vimages** e, diferentemente das tradicionais máquinas virtuais, não possuem um sistemas operacional inteiro rodando no hardware emulado, ao invés disso elas executam o mesmo kernel e compartilham o sistema de arquivos, processador, memória, clock e outros recursos disponíveis. Assim os dispositivos emulados podem ser projetados em diversas linguagens de programação, como por exemplo Python, uma

[1] https://www.nrl.navy.mil/itd/ncs/products/core
[2] http://www.boeing.com.au/products-services/research-technology.page

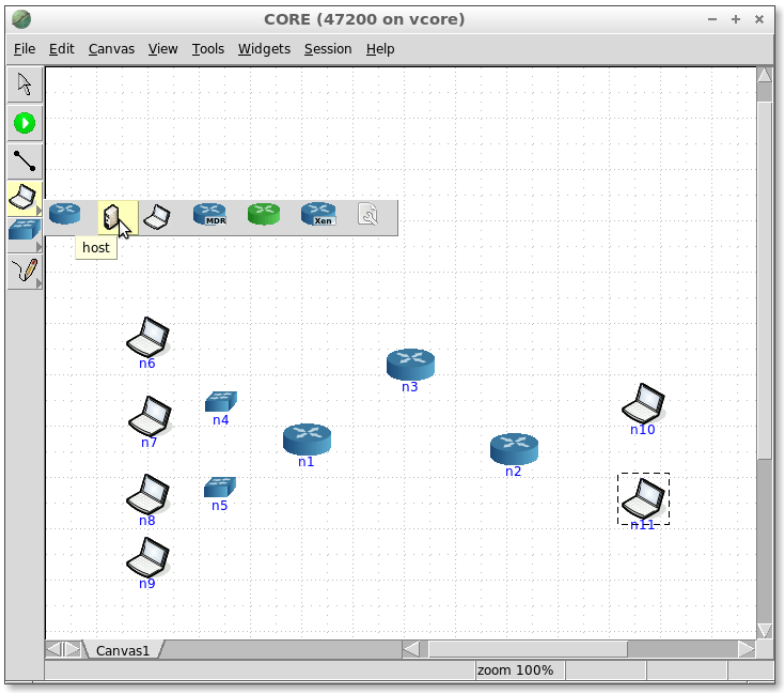


Figura 5.1 – Ambiente de emulação do CORE
Fonte: Ahrenholz et al. (2008)

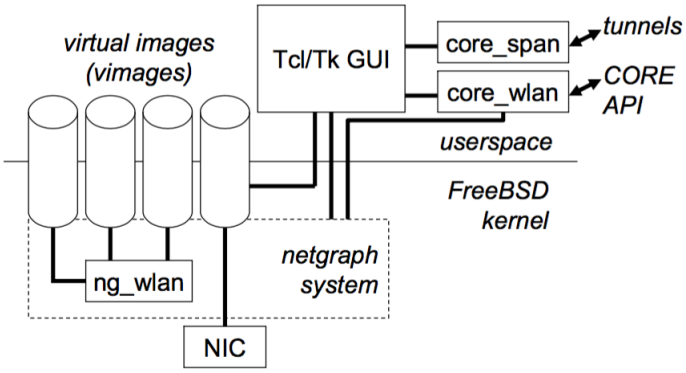


Figura 5.2 – Visão Geral dos componentes do CORE
Fonte: Ahrenholz et al. (2008)

vez que estas linguagens estejam devidamente instaladas na instância inicial de emulação, a visão geral dos componentes do CORE está caracterizada na Figura 5.2. Assim tornou-se possível a emulação de diversos Servidores de Borda e Gateways independentes neste ambiente, onde estes são uma emulação de seu respectivo objeto real no ambiente gerenciado pelo EXEHDA-FOG.

5.2 Área do Estudo de Caso: Viticultura de Precisão

A Viticultura de Precisão (VP), tal como a agricultura de precisão (AP), pode ser entendida como a gestão da variabilidade temporal e espacial das áreas cultivadas com o objetivo de

melhorar o rendimento econômico da atividade agrícola, quer pelo aumento da produtividade e/ou qualidade, quer pela redução dos custos de produção, reduzindo também o seu impacto ambiental e risco associado. Variabilidade temporal e espacial diz respeito à característica da área agrícola que se refere às variações dos fatores importantes às culturas (físico/químicos do solo ou climáticos) ao longo da área plantada e do tempo de desenvolvimento da planta (BRAGA; PINTO, 2009).

Nesse sentido, uma das principais técnicas utilizadas na agricultura de precisão é a tecnologia de taxa variável. Essa técnica consiste em uma detalhada caracterização da variabilidade espacial da cultura e em seguida na gestão de fatores de produção (rega, fertilizantes, etc.) e operações culturais (BRAGA; PINTO, 2009).

Uma das questões fundamentais na produção de vinhos de alta qualidade é o momento certo de irrigar. O uso da água por um vinhedo varia conforme o estágio de desenvolvimento da cultura. Há, normalmente, uma baixa demanda no início da fase de crescimento, devido à menor área foliar das plantas. Segue-se um período de alta demanda que ocorre quando o dossel está plenamente desenvolvido. Isso caracteriza a necessidade de mudança das regras que controlam os sistemas de irrigação ao longo do tempo (CONCEIÇÃO, 2008).

Outro aspecto importante em relação ao gerenciamento de sistemas de irrigação, diz respeito à capacidade do solo em reter a água. Nesse aspecto, os solos podem ser classificados como de alta, média e baixa capacidade de retenção de água, havendo ainda outros tipos intermediários. Dependendo da extensão do vinhedo, esses vários tipos de solo podem estar presentes. Para não prejudicar o desenvolvimento e a produção de frutos das videiras, deve-se evitar que a reserva hídrica do solo se esgote. Portanto, as estratégias de gerenciamento da irrigação devem variar espacialmente ao longo da área cultivada (CONCEIÇÃO, 2008).

A partir das informações referentes aos atributos físicos e químicos do solo, determinados pela análise de amostras coletadas ao longo de toda área cultivada, são determinadas as Zonas de Manejo. Zona de manejo consiste em uma porção da área cultivada em que o solo possui característica hídrica aproximadamente homogênea e por isso permite que seja adotada a mesma estratégia de irrigação. Isso significa dizer que o momento de irrigar e a quantidade de água utilizada é a mesma ao longo de toda Zona de Manejo, assim não se teria falta nem excesso de água em nenhum ponto. A quantidade de água total a ser distribuída uniformemente na Zona de Manejo é determinada pelo especialista responsável tendo em vista a capacidade de campo e o valor da tensão de água no solo no momento em que foi iniciada a irrigação.

5.3 Aspectos considerados no Estudo de Caso

Para determinar o momento correto de irrigar na proposta de Viticultura de Precisão é necessário a avaliação de diversas variáveis físicas. No cenário considerado, as variáveis contextuais monitoradas são as seguintes:

- tensão de água no solo: a tensão pode ser obtida através de Tensiômetros acoplados aos Gateways;
- temperatura e umidade do ar: a temperatura e umidade do ar podem ser obtidas através de sensores 1-Wire conectados a Gateways, alocados em posições estratégicas no interior dos vinhedos.

Os comandos de atuação disparados através do processamento de regras para tratamento contextual podem ter as seguintes natureza:

- alertas visuais;
- alertas sonoros;
- envio de mensagens (SMS/e-mail);
- ativação de transdutores elétricos para acionamento dos sistemas de irrigação.

O envio de mensagens deve ser processado pelo Servidor de Contexto por necessitar de informações de contato dos usuários, enquanto os demais alertas poderiam ser processados nos próprios Servidores de Borda (infraestrutura de FOG) quando necessário. Segundo as regras especificadas através do aplicativo de configuração do EXEHDA-FOG estes alertas são considerados eventos do ambiente monitorado e são repassados para o Servidor de Contexto quando necessário.

A Figura 5.3 representa um vinhedo típico da região sul do Brasil com suas respectivas Zonas de Manejo. Visando uma qualificação da informação contextual obtida e a respectiva validação de sua consistência foi considerado o uso de diversos sensores distribuídos ao longo das diferentes Zonas de Manejo da propriedade. Em função disso, cada Zona de Manejo é equipada com um Servidor de Borda para gerenciamento dos Gateways necessários para ativação dos sensores atribuídos a esta zona.

Figura 5.3 – Zonas de manejo em um parreiral
Fonte: O Autor

Com a infraestrutura de FOG provida pelos Servidores de Borda distribuídos, pretende-se evitar que o controle de irrigação, que se dá em nível de Zona de Manejo, não seja inviabilizado por eventuais quedas de conexão de rede e a consequente perda de contato com o Servidor de Contexto. Por sua vez, com a tomada de decisões podendo acontecer na infraestrutura de FOG, mesmo que parcialmente, se tem a premissa de preservar a largura de banda do canal de comunicação de longa distância evitando publicações desnecessárias.

Uma atividade coordenada entre os Servidores de Borda vizinhos também foi desenvolvida afim de determinar o momento de irrigar, distribuindo assim o processamento de informações contextuais entre os Servidores de Borda. O Servidor de Contexto fica localizado distante da lavoura, por exemplo na sede da fazenda, no escritório, ou mesmo em em algum servidor localizado na *Cloud*. Ainda na figura 5.3 é possível visualizar os Servidores de Borda e os Gateways já distribuídos nas suas Zonas de Manejo.

Com este cenário é avaliada a arquitetura do EXEHDA-FOG no que diz respeito a sua operação distribuida. Atualmente, parte das áreas rurais dependem de conexões com a Internet providas através da rede de telefonia celular, tendo características operacionais sujeitas à conexões instáveis, baixa largura de banda, ou ainda volume total de dados mensais limitados através de franquias. Esses aspectos apontam para a necessidade de um uso otimizado do tráfego de dados através da Internet, bem como o emprego de estratégias que garantam o processamento dos dados contextuais provenientes dos vinhedos em momentos que o acesso ao Servidor de Contexto não é possível.

Configuração e Regras de Controle

Para realização do estudo de caso foi considerada uma distribuição de Gateways e sensores ao longo da grande área física típica de um vinhedo. Esta distribuição de Servidores de Borda foi emulada através do CORE, considerando Zonas de Manejo, onde cada uma é composta por:

- **Servidor de Borda:** cada Zona de Manejo é gerenciada por um Servidor de Borda sendo responsável pela centralização dos dados contextuais coletados e pela gerência autônoma de irrigações a serem realizadas;
- **Gateway:** cada Servidor de Borda gerencia 6 Gateways dentro da sua Zona. Os Gateways interagem via WIFI com o Servidor de Borda e fisicamente com os sensores, por isso, necessitam de uma distribuição maior ao longo da Zona de Manejo;
- **Sensoriamento:** cada Gateway coordena a operação 5 sensores de tensão de solo (tensiômetros).

Com a organização acima tem-se no estudo de caso 180 sensores, 36 Gateways, 6 Servidores de Borda e 1 Servidor de Contexto sendo emulados através do CORE. A distribuição dos Gateways e Servidores de Borda ao longo do Vinhedo é feita conforme apresentado na Figura 5.4. Essa figura resume o esforço de concepção no ambiente provido pelo CORE para acomodar de forma distribuída os Servidores de Borda e os Gateways conectados aos mesmos. Através da interface gráfica é possível ver as interconexões da infraestrutura de FOG, bem como ter acesso aos seus aspectos operacionais clicando sobre os diferentes ícones.

A tensão de água no solo de cada Zona de Manejo é monitorada e comparada com valor da tensão crítica da cultura que corresponde ao valor de tensão em que se deve proceder a irrigação. O valor de tensão crítica para Uvas Viníferas está entre 15kPa a 50kPa. Estes valores são responsáveis por determinar o momento correto de realizar a irrigação do vinhedo, o valor mais baixo da tensão crítica deve ser usada em situações de elevada evapotranspiração, perda de água do solo por evaporação e a perda de água da planta por transpiração, e ao se atingir o valor mais alto deve-se sempre realizar a irrigação. O valor da tensão de água do solo de uma Zona de Manejo é obtida através do cálculo da média dos valores obtidos através dos sensores para aquela zona.

Os dados utilizados na emulação foram obtidos a partir de dados reais obtidos em parceria com a EMBRAPA Clima temperado(Figura 5.5) e, a partir destes, foi possível implementar

Figura 5.4 – Estudo de caso CORE
Fonte: O Autor

cenários de testes visando uma avaliação da operação do EXEHDA-FOG , principalmente, no que diz respeito ao processamento contextual distribuído as bordas computacionais e capacidade de redução de volume de dados transmitidos via Internet.

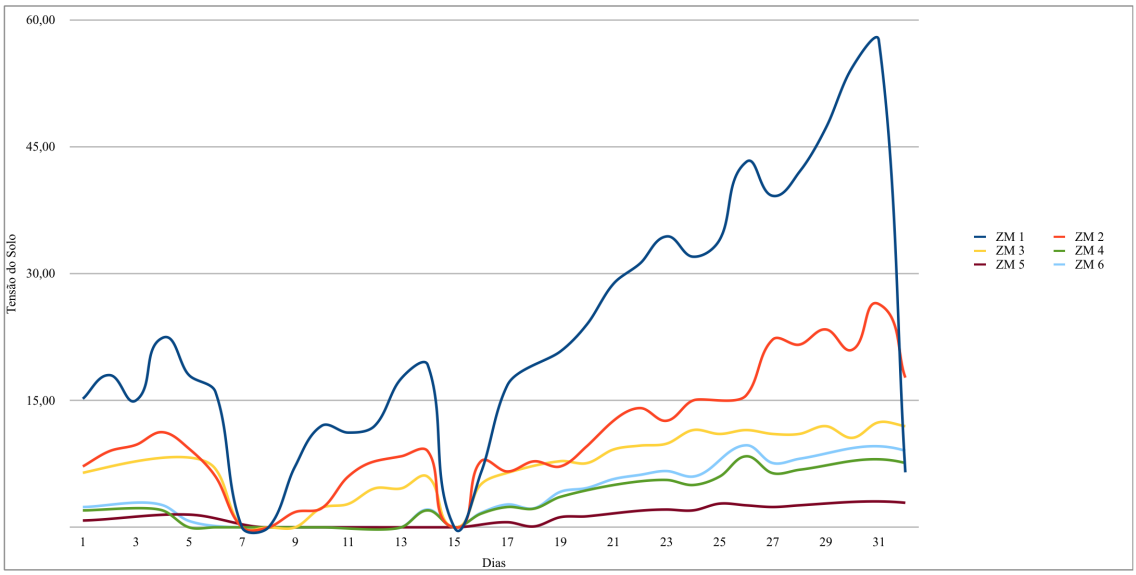


Figura 5.5 – Gráfico da Tensão de Solo
Fonte: O Autor

A lógica de controle proposta para o estudo de caso está descrita a seguir:

- **SE** o valor da tensão de água do estiver abaixo do valor considerado crítico **ENTÃO** a irrigação é imediatamente acionada e alertas são gerados (luminoso, sonoro, mensagens SMS/e-mail);

- **SE** o valor medido estiver abaixo do valor considerado baixo, mas não crítico, **SE** a temperatura ambiente está alta **SE** a umidade relativa do ar estiver baixa **SE** alguma Zona de Manejo vizinha já tenha identificado valor crítico de umidade do solo **ENTÃO** a irrigação é imediatamente acionada e alertas são gerados (luminoso, sonoro, mensagens SMS/e-mail).

Partindo da lógica de controle foram criadas regras para o ambiente emulado descrito. As regras de controle especificadas na Figura 5.6 estão especificadas em função do Servidor de Borda da Zona de Manejo 1. Os demais Servidores de borda possuem regras similares.

Nome da Regra	Evento	Condição	Ação
tensãoCríticaAltaZN1	tensãoMédiaCalcZN1	tensãoMédia>50k	Irriga e notifica
tensãoCríticaBaixaZN1	tensãoMédiaCalcZN1	tensãoMédia>15k e evapoTranspiração==alta	Irriga e notifica
tensãoCríticaBaixa2	tensãoCríticaAltaExternaZN2	tensãoMédia>15k	Irriga e notifica
tensãoCríticaBaixa3	tensãoCríticaAltaExternaZN3	tensãoMédia>15k	Irriga e notifica
tensãoCríticaBaixa4	tensãoCríticaAltaExternaZN4	tensãoMédia>15k	Irriga e notifica
tensãoCríticaBaixa5	tensãoCríticaAltaExternaZN5	tensãoMédia>15k	Irriga e notifica
tensãoCríticaBaixa6	tensãoCríticaAltaExternaZN6	tensãoMédia>15k	Irriga e notifica

Figura 5.6 – Regras Zona de Manejo 1
Fonte: O Autor

Ao ser criada uma regra ela se torna um evento composto que pode ser usado para disparar outras regras, operando de forma idêntica ao tratamento dado aos eventos primitivos. O nome da regra será a forma de acesso a esse evento que também pode ser acessado remotamente através de uma subscrição. Assim, Servidores de Borda remotos, ao se subscreverem em um evento remoto, terão uma regra remota que irá tratar esse evento. Já no Servidor de Borda local irá existir um evento vinculado ao evento remoto que poderá ser usado para disparar regras localmente.

5.4 Avaliações Realizadas

Tendo por base que uma das características centrais dos sistemas baseados em FOG está na sua capacidade de realizar qualificações/transformações nos dados contextuais a nível de borda computacional, ou seja, antes desses dados serem transmitidos ao Servidor de Contexto, fazendo assim com que possam ser minimizadas os custos associados às transmissões de longa distância. De forma a avaliar essa capacidade da arquitetura, no que diz respeito a filtragem e transformação de informações contextuais, conforme previamente discutido na seção 2.4.3, foram realizados testes com o EXEHDA-FOG relativos ao volume de dados publicados através da Internet, considerando o emprego ou não dos recursos de FOG Computing.

Contexto de Interesse: Viticultura SB1 Sensor: Sensor-29

16/12	
Méd.	19,16 kPa
Máx.	22,38 kPa
Mín.	16,69 kPa
17:30	16,69 kPa
17:00	16,78 kPa
16:30	16,88 kPa
16:00	16,98 kPa
15:30	17,08 kPa
15:00	17,17 kPa
14:30	17,27 kPa
14:00	17,37 kPa
13:30	17,47 kPa
13:00	17,56 kPa
12:30	17,66 kPa
12:00	17,76 kPa
11:30	17,85 kPa
11:00	17,95 kPa
10:30	18,11 kPa
10:00	18,32 kPa
9:30	18,53 kPa
9:00	18,75 kPa
8:30	18,96 kPa
8:00	19,18 kPa
7:30	19,39 kPa
7:00	19,60 kPa

15/12	
Méd.	17,44 kPa
Máx.	21,71 kPa
Mín.	15,05 kPa
23:30	21,71 kPa
23:00	21,35 kPa
22:30	20,99 kPa
22:00	20,63 kPa
21:30	20,27 kPa
21:00	19,91 kPa
20:30	19,55 kPa
20:00	19,19 kPa
19:30	18,83 kPa
19:00	18,47 kPa
18:30	18,11 kPa
18:00	17,75 kPa
17:30	17,39 kPa
17:00	17,03 kPa
16:30	16,67 kPa
16:00	16,31 kPa
15:30	15,95 kPa
15:00	15,59 kPa
14:30	15,23 kPa
14:00	15,05 kPa
13:30	15,20 kPa
13:00	15,34 kPa

14/12	
Méd.	15,88 kPa
Máx.	16,80 kPa
Mín.	15,21 kPa
23:30	16,80 kPa
23:00	16,67 kPa
22:30	16,53 kPa
22:00	16,40 kPa
21:30	16,26 kPa
21:00	16,12 kPa
20:30	15,99 kPa
20:00	15,85 kPa
19:30	15,72 kPa
19:00	15,58 kPa
18:30	15,44 kPa
18:00	15,31 kPa
17:30	15,21 kPa
16:00	15,52 kPa
15:30	15,38 kPa
15:00	15,25 kPa

Figura 5.7 – Aplicação web para vizualização de dados contextuais
Fonte: O Autor

Os dados contextuais obtidos, podem ser visualizados no EXEHDA-FOG empregando uma aplicação web desenvolvida especificamente para esta finalidade. Na Figura 5.7 é possível visualizar os dados numericamente exibidos em uma tabela, por sua vez na Figura 5.8 estes

dados podem ser visualizados através de interface com exibição gráfica. Os dados empregados nas avaliações realizadas a seguir foram obtidos através dessas aplicações de visualização de resultados.

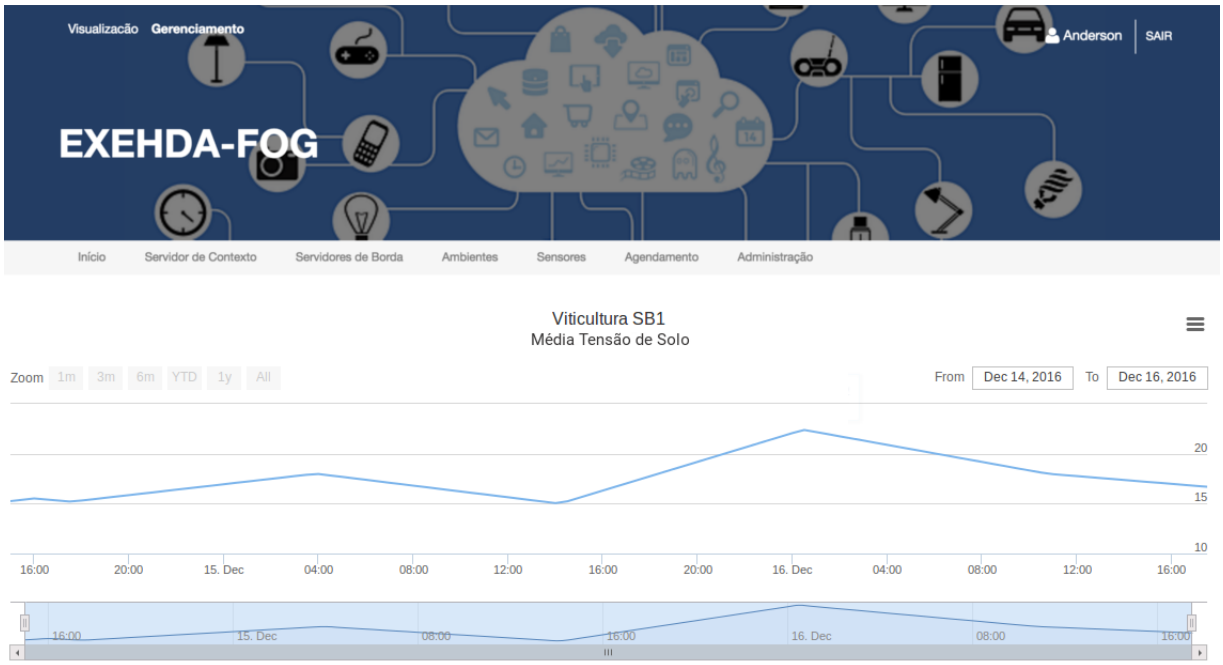


Figura 5.8 – Aplicação web para vizualização de dados contextuais
Fonte: O Autor

5.4.1 Leituras Regulares

A avaliação foi realizada no ambiente da Figura 5.4 considerando as 6 Zonas de Manejo ativas. No procedimento de avaliação empregado foi considerada uma aquisição dos dados de sensores de tensão de solo a cada de 5 minutos. Abaixo uma descrição das operações realizadas:

- **Operação com FOG**
  - realiza uma operação de filtragem e agregação de dados contextuais, nesse caso, uma publicação da média dos valores de tensão do solo coletados é realizada no Servidor de Contexto, em períodos de 1 hora para fins de histórico;

– de maneira a qualificar o controle do ambiente, sempre que um evento de tensão crítica alta ou tensão crítica baixa ocorrer, conforme definidos através das regras na Figura 5.6, as operações de publicação dos eventos e média da tensão do solo para o Servidor de Contexto são antecipadas, de maneira a preservar-se o histórico e registro de que um evento ocorreu e foi necessária uma atuação sob o ambiente.

- **Operação sem FOG**

  – publica todos sensores quando a medição é realizada, média é realizada no Servidor de Contexto, nesse caso também não há a noção de eventos, ou seja, não há históricos de que houve ou não irrigação, pois nesse caso não há suporte a registro de eventos originados de regras;

  – recebe toda comunicação sobre estado crítico de uma zona e dispara atuação remota, porém esse estado não é armazenado em uma persistência local ou remota.

	Sem Fog	Com Fog
1 hora	0,7MBytes	0,004MBytes
1 dia	19MBytes	0,17MBytes
7 dias	133MBytes	1,19MBytes
30 dias	572MBytes	4,39MBytes

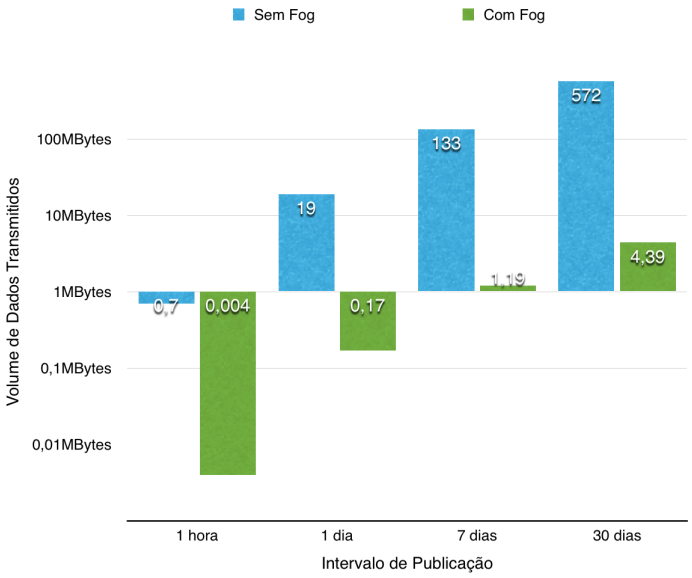


Figura 5.9 – Gráfico do volume de dados transferido ao Servidor de Contexto
Fonte: O Autor

Na Figura 5.9 são apresentados os resultados encontrados nesta avaliação, eles foram agrupados em quatro intervalos de tempo: hora, dia, semana e mês. Nesta figura é possível comparar o volume de dados trafegado considerando ou não o uso de FOG para os diferentes intervalos de tempo previstos.

A redução do volume de dados observada na Figura 5.9 é especialmente significativa quando se tem a comunicação com o Servidor de Contexto através de Internet móvel, com franquia limitada e/ou de baixa qualidade, comum em ambientes rurais como no cenário analisado.

A economia de recursos com o emprego de FOG aumenta conforme o período observado também cresce, isso dá-se devido as comunicações entre os Servidores de Borda e os controladores de atuação (sistema de irrigação) serem administradas pelos próprios servidores de borda, sendo assim praticamente não há necessidade de intervenção do Servidor de Contexto, aumentando a autonomia do ambiente de FOG em gerenciar-se a partir de parâmetros pré-estabelecidos. Essa característica também potencializa a escalabilidade a nível de borda, possibilitando que mais sensores possam ser conectados aos Servidores de Borda potencializando a complexidade das atividades que podem ser gerenciados pelos mesmos.

5.4.2 Leituras em Intervalos Reduzido

Em Viticultura de Precisão muitas vezes é necessário aumentar a frequência de obtenção das grandezas físicas do ambiente, ou seja, reduzir o intervalo de leitura dos sensores, e por consequência das potenciais atuações síncronas sob os vinhedos, seja porque há uma necessidade de manter um rigoroso controle sobre o ambiente, ou porque se deseja manter um registro das possíveis variações das informações contextuais entendidas como relevantes.

Porém intervalos pequenos para aquisição de informações contextuais podem constituir um problema operacional para arquiteturas que trabalham com um conceito de inteligência centralizada, uma vez que todos dados contextuais devem ser transmitidos para estes locais e, principalmente, os que necessitam de uma transmissão via Internet os quais podem sobrecarregar a rede existente ou limitar a banda disponível para outras transações de interesse do usuário, causando assim um dilúvio de dados ou congestionamento do canal de transmissão.

A Figura 5.10 demonstra a comparação ao se utilizar uma estratégia de FOG quando da necessidade de uma verificação mais frequente das variáveis contextuais. Neste teste foi observado o ambiente no período emulado de 30 dias, onde no ambiente com FOG são realizadas publicações para fins históricos a cada hora, enquanto no ambiente sem FOG as publicações são realizadas de acordo com a coleta dos dados.

É possível notar que o volume de dados enviado ao Servidor de Contexto é reduzido significavelmente no EXEHDA-FOG, sem modificar a operação do ambiente, ou seja, as atuações e processos são os mesmos no ambiente com FOG e sem FOG. O EXEHDA-FOG permite que

	Sem Fog	Com Fog
1 minuto	2.877MBytes	4,39MBytes
5 minutos	572MBytes	4,39MBytes
10 minutos	298MBytes	4,39MBytes

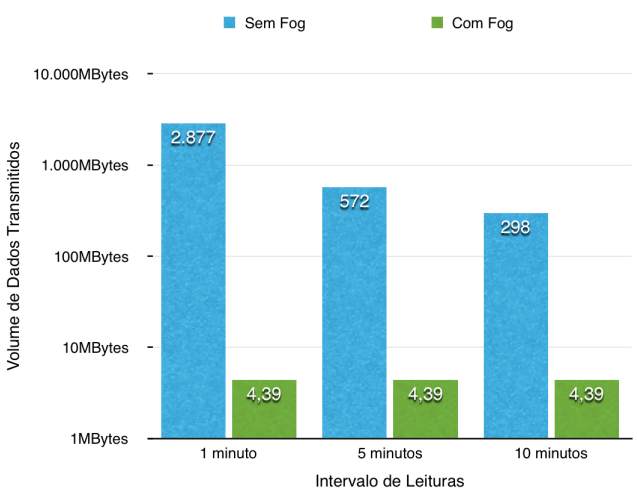


Figura 5.10 – Gráfico em função do intervalo de aquisição
Fonte: O Autor

o tempo de aquisição seja menor mantendo a dinamicidade do ambiente e das regras, porém sendo capaz em reduzir muito a latência das decisões originadas por aquisições síncronas, onde a requisição de leitura é feita através de um tempo pré-determinado.

5.4.3 Registro Histórico no Servidor de Contexto

A avaliação realizada considera as as 6 Zonas de Manejo com os dados coletados para um período de 30 dias, o objetivo desta avaliação é verificar o impacto no consumo de banda decorrente da alteração no período de registro histórico dos dados contextuais no Servidor de Contexto. Abaixo uma descrição das operações realizadas:

- o intervalo de de aquisição dos dados contextuais providos dos sensores não é o foco dessa avaliação;
- o intervalo de publicação da média da tensão de solo de cada Zona de Manejo foi avaliado para três valores de intervalo de publicação: 5 minutos, 30 minutos e 60 minutos;
- a última avaliação desconsidera a necessidade de manter-se um histórico das médias das tensões de solo das Zonas de Manejo, publicando para o Servidor de Contexto apenas os eventos ocorridos, ou seja, quando ocorreram, ou não, tensões de solo críticas altas e baixas e as irrigações.

Os resultados dessa avaliação são apresentados na Figura 5.11.

Publicação SC	Com Fog
5 minutos	21,978MBytes
30 minutos	5,94MBytes
60 minutos	4,39MBytes
Somente eventos	0,8MBytes

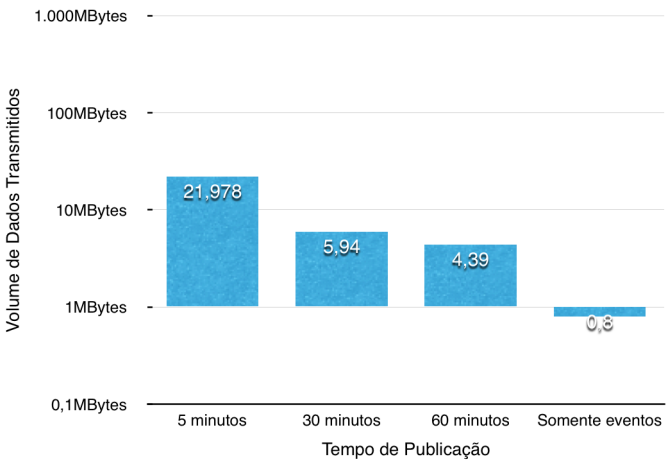


Figura 5.11 – Gráfico em função do intervalo de publicação no SC
Fonte: O Autor

Com as regras para atuação sendo processadas diretamente nos Servidores de Borda, o tempo de publicação no Servidor de Contexto não impacta nos procedimentos de controle de irrigação das Zonas de Manejo. Deste modo, o EXEHDA-FOG através do controle de quando os dados devem ou não serem publicados, permitindo poupar recursos de rede.

5.4.4 Escalabilidade dos Sensores Empregados

De forma a avaliar a escalabilidade nas bordas computacionais foram realizados testes na distribuição de sensores para os Gateways e, por associação por Servidor de Borda. O resultado do volume de dados transmitido alterando-se a quantidade de sensores pode ser visto na Figura 5.12.

Foi observado que a quantidade de sensores, para este ambiente, não apresentam mudanças significativas os dados enviados para o Servidor de Borda, isto deve-se ao foto do EXEHDA-FOG processar os dados a nível de borda e, utilizando de agregação de eventos de leitura com a média os dados são consolidados antes de serem propagados para o Servidor de Contexto, sendo possível assim a redução dos pacotes transmitidos.

Além disso, a estratégia de distribuição adotada no EXEHDA-FOG permite expandir o número de dispositivos conforme o aumento do volume de informações geradas, evitando assim possíveis sobrecargas nos dispositivos envolvidos. Isso significa que se lavoura fosse expandida e aumentasse a quantidade de Zonas de Manejo, mais Servidores de Bordas e Gateways seriam

Sensores por GW	Sem Fog	Com Fog
1 sensor	115MBytes	4,39MBytes
3 sensores	350MBytes	4,39MBytes
5 sensores	572MBytes	4,39MBytes

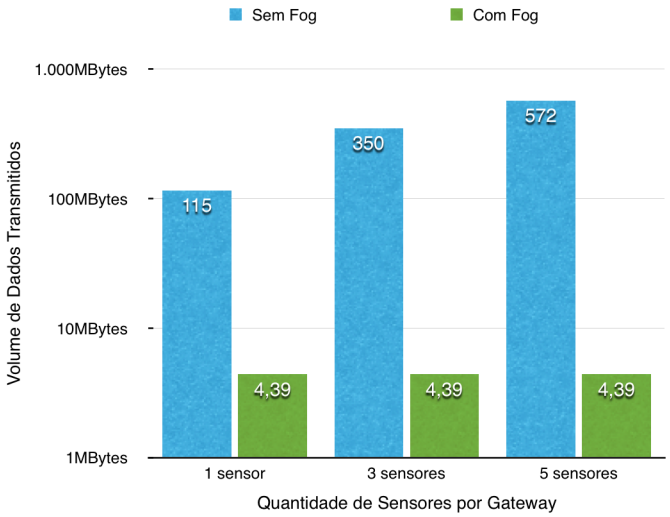


Figura 5.12 – Gráfico em função da quantidade de sensores no ambiente
Fonte: O Autor

utilizados, de modo que a quantidade de informações a ser tratada por cada dispositivo possa ser mantida sobre controle.

5.5 Considerações Sobre o Capítulo

Neste capítulo foi apresentado o estudo de caso desenvolvido para avaliação da arquitetura do EXEHDA-FOG. O cenário explora características típicas da IoT, como por exemplo: processamento contextual distribuído, conexão com Internet, atuação e gerência dinâmica de regras. O EXEHDA-FOG mostrou-se robusto e capaz de lidar com uma grande quantidade de dados contextuais de maneira distribuída, mantendo sua operação mesmo em desconexões com a internet e evidenciando a sua contribuição para Middlewares direcionados a IoT.

6 CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as principais conclusões relacionadas à concepção do EXEHDA-FOG, assim como as publicações realizadas e os trabalhos futuros.

6.1 Principais Conclusões

O cenário da IoT têm atraído interesse mundial, tanto academicamente quanto comercialmente. Do ponto de vista comercial, em grande parte, pela previsão de que em um futuro próximo bilhões de dispositivos inteligentes serão conectados em rede. Este crescente número de dispositivos que vem sendo conectados constituem uma fonte geradora de informações contextuais, o que potencializa o desenvolvimento de uma infinidade de aplicações nos mais diferentes domínios.

Por sua vez, no âmbito acadêmico verificou-se que o processamento de contexto na IoT é uma área de grande interesse, sendo perseguida por grupos renomados no cenário nacional e internacional, porém soluções efetivamente distribuídas para *middlewares* são ainda pouco exploradas e representam, segundo a literatura, uma importante característica para a descentralização da inteligência dos sistemas voltados para a IoT.

A revisão de literatura na área apontou que a estratégia de *Fog Computing*, baseada em distribuição de eventos, mostra-se promissora para promover a descentralização dos procedimentos de processamento contextual, expandindo as atuais soluções baseadas na *Cloud Computing*.

Considerando que a área de *Fog Computing* é uma área emergente no cenário internacional, se fez necessária uma busca criteriosa de trabalhos relacionados, tendo sido selecionados os mais relevantes na perspectiva da proposta buscada por esta dissertação de mestrado. Empregando como referência as características dos trabalhos relacionados, foram concebidos os mecanismos empregados na proposta arquitetural do EXEHDA-FOG, a qual se vale dos conceitos de *Fog Computing* na coleta e processamento de dados contextuais. Sua operação acontece de maneira distribuída, empregando as bordas computacionais, bem como considera o emprego de regras na momento da especificação para atender as demandas de processamento contextual das diferentes aplicações.

As avaliações feitas com o estudo de caso exploraram as funcionalidades em relação a gerência dinâmica e processamento contextual distribuído. Os testes realizados caracterizaram o EXEHDA-FOG como capaz de prover ciência de contexto empregando as bordas computa-

cionais de maneira distribuída, conseguindo manter sua operação mesmo com eventuais desconexões com a Internet. Uma característica que mostrou-se oportuna é a dinamicidade das regras de processamento contextual, e os eventos gerados por estas, provendo suporte a alteração ou modificação destas sem a necessidade de codificação pelo usuário, recompilação de código ou interrupção na operação gerenciada pelo EXEHDA-FOG.

6.2 Publicações Realizadas

Nesta seção encontram-se descritos os artigos publicados no andamento desta dissertação de mestrado. Os artigos contemplam os principais esforços alcançados até o momento, bem como sua projeção nas pesquisas realizadas junto ao grupo Grupo de Pesquisa em Processamento Paralelo e Distribuído (G3PD) da UCPel .

- DAVET, P.; DILLI, R.; XAVIER, L.; SOUZA, R.;**CARDOZO, A.**; WARKEN, N.; MACHADO, N.; PARREIRA, W.; YAMIN, A. C. Uma Abordagem Ciente de Contexto na IoT Aplicada à Pesquisa Agropecuária. In: **XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais**. Santarém-PA, Brasil: SBRT, 2016.

- **CARDOZO, A.**; YAMIN, A. C; DAVET, P.; SOUZA, R.; LOPES, J. L. B.; GEYER, C. Sensing And Actuation in IoT: an Autonomous Rule Based Approach. In: **IEEE International Workshop on Data Science for Internet of Things**. Brasília - BR: IEEE MASS DS-IoT, 2016.

- **CARDOZO, A.**; YAMIN, A. C; XAVIER, L.; SOUZA, R.; LOPES, J. L. B.; GEYER, C. An Architecture Proposal to Distributed Sensing in Internet of Things. In: **IEEE - International Symposium on Instrumentation Systems, Circuits and Transducers**. Belo Horizonte - BH: IEEE - INSCIT, 2016.

- SOUZA, R.; LOPES, J. L. B.; **CARDOZO, A.**; CARVALHO, T.; DAVET, P.; WOLF, A.; BARBOSA, J.; YAMIN, A. C; GEYER, C. Uma arquitetura para IoT direcionada à ciência do contexto baseada em eventos distribuídos. In: **8º SBCUP - Simpósio Brasileiro de Computação Ubíqua e Pervasiva**. Porto Alegre - RS: SBCUPT, 2016.

- **CARDOZO, A.**; YAMIN, A. C. Uma proposta baseada em Cloud of Things para o Middleware EXEHDA. In: **Salão Universitário UCPEL**. Pelotas - RS: UCPEL/RS, 2015.

- **CARDOZO, A.**; YAMIN, A. C. EXEHDA-FOG: Uma Abordagem de Fog Computing para o Subsistema de Reconhecimento de Contexto e Adaptação do Middleware EXEHDA. In: **Salão Universitário UCPEL**. Pelotas - RS: UCPEL/RS, 2016.

6.3 Submissões em Andamento

- SOUZA, R.; LOPES, J. L. B.;**CARDOZO, A.**; YAMIN, A. C.; GEYER A., An Architecture for IoT Management Targeted to Context Awareness of Ubiquitous Applications. In:**15th IFIP/IEEE International Symposium on Integrated Network Management**. Lisboa, Portugal , 2017.

6.4 Convites para Submissão em Revistas

- **CARDOZO, A.**; SOUZA, R.; LOPES, J. L. B.; CARVALHO, T.; DAVET, P.; WOLF, A.; BARBOSA, J.; YAMIN, A. C; GEYER, C. EXEHDA-FOG: An IoT Architecture directed to Context Awareness based on distributed events. In: **Journal of Applied Computing Research**. 2016.

- **CARDOZO, A.**; YAMIN, A. C; DAVET, P.; SOUZA, R.; LOPES, J. L. B.; GEYER, C.; CARVALHO, T. A Fog Computing Approach to Distributed Sensing and Actuation. In:**Advances in Science, Technology and Engineering Systems Journal - ASTESJ**. 2017.

Um registro dos diferentes trabalhos desenvolvidos ao longo do Mestrado esta disponivel em <http://olaria.ucpel.tche.br/acardozo/>.

6.5 Trabalhos Futuros

A arquitetura desenvolvida nesta dissertação de mestrado apresenta contribuições para o Subsistema de Reconhecimento de Contexto e Adaptação do Middelware EXEHDA em relação aos trabalhos relacionados no processamento de dados contextuais de maneira distribuída e na sua gerência. Durante o desenvolvimento do EXEHDA-FOG, e da escrita desta dissertação, foram identificadas possibilidades da continuidade deste trabalho, entre elas:

- **Expansão do suporte a Processadores de Contexto:** os serviços de processamento de contexto criados para o EXEHDA-FOG, Filtragem, Encadeamento e Agregação, podem serem combinados provendo diferentes alternativas de processamento contextual, porém é interessante que o Middleware seja capaz de empregar outros serviços de processamento de contexto.

- **Melhorias na interface gráfica de construção de regras CEP:** A interface gráfica presente no EXEHDA-FOG oferece suporte a construção de regras utilizando-se os serviços descritos na Seção 4.3, porem a construção de regras mais complexas de Encadeamento necessitam ser configuradas empregando diversas seções da interface.

- **Eventos com resultados customizados:** No EXEHDA-FOG os eventos produzidos através das regras de processamento contextual possuem resultados binários, ou seja, ou os eventos ocorreram ou não ocorream (True ou False). Uma customização do resultado de um evento pode propiciar interpretações mais ricas por parte das aplicações que empregam as informações produzidas pelo EXEHDA-FOG.

- **Acesso Localizado:** Para visualizar os dados coletados é necessário acessar a interface web localizada no Servidor de Contexto. O Acesso localizado é uma característica da *Fog Computing* que visa prover a capacidade de visualização de dados contextuais coletados as bordas computacionais, nesse caso ao Servidores de Borda. Dessa forma pode-se utilizar-se a rede local para visualização dos dados coletados quando necessário, mesmo que estes não sejam enviados para o Servidor de Contexto. Essa característica também permitiria a requisição de coleta para uma observação próxima a circunstancia do usuário.

A expectativa é que os esforços de pesquisa empregados no EXEHDA-FOG sejam continuados pelo grupo de pesquisa, expandindo a sua utilização em novas aplicações e cenários de uso.

REFERÊNCIAS

AAZAM, M.; HUH, E.-N. Fog computing and smart gateway based communication for cloud of things. In: IEEE. **Future Internet of Things and Cloud (FiCloud), 2014 International Conference on**. [S.l.], 2014. p. 464–470.

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

ASHTON, K. That 'internet of things' thing. **RFiD Journal**, v. 22, n. 7, p. 97–114, 2009.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer networks**, Elsevier, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. **International Journal of Ad Hoc and Ubiquitous Computing**, Inderscience Publishers, v. 2, n. 4, p. 263–277, 2007.

BELLAVISTA, P. et al. A survey of context data distribution for mobile ubiquitous systems. **ACM Computing Surveys (CSUR)**, ACM, v. 44, n. 4, p. 24, 2012.

BONOMI, F. Connected vehicles, the internet of things, and fog computing. In: **The Eighth ACM International Workshop on Vehicular Inter-Networking (VANET), Las Vegas, USA**. [S.l.: s.n.], 2011. p. 13–15.

BONOMI, F. et al. Fog computing: A platform for internet of things and analytics. In: **Big Data and Internet of Things: A Roadmap for Smart Environments**. [S.l.]: Springer, 2014. p. 169–186.

BONOMI, F. et al. Fog computing and its role in the internet of things. In: ACM. **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**. [S.l.], 2012. p. 13–16.

BOTTA, A. et al. Integration of cloud computing and internet of things: a survey. **Future Generation Computer Systems**, Elsevier, v. 56, p. 684–700, 2016.

BRAGA, R.; PINTO, P. Alterações climáticas e agricultura. **Inovação e Tecnologia na Formação Agrícola**, v. 12, n. 2, p. 34–56, 2009.

CARDOZO, A. et al. Sensing and actuation in IoT: an autonomous rule based approach. In: **The 1st International Workshop on Data Science for Internet of Things (DS-IoT 2016) (DS-IoT 2016)**. Brasilia, Brazil: [s.n.], 2016.

CARDOZO, A. et al. Sensing and actuation in iot: An autonomous rule based approach. In: IEEE. **Mobile Ad Hoc and Sensor Systems (MASS), 2016 IEEE 13th International Conference on**. [S.l.], 2016. p. 355–360.

CARDOZO, A. et al. An architecture proposal to distributed sensing in internet of things. In: IEEE. **Instrumentation Systems, Circuits and Transducers (INSCIT), International Symposium on**. [S.l.], 2016. p. 67–72.

CARRIOTS. **Carriots IoT Application Platform**. 2015. Disponível em :<https://www.carriots.com/>. Acesso em julho de 2015.

CONCEIÇÃO, M. A. F. C. **A irrigação na produção de uvas para elaboração de vinhos finos**. [S.l.]: Embrapa Uva e Vinho, 2008.

COSTA, C. A. da; YAMIN, A. C.; GEYER, C. F. R. Toward a General Software Infrastructure for Ubiquitous Computing. **IEEE Pervasive Computing**, v. 7, n. 1, p. 64–73, jan 2008. ISSN 1536-1268. Available from Internet: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4431859>.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. **ACM Computing Surveys (CSUR)**, ACM, v. 44, n. 3, p. 15, 2012.

DAVET, P. et al. Uma abordagem ciente de contexto na IoT aplicada à pesquisa agropecuária. In: **XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2016) (SBrT 2016)**. Santarém, Brazil: [s.n.], 2016.

DELICATO, F. C.; PIRES, P. F.; BATISTA, T. **Middleware Solutions for the Internet of Things**. [s.n.], 2013. 57–73 p. ISBN 978-1-4471-5480-8. Available from Internet: <http://link.springer.com/10.1007/978-1-4471-5481-5>.

DUTT, V.; GONZALEZ, C. Cyber situation awareness through instance-based learning: Modeling the security analyst. **Situational Awareness in Computer Network Defense: Principles, Methods and Applications: Principles, Methods and Applications**, IGI Global, p. 125, 2012.

ENDSLEY, M. R. Toward a Theory of Situation Awareness in Dynamic Systems. **Human Factors: The Journal of the Human Factors and Ergonomics Society**, v. 37, n. 1, p. 32–64, mar 1995. ISSN 00187208. Available from Internet: <http://openurl.ingenta.com/content/xref?genre=article{&}issn=0018-7208{&}volume=37{&}issue=1>.

ETZION, O.; NIBLETT, P. **Event processing in action**. [S.l.]: Manning Publications Co., 2010.

FLEISCHMANN, A. M. P. **Sensibilidade à situação em sistemas educacionais na web**. 164 p. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2012. Available from Internet: <http://www.lume.ufrgs.br/bitstream/handle/10183/56845/000860091.pdf?sequence=1>.

FONSECA, J.; FERRAZ, C.; GAMA, K. A policy-based coordination architecture for distributed complex event processing in the internet of things: doctoral symposium. In: ACM. **Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems**. [S.l.], 2016. p. 418–421.

FREMANTLE, P. A reference architecture for the internet of things. **WSO2 White Paper**, 2015.

GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. **Future Generation Computer Systems**, Elsevier, v. 29, n. 7, p. 1645–1660, 2013.

GUBBI, J. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. **Future Generation Computer Systems**, Elsevier B.V., v. 29, n. 7, p. 1645–1660, sep 2013. ISSN 0167739X. Available from Internet: <http://dx.doi.org/10.1016/j.future.2013.01.010http://linkinghub.elsevier.com/retrieve/pii/S0167739X13000241>.

HASHEM, I. A. T. et al. The rise of “big data” on cloud computing: Review and open research issues. **Information Systems**, Elsevier, v. 47, p. 98–115, 2015.

HELMER, S.; POULOVASSILIS, A.; XHAFA, F. **Reasoning in event-based distributed systems**. [S.l.]: Springer, 2011.

HONG, K. et al. Mobile fog: A programming model for large-scale applications on the internet of things. In: ACM. **Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing**. [S.l.], 2013. p. 15–20.

JANKOWSKI, S. et al. The internet of things: Making sense of the next mega-trend. **Goldman Sachs**, 2014.

KNAPPMEYER, M. et al. Survey of Context Provisioning Middleware. **IEEE Communications Surveys & Tutorials**, v. 15, n. 3, p. 1492–1519, jan 2013. ISSN 1553-877X.

KURNIAWAN, A. **Sparkfun ESP8266 Thing Development Workshop**. Canada: PE PRESS, 2016. An optional note.

LAN, L. et al. An event-driven service-oriented architecture for the internet of things service execution. **International Journal of Online Engineering**, v. 11, n. 2, 2015.

LANGTON, L. **Seed Testing International**. 2015. <http://www.iiclouds.org/20150115/towards-a-new-paradigm-fog-computing/. Acesso em Março de 2016. Available from Internet: <http://www.iiclouds.org/20150115/towards-a-new-paradigm-fog-computing/>.

LIGHT, R. Mosquitto-an open source mqtt v3. 1 broker. **URL: http://mosquitto. org**, 2013.

LINKSMART. **Middleware for Networked Embedded Systems**. 2015. Disponível em :<https://linksmart.eu/redmine/projects/linksmart-opensource/wiki>. Acesso em julho de 2015.

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

LUCKHAM, D. **The power of events: An introduction to complex event processing in distributed enterprise systems**. [S.l.]: Springer, 2008.

MACEDO, A. Q.; MARINHO, L. B.; SANTOS, R. L. Context-aware event recommendation in event-based social networks. In: ACM. **Proceedings of the 9th ACM Conference on Recommender Systems**. [S.l.], 2015. p. 123–130.

MIORANDI, D. et al. Internet of things: Vision, applications and research challenges. **Ad Hoc Networks**, Elsevier B.V., v. 10, n. 7, p. 1497–1516, sep 2012. ISSN 15708705. Available from Internet: <http://dx.doi.org/10.1016/j.adhoc.2012.02.016http://linkinghub.elsevier.com/retrieve/pii/S1570870512000674>.

MLADENOW, A.; KRYVINSKA, N.; STRAUSS, C. Towards cloud-centric service environments. **Journal of Service Science Research**, Springer, v. 4, n. 2, p. 213–234, 2012.

PARWEKAR, P. From internet of things towards cloud of things. In: IEEE. **Computer and Communication Technology (ICCCT), 2011 2nd International Conference on**. [S.l.], 2011. p. 329–333.

PERERA, C. et al. Context aware computing for the internet of things: A survey. **Communications Surveys & Tutorials, IEEE**, IEEE, USA, v. 16, n. 1, p. 414–454, 2013.

PERERA, C. et al. Context aware computing for the internet of things: A survey. **IEEE Communications Surveys and Tutorials**, v. 16, n. 1, p. 414–454, jan 2014. ISSN 1553877X.

PERERA, C. et al. Context aware computing for the internet of things: A survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 1, p. 414–454, 2014.

PINHO, T.; BOAVENTURA-CUNHA, J.; MORAIS, R. Tecnologias da eletrónica e da computação na recolha e integração de dados em agricultura de precisão. **Revista de Ciências Agrárias**, Sociedade de Ciências Agrárias de Portugal, v. 38, n. 3, p. 291–304, 2015.

PIRES, P. F. et al. Plataformas para a Internet das Coisas. In: **Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Vitória: Sociedade Brasileira de Computação (SBC), 2015. chp. 3.

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

PRIVAT, G. et al. Edge-of-cloud fast-data consolidation for the internet of things. **Proceedings of the 9th ACM Conference on Recommender Systems**, 2016.

RASPBERRY. **Raspberry Pi**. 2016. <http://www.raspberrypi.org>. Acessado em março de 2016. Available from Internet: <http://www.raspberrypi.org>.

SAVITZ, E. Gartner: Top 10 strategic technology trends for 2013. **CIO Network, Forbes**, 2012.

SOUZA, R. et al. Uma arquitetura para iot direcionada à ciência do contexto baseada em eventos distribuídos. In: **CSBC 2016 - SBCUP**. [s.n.], 2016. Available from Internet: <http://ebooks.pucrs.br/edipucrs/anais/csbc/assets/2016/anais-csbc-2016.pdf>.

SRI Consulting Business Intelligence. **Disruptive civil technologies: Six technologies with potential impacts on US interests out to 2025**. USA, 2008.

STOJMENOVIC, I.; WEN, S. The fog computing paradigm: Scenarios and security issues. In: IEEE. **Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on**. [S.l.], 2014. p. 1–8.

TORRES, A. B.; ROCHA, A. R.; SOUZA, J. N. de. Análise de desempenho de brokers mqtt em sistema de baixo custo. **15º WPerformance - Workshop em Desempenho de Sistemas Computacionais e de Comunicação**, 2016.

VERMESAN, O.; FRIESS, P. **Internet of Things-From research and innovation to Market Deployment**. [S.l.]: River Publishers, 2014.

VERMESAN, O.; FRIESS, P. **Building the Hyperconnected Society: IoT Research and Innovation Value Chains, Ecosystems and Markets", vol.** [S.l.]: River Publishers, 2015.

WANT, R. et al. **Ubiquitous Computing Fundamentals**. [S.l.]: Chapman & Hall/CRC, 2010. ISBN 9781420093605.

WANT, R.; SCHILIT, B. N.; JENSON, S. Enabling the internet of things. **IEEE Computer**, Citeseer, v. 48, n. 1, p. 28–35, 2015.

Xiao Hang Wang et al. Ontology based context modeling and reasoning using OWL. In: **Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on**. IEEE, 2004. p. 18–22. ISBN 0-7695-2106-1. Available from Internet: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1276898>.

XIVELY. **Xively**. 2016. <http://www.xively.com>. Acessado em janeiro de 2016. Available from Internet: <http://www.xively.com>.

ZASLAVSKY, A.; PERERA, C.; GEORGAKOPOULOS, D. Sensing as a service and big data. **arXiv preprint arXiv:1301.0159**, 2013.

ANEXO A — HARDWARES DE BORDA EMPREGADOS NO EXEHDA-FOG

Nesse anexo estão descritos os hardwares utilizados pelo G3PD para modelagem de Servidores de Borda e de Gateways, para uso em aplicações desenvolvidas pelo grupo. Sua descrição tem por objetivo caracterizar a infraestrutura emulada quando do Estudo de Caso (Capítulo 5). Estes hardwares estão sendo empregados em aplicações já em produção tanto na Embrapa Clima Temperado, como na Universidade Federal de Pelotas. Estas aplicações são pertinentes a etapa inicial de modelagem do EXEHDA-FOG, e estão descritas nas publicações (CARDOZO et al., 2016b) e (CARDOZO et al., 2016c)

A.0.1 Servidor de Borda

O Servidor de Borda utilizado pelo G3PD é microcomputador Raspberry PI Modelo B+ (vide Figura A.1a) sobre o Sistema Operacional Raspbian. A Raspberry Pi (RASPBERRY, 2016) é um computador desenvolvido pelo Laboratório de Computação da Universidade de Cambridge que tem como principais características o tamanho reduzido, baixo custo, baixo consumo energético, com comunidade de usuários bastante ativa e que baseia-se nos princípios de software livre. A utilização da Raspberry Pi como Servidor de Borda se interessante, pois esta adequa-se perfeitamente ao cenário da IoT, possuindo uma boa capacidade de processamento, possibilitando o processamento de regras complexas e capacidade de armazenamento local de forma a manter o estado dos dados por um período suficiente para serem analisados.

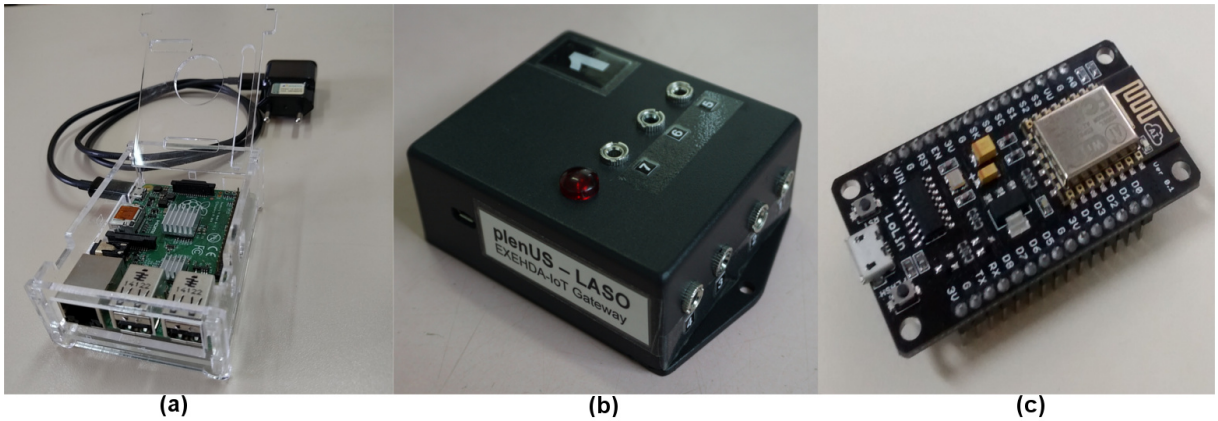
O Raspberry Pi também apresenta boa conectividade com a Internet através da WI-FI, podendo assim comunicar-se com os outros dispositivos da arquitetura sem fios com o Gateway, por exemplo.

A.0.2 Gateways

Os Gateways são dispositivos mais simples e possuem um poder computacional reduzido, sendo utilizados principalmente como forma de abstração dos protocolos de comunicação entre os sensores e os servidores de borda. Desta forma, visando um baixo consumo energético e o emprego de uma plataforma de software bem disseminada foi utilizado o processador ESP8266-12 (KURNIAWAN, 2016) (vide Figura A.1c) para o desenvolvimento do Gateway Nativo (vide Figura A.1b), um dispositivo com suporte a comunicação WI-FI e que possiblita a

programação através do software Arduino[1].

Para realizar as demandas de sensoriamento é utilizada a tecnologia 1-Wire como protocolo físico de acesso aos sensores. Esta tecnologia tem como característica a transmissão de dados em rede através de dispositivos eletrônicos endereçáveis, apresentando robustez e não exige um cabeamento especializado para a sua operação.



(a) (b) (c)

Figura A.1 – Dispositivos Utilizados: (a) Raspberry PI;
(b) Gateway Nativo (GW-SB) ; (c) ESP8266-12

[1]https://www.arduino.cc/en/Reference/

ANEXO B — APLICAÇÃO DE GERENCIAMENTO DO EXEHDA-FOG

De forma a auxiliar o gerenciamento das funcionalidades implementadas no EXEHDA-FOG foi desenvolvida uma aplicação web disponibilizada no Servidor de Contexto, algumas das suas interfaces são apresentadas a seguir.

B.0.1 Funcionalidade de Agendamento de Aquisição de Dados Contextuais

No EXEHDA-FOG é possível gerenciar a frequência com que a aquisição dos dados contextuais acontece. A Figura B.1 exibe um agendamento para leitura de uma grandeza de tensão de solo a cada minuto. Os agendamentos, assim como as regras de processamento contextual, são dinâmicos e podem ser alterados conforme a necessidade sem interromper a operação do EXEHDA-FOG. Após realizada a configuração das regras na interface estas são enviadas em formato JSON para os Servidores de Borda que operam os sensores selecionados.

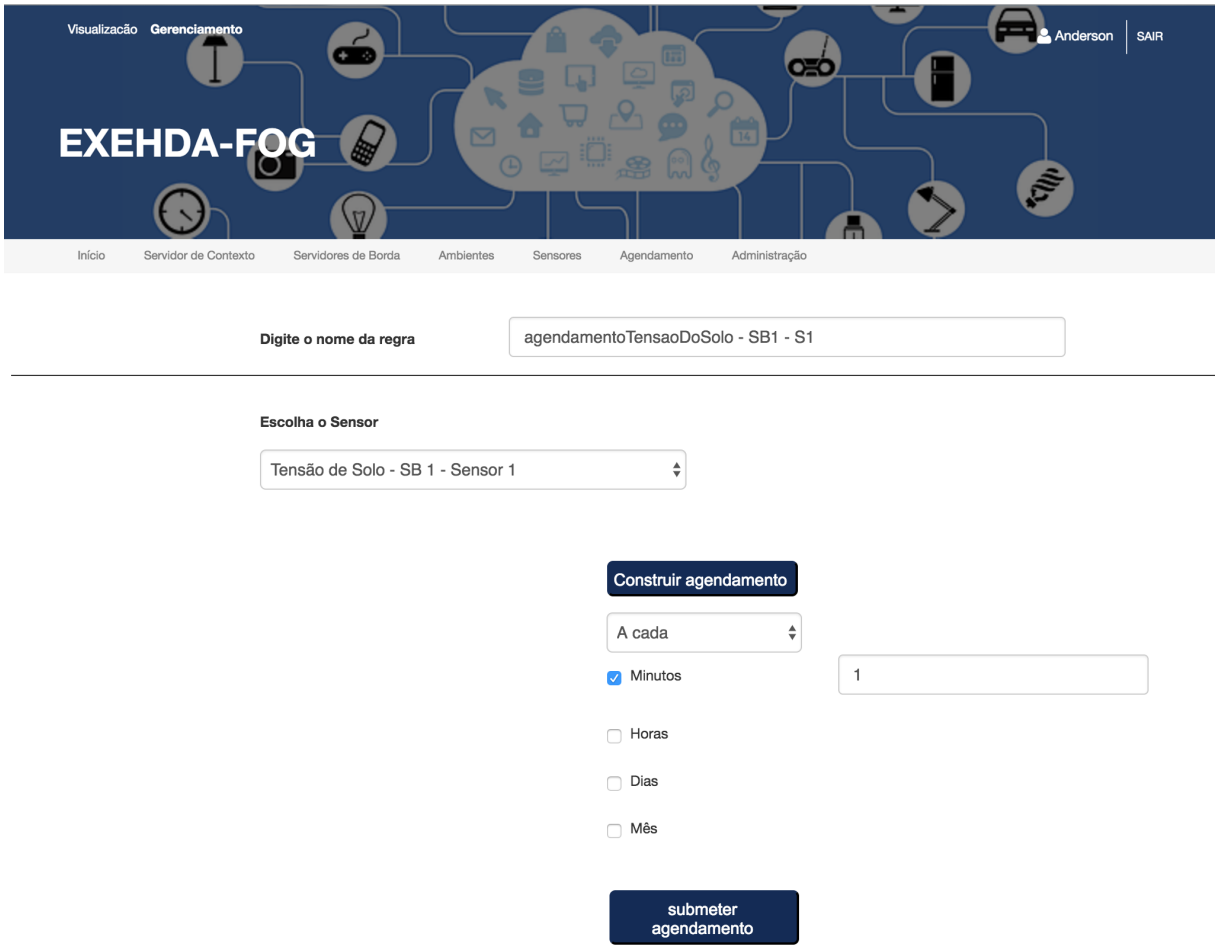


Figura B.1 – Interface de agendamento de aquisição contextual

O EXEHDA-FOG disponibiliza 4 opções, com a finalidade de possibilitar algumas configurações de agendamentos para o usuário, apresentadas a seguir:

- **A cada**: Neste tipo de agendamento, o usuário pode escolher um período, que pode ser estipulado por horas e/ou minutos, para execução de um evento no servidor de Borda;
- **Exatamente**: Neste agendamento o usuário seleciona uma data e hora exata para uma coleta ocorra. Todos campos não preenchidos são ignorados pela aplicação, a menos que nenhum campo seja preenchido;
- **Todos os dias**: Neste tipo de agendamento, o usuário pode escolher um horário fixo de coleta que ocorrerá todos os dias;
- **Todos os meses**: Tem um funcionamento semelhante ao tipo anterior, no entanto a escolha do usuário esta relacionada ao dia e hora de coleta. Por exemplo, neste tipo de agendamento pode-se determinar que a verificação de uma dado contextual dar-se-á em um determinado dia, hora e minuto do mês.

B.0.2 Funcionalidade de Agrupamento de Regras

A possibilidade de combinar regras de processamento contextual ou agendamento em grupos possibilita que diversos aspectos de monitoramento e processamento possam ser controlados autonomamente pelo EXEHDA-FOG. Na Figura B.2 pode ser visualizado um agrupamento de regras de processamento contextual denominado Janela Dia. Uma vez criada uma regra do tipo **Grupo de Regras** é possível que estas sejam ativadas ou desativas através de ações de outras regras de processamento contextual ou por elas mesmas.

A funcionalidade de combinar regras de processamento contextual e agendamento em grupos provida pelo EXEHDA-FOG possibilita que o ambiente reaja a situações críticas ativando regras específicas para quando estas ocorrerem ou então preservando recursos em situações em que o ambiente encontra-se estável.

Uma regra do tipo **Grupo de Regras** também gera eventos, sendo possível que outras regras subscrevam-se na informação de a "Janela Dia", da Figura B.2, foi ativada ou desativada, possibilitando que ações sejam tomadas a partir dessa informação.

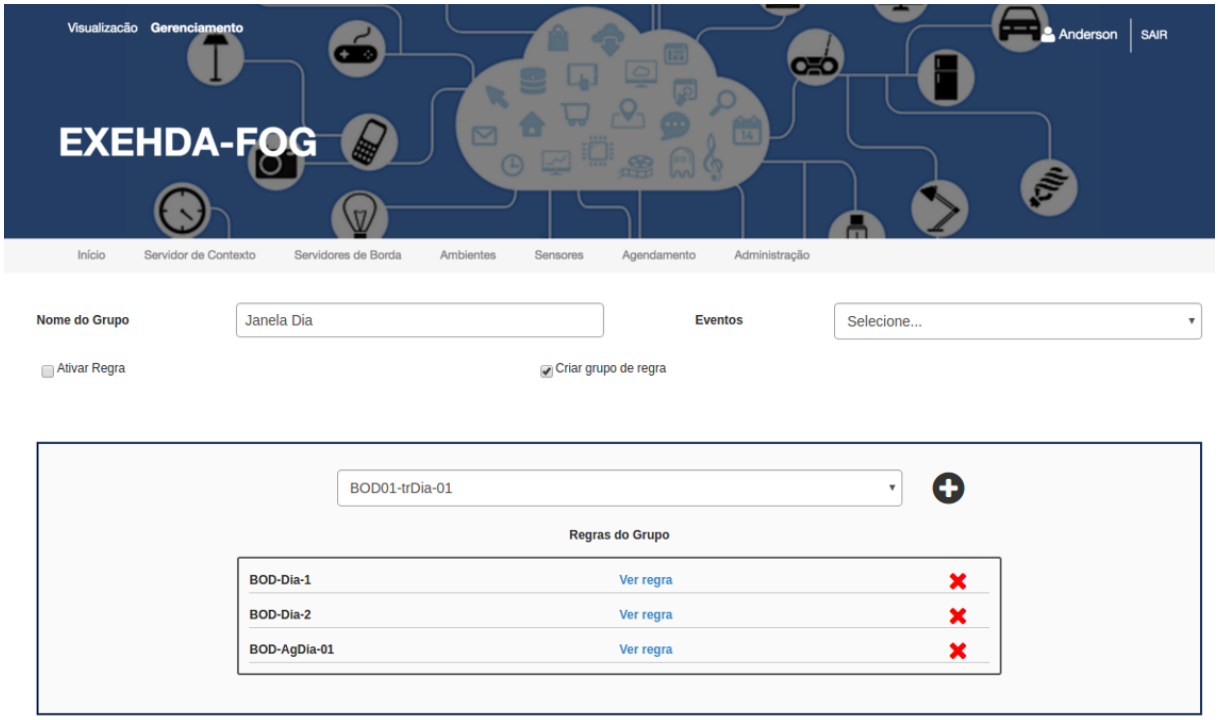


Figura B.2 – Interface com exemplo de grupo de regras