UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MECANISMO DE TRATAMENTO DE EXCEÇÕES SENSÍVEL AO CONTEXTO APLICADO A INTERNET DAS COISAS

Gabriel Henrique Faustini Gomes

Orientador(a): Joice Lee Otsuka, PhD

São Carlos – SP

Setembro/2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MECANISMO DE TRATAMENTO DE EXCEÇÕES SENSÍVEL AO CONTEXTO APLICADO A INTERNET DAS COISAS

Gabriel Henrique Faustini Gomes

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software
Orientador(a): Joice Lee Otsuka, PhD

São Carlos – SP

Setembro/2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato Gabriel Henrique Faustini Gomes, realizada em 08/10/2020.

Comissão Julgadora:

Prof. Dr. Delano Medeiros Beder (UFSCar)

Prof. Dr. Auri Marcelo Rizzo Vincenzi (UFSCar)

Prof. Dr. Jó Ueyama (ICMC/USP)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Agradecimentos

Dedico essa dissertação a todas pessoas que me ajudaram nessa jornada desde os meus 14 anos de idade quando decidi trabalhar com computação. Coloco aqui a minha gratidão a todos os professores que colocaram cada peça desse quebra cabeça constituindo a minha base de conhecimento, isso inclui meus professores do SENAC Ribeirão Preto em especial o Gilberto Caldeira e o Renato Rodrigues, meus professores do Instituto Federal de Educação Ciência e Tecnologia (IFSP) São Carlos em especial os professores Lucas Bueno, Pablo Dalbem e Mariana Feres. Coloco aqui também os meus agradecimentos aos colaboradores da Universidade Federal de São Carlos (UFSCar) em especial os professores Delano Beder e Estevam Hruschka, além da TA Rosangela Florian.

Agradeço a todas as pessoas que estiveram ao meu lado durante o meu processo de recuperação do acidente que tive no ano de 2019. Fica aqui minha eterna gratidão aos médicos e enfermeiros do HC Ribeirão Preto, que com seu trabalho de excelente qualidade e com as mãos de Deus, fizeram um trabalho excelente para que eu pudesse hoje estar finalizando este trabalho de pesquisa. Obrigado a todos os colaboradores da empresa Hexagon pela contribuição em meus conhecimentos e suporte durante o período que estive no hospital. Agradeço também a contribuição de meus amigos Luciano Bis, William Texeira, Jasiel Macagnan, Carlo Fiore e Alexander Maksimow pela amizade e parceria de muitos anos, me dando suporte nos momentos felizes e também nos difíceis. Escrevo aqui o meu muito obrigado ao meu pai Lúcio, e a minha mãe Miriam, por todo o suporte de uma vida e todo conhecimento passado que hoje constitui os meus princípios e a pessoa que me tornei. Por fim agradeço a Deus, pelos milagres que aconteceram em minha vida, que só confirmam o fato de não cair uma folha sem que seja da vontade dele.

Lista de Abreviaturas e Siglas

ACM – *Association for Computing Machinery*

BPM – *Business Process Management*

CAEH – *Context-Aware Exception Handling*

CIS – *Context Information Service*

ECA – *Evento-Condição-Ação*

EHC – *Exception Handling Component*

IoT – *Internet of Things*

LIS – *Location Inference Service*

MQTT – *Message Queue Telemetry Transport*

OMA – *Open Mobile Alliance*

SA – *Situação-Ação*

SRM – *Symbolic Region Manager*

TBDM – *Task-Based Devolopment Model*

WSN – *Wirelesss Sensor Networks*

Resumo

Dispositivos da *Internet of Things* (IoT), que podem variar de relógios inteligentes a monitores de saúde, estão cada vez mais intrínsecos e ubíquamente conectados ao cotidiano das pessoas. Com esses dispositivos e aplicativos onipresentes crescendo em números, e a nossa sociedade cada vez mais confiando em tais sistemas, diversos desafios de desenvolvimento de software surgiram. Aplicações mais sofisticadas devem ser capazes de controlar e explorar a dinâmica da mudança de informações contextuais dos usuários e do ambiente circundante. Assim, tais sistemas precisam atender aos requisitos rigorosos relacionados à tolerância a falhas, integridade e disponibilidade. Esta questão fundamental no desenvolvimento de sistemas onipresentes confiáveis está na maneira correta de lidar com erros na presença de frequentes mudanças contextuais e comunicação assíncrona. Neste contexto, com o objetivo de superar esses desafios inerentes, esta dissertação propõe um mecanismo de tratamento de exceções sensível ao contexto aplicado à Internet das Coisas. O mecanismo é composto por um middleware publish/subscribe para comunicar os dispositivos em conjunto com uma arquitetura proposta para realizar o tratamento de exceções contextuais. A validação do mecanismo de tratamento de exceção proposto foi realizada por meio do desenvolvimento da simulação de uma aplicação prática (estudo de caso) que explora e avalia seu uso na construção de sistemas assíncronos sensíveis ao contexto. Todo o código do mecanismo bem como a simulação foi disponibilizado em uma plataforma de forma que desenvolvedores possam utilizar das diretrizes criadas para implementar o tratamento de exceções.

Palavras-chave: Dispositivos móveis, internet das coisas, tratamento de exceções, sensibilidade ao contexto

Abstract

Internet of Things (IoT) devices, which can range from smart watches to health monitors, are increasingly intrinsic and ubiquitously connected in people's routine. With such ubiquitous devices and applications growing in numbers, and our society increasingly relying on such systems ubiquitous, a number of software development challenges have emerged. More sophisticated applications should be able to control and exploit the dynamics of changing contextual information of users and the surrounding environment. Thus, such systems need to meet stringent requirements related to application-specific fault tolerance, integrity, and availability. This fundamental issue in the development of reliable ubiquitous systems lies in how properly handle application errors in the presence of frequent contextual changes and asynchronous communication. In this context, aiming to overcome these inerent challenges, this research proposes a Context-aware Exception Handling (CAEH) Mechanism Applied to Internet of Things. The mechanism consists of publish/subscribe middleware to communicate the devices in conjunction with a proposed architecture to handle contextual exceptions. The validation of the proposed CAEH mechanism was accomplished through the development of a the simulation of a practical application (case study) that explore and evaluate its use in the construction of context-sensitive asynchronous systems. All source code as the simulation was made available on a platform so that developers can use the guidelines created to implement exception handling.

Keywords: Mobile devices, internet of things, exception handling, context-aware

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Sumário

Capítulo 1

Introdução

Esta dissertação tem como objetivo apresentar um mecanismo de tratamento de exceções sensível ao contexto para dispositivos da internet das coisas. A Seção 1.1 apresenta uma visão geral sobre internet das coisas e sua relação com computação ubíqua. Em seguida, na Seção 1.2 é apresentada a motivação que levou a criação do mecanismo assim como as vantagens de sua utilização. Após isso, é apresentada a hipótese na Seção 1.3. Já a Seção 1.4 mostra os objetivos do trabalho e a sua contribuição. Por fim, na Seção 1.5 temos a organização do documento.

1.1 Contextualização

O conceito de computação ubíqua foi cunhado pela primeira vez por Weiser (1991), e segundo ele, as tecnologias mais profundas são aquelas que se misturam com os elementos do cotidiano até ficarem invisíveis. Nesse contexto, os objetos (e.g. celulares, relógios, geladeiras, cafeteiras) integram o cotidiano das pessoas e passam a carregar poder computacional para auxiliar em suas atividades. Esses objetos no século XXI integram todos os ambientes desde lugares públicos a lugares privados, são os chamados ambientes ubíquos, em que se inicia uma nova era em que a comunicação ubíqua e a conectividade não são mais um sonho ou desafio, e sim realidade. Após a concretização da computação ubíqua, o foco foi mudado um pouco para a integração de pessoas e dispositivos que convergem o reino físico em ambientes virtuais produzidos pelos humanos, criando assim o famoso termo Internet das Coisas (KRISHNAMURTHY; MAHESWARAN, 2016).

Para que a aplicação inserida nos dispositivos IoT consiga realizar seu proposito, é

necessário que ela tenha o conhecimento do contexto em que está inserida (NUGROHO, 2016). Segundo Dey e Abowd (1999), o contexto pode ser um conjunto de informações relevantes sobre o ambiente físico ou lógico, que podem ser os próprios usuários e até mesmo o sistema. Também, segundo Dey e Abowd (1999) o acesso dos computadores ao contexto melhora a comunicação humano-computador e como consequência as aplicações conseguem se adequar melhor às necessidades dos usuários. Entretanto, o desenvolvimento de aplicações sensíveis ao contexto não é uma tarefa fácil, e manter a aplicação funcionando de forma correta em diferentes contextos é um processo complicado.

Os sistemas sensíveis ao contexto por terem natureza proativa com mínima, ou nenhuma intervenção de usuário, tem que ser confiáveis (FILHO et al., 2014). A confiabilidade de software pode ser dividida em dois aspectos gerais: corretude e robustez, sendo que a corretude é o quanto o sistema atende as suas especificações, já a robustez é a habilidade do software de reagir de forma correta a situações anormais (MEYER, 1988).

Um sistema tolerante a falhas é aquele projetado para funcionar confiavelmente apesar da ocorrência de falhas parciais durante a sua execução. De acordo com a terminologia definida por Lee e Anderson (1990), as respostas dos componentes de um sistema podem ser classificadas em duas categorias: normal e anormal. As exceções são definidas como sendo as respostas anormais ou excepcionais de um componente. Desta forma, exceções proveem uma estrutura adequada para modelar a implementação de mecanismos de tolerância a falhas. Esta estrutura permite uma separação clara das atividades executadas em um estado errôneo (mecanismos de recuperação de erros) das atividades funcionais do sistema. Dessa maneira, um sistema tolerante a falhas é capaz de preservar as suas funcionalidades, sem apresentar defeitos mesmo na presença de falhas. Quando o sistema se encontra em um estado errôneo, um componente deve gerar as exceções que modelem o erro e ser capaz de tratar essas exceções que foram levantadas. Assim, como outros tipos de sistemas, os sistemas ubíquos podem apresentar falhas. Na literatura foi constatado a utilização de diferentes middlewares e frameworks para solucionar os diversos problemas relacionados ao desenvolvimento para dispositivos ubíquos (LOPES, 2008; BAZZANI et al., 2012; BATISTA, 2016), mas eles não possuem as abstrações necessárias para realizar o tratamento de exceções de forma adequada.

A pesquisa realizada efetivou um estudo para verificar qual seria o middleware mais adequado baseado em suas características que em conjunto com novas abstrações, pudesse realizar o tratamento de exceções contextuais de forma adequada. Tais abstrações que terão como responsabilidade lidar de dispositivos, entidades, atributos e elementos de

contexto. Essas abstrações serão implementadas com a utilização de uma linguagem orientada a objetos, que dê suporte a chamada introspecção de objetos, onde pode-se visualizar os atributos e métodos de um objeto baseado em um texto, em tempo de execução. Para a implementação de cada uma das abstrações foram criadas classes que representam componentes na arquitetura, cada uma com sua própria responsabilidade dentro do mecanismo. O código do mecanismo foi disponibilizado em uma plataforma e tem sua *url* exibida no Capítulo 4 em que são exibidos os trechos de código. Também nesse Capítulo são disponibilizadas as diretrizes, de forma que qualquer desenvolvedor possa entender como implementar o mecanismo. A pesquisa foca no tratamento de exceções de forma sequencial onde não serão tratadas exceções concorrentes. Exceção concorrente é caracterizada quando duas exceções são levantadas que poderiam compor outra exceção formada pelas duas anteriores e deve ser tratada de forma específica.

1.2 Motivação

Um dos pilares de construção da IoT, é o processamento de eventos sensível ao contexto (KRISHNAMURTHY; MAHESWARAN, 2016). Esse processamento de eventos é necessário para que se possa verificar os contextos excepcionais. Segundo Damasceno et al. (2006) os contextos excepcionais significam uma ou mais condições associadas com os tipos de contexto, as quais juntas indicam o ambiente, hardware ou falha de software, ou seja, o contexto excepcional caracteriza alguma situação excepcional de uma entidade no qual "excepcional" também varia de acordo com a aplicação. Este trabalho vem para preencher a lacuna na literatura da impossibilidade de lidar com esses contextos excepcionais de forma sistemática, explícita e guiada. Para isso é proposto um mecanismo de forma a facilitar o desenvolvimento do software de tratamento de exceções sensível ao contexto através do levantamento bibliográfico acerca do estado da arte para compor o mecanismo. A dinamicidade e dificuldades de se lidar com esse tipo de ambiente podem ser exemplificados por Damasceno et al. (2006), em que um exemplo de contexto excepcional é caracterizado pela temperatura de um local como escritório ou sala pública que ocasionalmente excede um limite máximo de acordo com as preferências do usuário. Isso pode causar problemas sérios ao sistema de aquecimento no caso dessa aplicação em específico, e para lidar com esse tipo de situação, é necessário que o sistema realize possivelmente atividades de tratamento de exceções (fluxo de controle excepcional) visando o restabelecimento do sistema de fluxo normal de funcionamento. Sendo qualquer um desses fluxos, os dados de contexto serão gerados concorrentemente e podem ser expressos através de

eventos. Estes dados podem ser originários de diversos dispositivos e para lidar com grandes quantidades é necessário que haja um gerenciamento, dessa forma, "mecanismos de comunicação baseado em eventos que proveem suporte para composição de eventos concorrentes permitem uma maior expressividade na declaração de interesses" (LOPES, 2008). No caso do exemplo de Damasceno et al. (2006), os eventos servem para transmitir as informações do contexto em que o dispositivo se encontra e consequentemente o mecanismo irá receber a informação, lidar de forma específica da propagação de exceções e da mudança do fluxo normal para o fluxo excepcional quando a exceção for levantada (DAMASCENO et al., 2006). Além disso, o mecanismo também deve ser responsável por suportar diferentes estratégias de fluxos excepcionais e buscar os tratadores apropriados para cada tipo de exceção.

1.3 Hipótese

A hipótese a ser investigada nesse trabalho de pesquisa é: A construção de um mecanismo de tratamento de exceções sensível ao contexto voltado para dispositivos *IoT* para utilização por engenheiros de software de forma sistemática.

1.4 Objetivos

O objetivo principal desse trabalho é desenvolver um mecanismo que seja capaz de facilitar a tarefa de tratamento de exceções sensível ao contexto para dispositivos *IoT*. Para que esse objetivo seja completado, alguns objetivos tem que ser definidos e alcançados, dentre eles:

- Revisão do estado da arte sobre o tratamento de exceções sensível ao contexto;
- O desenvolvimento das abstrações presentes nos componentes do mecanismo, abstrações estas que serão capazes de fazer a resolução de exceções, propagar exceções, tratar exceções e assim por diante.
- Desenvolver um protótipo do modelo de tratamento de exceções proposto utilizando um middleware que respeita o paradigma *publish-subscribe*.
- Propor diretrizes com os passos necessários para a utilização do mecanismo, tais diretrizes fazem uso do modelo proposto e auxiliam a interação dos desenvolvedores com o mecanismo.

- Validar o mecanismo e as diretrizes através de um estudo de caso.

1.5 Organização do documento

A organização desta dissertação é descrita a seguir. O Capítulo 2 aborda os conceitos básicos necessários para compreender este trabalho. Já o Capítulo 3 apresenta o levantamento bibliográfico do estado da arte com relação a sistemas sensíveis ao contexto. O Capítulo 4 mostra os conceitos principais do mecanismo como a motivação, objetivos, arquitetura e implementação. Em adição, esse capítulo também apresentada as diretrizes para utilização do mecanismo. O Capítulo 5 exibe a validação do mecanismo e das diretrizes através de um estudo de caso. Por fim o Capítulo 6 apresenta as conclusões e considerações finais sobre este trabalho de pesquisa.

Capítulo 2

Conceitos Básicos

*Primeiramente na Seção 2.1 será explicado o conceito de internet das coisas (*IoT*), o hardware da pesquisa corrente. Após isso na Seção 2.2 será explicado o que é sensibilidade ao contexto e como isso afeta os dispositivos* IoT. *Além disso na subseção 2.2.1 que pertence a sensibilidade ao contexto, será explanado também o que são middlewares publish-subscribe e suas características. Por último na Seção 2.3 é exibido o conceito de tratamento de exceções e como ele pode ser realizado no âmbito da internet das coisas auxiliado pelas informações de contexto.*

2.1 Internet das Coisas

O termo *Internet of Things*(IoT) foi cunhado pela primeira vez por Ashton (2009) em uma apresentação feita na Procter & Gamble (P&G) em 1999. Ele acreditava que as "coisas" e o mundo físico que nos cerca precisavam ser repensados e que as ideias e informações são importantes, mas "coisas" são muito mais. Desde então o termo vem sendo amplamente utilizado para o que se tornou um novo paradigma que se baseia em fazer as "coisas" parte do ambiente da internet. Essas "coisas" podem incluir dispositivos eletrônicos ou aparelhos domésticos como dispositivos médicos, geladeiras, e sensores (KRISHNAMURTHY; MAHESWARAN, 2016). Segundo DUMITRU (2017) a IoT geralmente permite o controle remoto dos objetos fazendo uso da infraestrutura de comunicação existente e a integrando com a rede de computadores clássica. O autor também cita que em 2017 foi estimado que existem 50 bilhões de objetos IoT já conectados a rede. Dessa forma podemos notar que sua utilização já é feita em larga escala e por isso questões envolvendo os objetos IoT tem sido levantadas. Com a revisão da literatura, podemos notar uma questão que não tem sido muito abordada no paradigma, tal questão

é o tratamento de exceções voltado para os dispositivos da *IoT*.

2.2 Sensibilidade ao Contexto

Sensibilidade ao contexto é a capacidade que um sistema tem para conhecer o estado e o ambiente físico ao redor do usuário (DEY; ABOWD, 1999). Como já mencionado na Seção 2.1, os dispositivos *IoT* tem uma natureza muito heterogênea e móvel, dessa forma, esse tipo de dispositivo tem a necessidade de ser sensível ao contexto que o cerca, como por exemplo um *smartwatch* que necessita de informações de contexto como localização, giro, batimentos cardíacos para poder realizar as medições corretas de atividades físicas, frequência cardíaca, movimentação durante o dia, entre outras funções as quais o algoritmo desse tipo de dispositivo se propõe a realizar. Apesar disso ser uma clara necessidade, segundo Viterbo et al. (2008), implementar a chamada sensibilidade ao contexto é um processo complexo devido a especificidade e heterogeneidade dos ambientes em execução. Por essa razão esse tipo de dispositivo tem que se adaptar ao contexto presente para melhor atender os usuários. Essa heterogeneidade pode ser um grande empecilho para os programadores que visam a criação de sistemas sensíveis ao contexto para dispositivos *IoT*, pensando nisso o desenvolvimento desse tipo de aplicação pode ser facilitado com a utilização de *middlewares* de provisão de contexto (VITERBO et al., 2008). Na próxima seção será apresentado o principal tipo de *middleware* utilizado na literatura para realizar o desenvolvimento de aplicações sensíveis ao contexto.

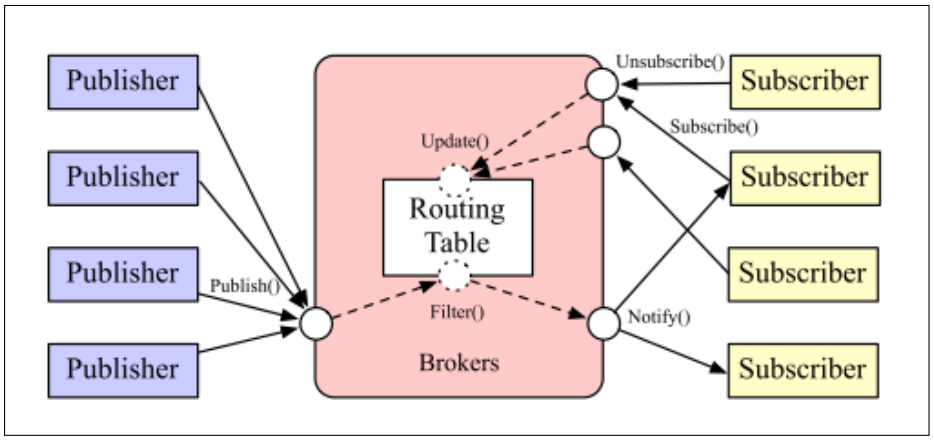
2.2.1 Middlewares Publish-Subscribe

Na computação distribuída há o crescimento da utilização de middlewares para facilitar a comunicação entres os componentes (DAMASCENO; CACHO, 2006). Isso se deve pelos benefícios que trazem ao processo de desenvolvimento de sistemas sensíveis ao contexto como abstrações que reduzem a complexidade da heterogeneidade dos dispositivos e também de falhas de rede. Segundo Capra *et. al* (2003), os *middlewares* da computação distribuída tradicional são baseados no princípio da transparência, em que detalhes de implementação são escondidos de usuários e desenvolvedores, dessa forma, o *middleware* constitui uma camada entre o sistema operacional de rede e a aplicação em si.

Para o foco em sistemas distribuídos móveis, seja para o modelo de agentes móveis ou para o novo paradigma *IoT*, foi constatado na literatura que parte dos autores utilizam de *middlewares publish-subscribe* para realizar as tarefas sensíveis ao contexto, isso

se deve pelo fato de que esse paradigma é uma alternativa possível para o tratamento de aplicações móveis em larga escala (GADDAH; KUNZ, 2003). Isso se deve pelas características que esse tipo de *middleware* possui como a comunicação assíncrona e a comunicação anônima em que o publicador do conteúdo não precisa conhecer o subscritor para que a comunicação ocorra. A Figura 2.1 mostra uma arquitetura *publish/subscribe* baseada em tópicos, em que os *subscribers* expressam interesse em determinados tipos de eventos e são notificados com a publicações geradas pelos *publishers*. Também na Figura 2.1 pode ser visto que os *brokers* são colocados no meio da infraestrutura para mediar a comunicação entre os *publishers* e os *subscribers*. Dessa forma, esse tipo de middleware se torna ideal para realizar o desenvolvimento de aplicações móveis sensíveis ao contexto, ainda mais com a utilização do protocolo de conectividade *Message Queue Telemetry Transport* (MQTT) [1] que foi desenvolvido para ser um mensageiro muito leve para mecanismos *publish/subscribe*.

Figura 2.1: Arquitetura Publish/Subscribe



Fonte: *(CHANG; DUAS; MELING, 2014)*

Segundo Chang, Duas e Meling (2014) o paradigma *publish/subscribe* tem atraído muito a atenção tanto da academia quanto da industria pelo fato de ser muito performático e escalável. Também segundo eles, enquanto usuários padrões que navegam pela internet não são tão afetadas com perda de pacotes, aplicações industriais de larga escala demandam um tempo rigoroso de resposta além de requerer uma tolerância a falhas rigorosa. Por essas características, o paradigma vem sendo adotado em larga escala para os *middlewares* desenvolvidos nos últimos anos.

O corrente trabalho teve como foco a busca de *middlewares* que lidam explicitamente com os dispositivos da *IoT*, sendo assim, foi encontrado através de uma busca *adhoc*, alguns *middlewares* recentes que entram nessa categoria, dentre eles estão:

[1] http://mqtt.org/

- *KAA* [2] é uma plataforma *middleware publish/subscribe* voltado para construção de soluções completas *end-to-end IoT* conectada com aplicações e produtos *smart*. O foco do *middleware* é no chamado gerenciamento de identidade ou *profiling*, ou seja, a capacidade de pegar a lista de dispositivos *IoT* que estão ativos e registrar no banco de dados. Esse dados de *profiling* podem ser qualquer tipo de informação relacionada ao dispositivo, como a versão do *firmware*, versão do hardware, extensões instaladas. A plataforma utiliza de meios como Apache Avro [3] para saber o tipo de dado com que está trabalhando e permite a construção de grupos com base na junção de profiles. Esses profiles podem ser agrupados com base na localização, versão de *firmware*, identidade do usuário, entre outros tipos de agrupamento. A aplicação permite notificações dos dispositivos inscritos e troca de mensagens entre os dispositivos. Todo processo é feito através de uma API REST [4]. Além disso o KAA utiliza de várias tecnologias e se conecta aos mais diversos tipos de sistemas operacionais (Figura 2.2).

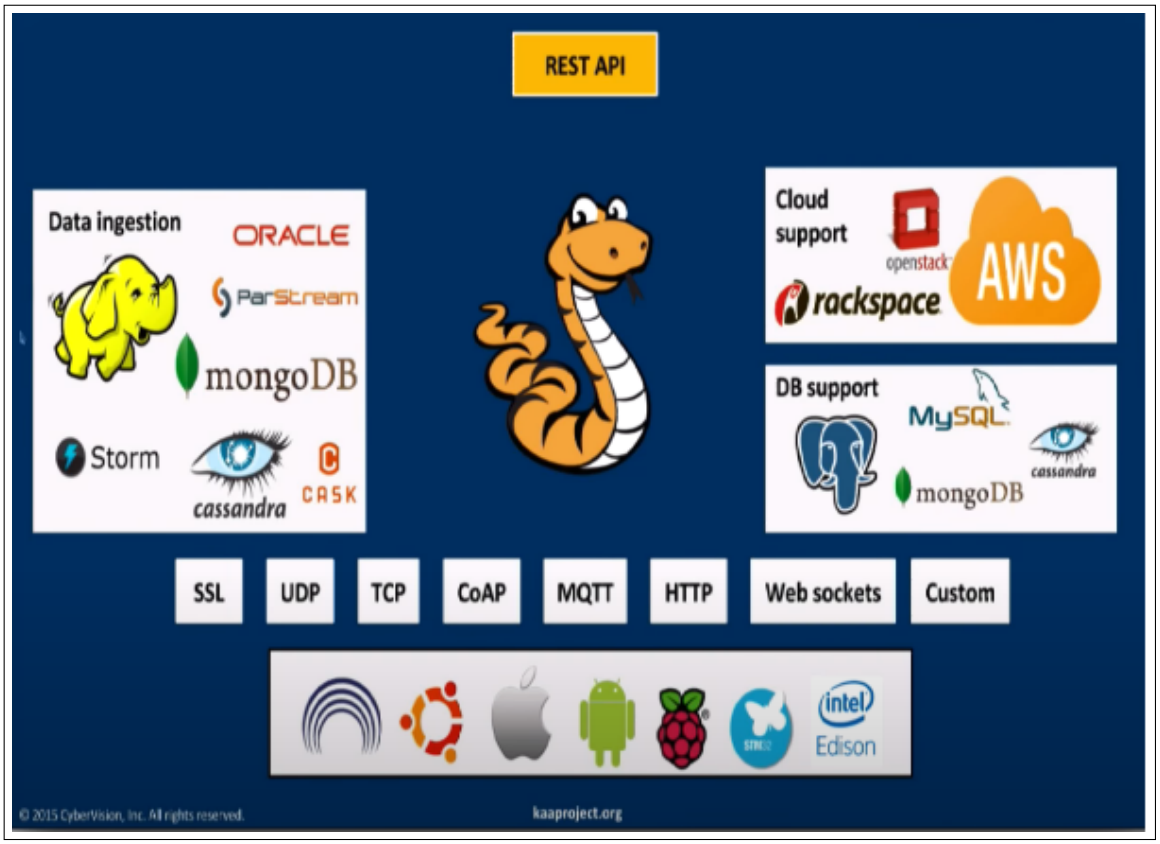
- *Sofia2* [5] é um *middleware publish/subscribe* que permite interoperabilidade de múltiplos sistemas e dispositivos, oferecendo uma semântica que permite colocar a informação do mundo real à disposição de aplicações *IoT*. É multilinguagem e multiprotocolo, permitindo assim a interligação de dispositivos heterogêneos. Proporciona mecanismos de publicação e subscrição, facilitando a coordenação de sensores e atuadores para monitorizar e atuar no ambiente.

- *OpenIoT* [6] é uma plataforma *middleware publish/subscribe IoT*, que inclui funcionalidades como a capacidade de compor dinamicamente e sob demanda serviços *IoT* não triviais. Essa plataforma está defasada e na atual data deste trabalho, a última atualização do repositório foi feita em novembro de 2015 e o projeto conta com vários problemas em aberto para resolução no repositório. Além disso o processo de instalação para utilização da plataforma é complicado, havendo problemas em sua instalação, levando a conclusão de que o projeto foi descontinuado.

- *FIWARE* [7] é uma plataforma *middleware publish/subscribe open source* [8] com objetivo de unir componentes de terceiros para acelerar o processo de desenvolvimento de

[2] https://www.kaaproject.org/
[3] http://avro.apache.org/
[4] *Application Programming Interface Representational State Transfer (é uma interface que fornece dados em um formato padronizado baseado em requisições HTTP.*
[5] http://sofia2.com/
[6] https://github.com/OpenIotOrg/openiot
[7] https://www.fiware.org/
[8] Código-fonte livre para ser adaptado para diferentes fins sem custo de licença

Figura 2.2: Tecnologias, Sistemas e Protocolos compatíveis com o KAA.



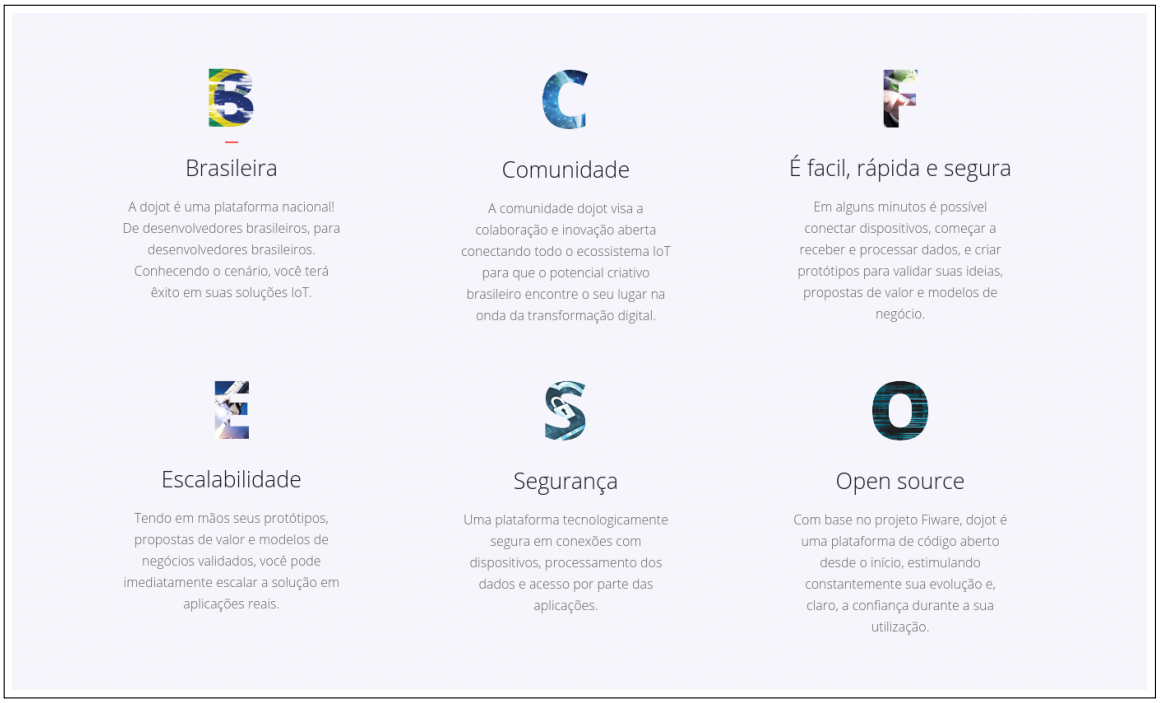
Fonte: *www.kaaproject.org*

soluções inteligentes. Ele promete a construção de soluções inteligentes e especifica claramente a necessidade de se ter a informação contextual, processá-la e informar a atores externos sobre ela, dessa forma, ele habilita o sistema a atualizar e acessar o atual estado de contexto. Ele propõe uma *API* robusta para o gerenciamento das informações de contexto. Empresas como IBM, Telefônica, utilizam de componentes deste *middleware* para construir suas próprias aplicações *IoT*.

- *Dojot* [9] é uma plataforma *middleware publish/subscribe* voltada para facilitar o desenvolvimento de soluções e o ecossistema *IoT* com conteúdo local voltado às necessidades brasileiras. A plataforma conta com uma interface amigável e uma fácil conexão dos dispositivos *IoT* através do protocolo *MQTT*. A limitação fica por conta do retorno da informação dos dispositivos *IoT* ficar atrelada a um mecanismo de programação de fluxo pela interface e não a uma *API*. As principais características do *Dojot* estão listadas na Figura 2.3.

[9] http://www.dojot.com.br/

Figura 2.3: Características do Dojot.



Fonte: *www.dojot.com.br*

2.3 Tratamento de Exceções

Ainda que os dispositivos da *IoT* estejam cada vez mais intrínsecos no dia a dia das pessoas, como mostrado na Seção 2.1, existe grande desconfiança por parte das pessoas com relação as falhas que esses dispositivos podem causar e os danos que podem acarretar ao dia a dia e a saúde dos seres humanos (ASQUITH, 2017). Dessa forma podemos notar que um requisito muito importante para os diversos tipos de sistemas incluindo também os que rodam nos dispositivos da *IoT* é a confiabilidade. A confiabilidade é um requisito muito importante de sistemas computacionais. Apesar disso, a confiabilidade no mundo real nem sempre pode ser alcançada em decorrência de falhas que naturalmente acontecem nos sistemas (WEBER, 2002). Dessa forma, na literatura são adotadas técnicas de tolerância a falhas, que lidam com as falhas que vem a surgir. Na definição de Lee e Anderson (1990), *Tolerância a falhas* é a capacidade de um sistema computacional de preservar a suas funcionalidades sem apresentar defeitos, mesmo na presença de falhas. Sendo assim, em situações em que o sistema se encontra em um estado errôneo um componente do sistema deve gerar as exceções que modelam o erro e o sistema tem por responsabilidade tratar as exceções que foram levantadas.

O tratamento de exceções é uma técnica de recuperação de erros por avanço que é

utilizada na melhoria da robustez e confiabilidade dos sistemas (ROCHA, 2013). Sendo assim, nesta seção iremos detalhar as principais características do tratamento de exceções voltado a sensibilidade de contexto.

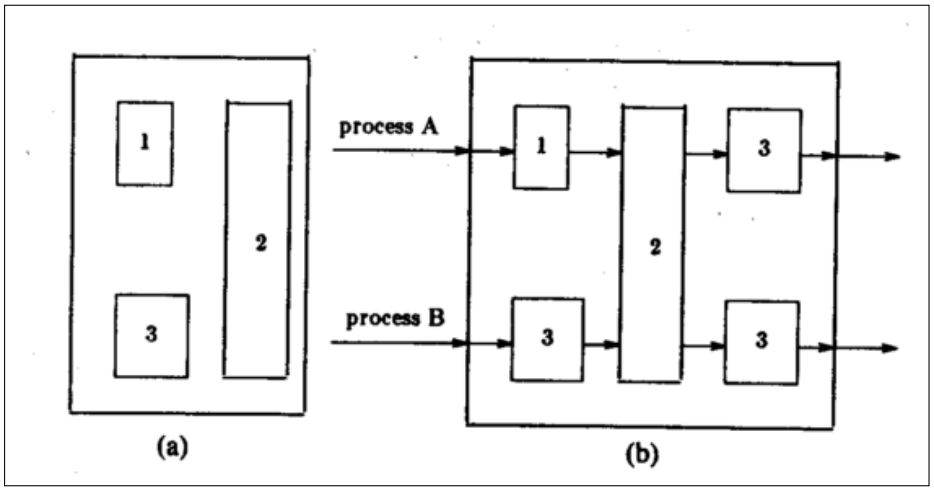
2.3.1 Falta, Erro, Falha e Exceção

Uma falha (*failure*) é um evento que ocorre quando o sistema que está em execução desvia da sua especificação e pode ser observada por usuários externos. Segundo Rocha (2013) a falha pode ser vista como o estado de um sistema em que o serviço correto estava sendo entregue, para um estado onde o serviço incorreto passa a ser entregue. Já um erro (*error*) pode ser definido como o estado total do sistema que pode posteriormente levar a uma falha (AVIZIENIS et al., 2004). A causa física ou algorítmica de um erro é chamado de falta (*fault*). Dessa forma um estado errôneo pode propagar-se desde falta até resultar em uma falha no sistema. Já uma exceção (*exception*) é um evento que modela uma situação em que o fluxo de execução do sistema não pode continuar, e dessa forma, para dar procedimento correto de sua execução uma computação adicional tem que ser feita (KNUDSEN, 1987).

2.3.2 Recuperação de Erros em Sistemas Assíncronos

Como falado anteriormente uma das maiores dificuldades de se implementar o tratamento de exceções para dispositivos IoT é a sua comunicação de forma assíncrona. Esse tipo de ambiente assíncrono caracteriza-se também por ter a chamada concorrência dos processos. Segundo Campbell e Randell (1986) implementar tolerância a falhas para sistemas com processos concorrentes é complicado pela possibilidade de comunicação de informações incorretas e a necessidade de coordenar os processos na recuperação. Os autores também disseram que generalizar um conceito para realizar o tratamento de exceções em sistemas assíncronos requer estruturas adicionais de sistema para realizar a cooperação e coordenação de processos individuas. Nesse sentido, para que seja realizada a recuperação de erros é necessária a implementação de ações atômicas como mostrado na Figura 2.4, em que em (a) temos a estrutura planejada de uma ação atômica, e em (b) temos a execução das ações atômicas entre os processos A e B, mostrando seu fluxo com os pontos de entrada e saída. Essa sincronização dos processos é muito importante para garantir a ocorrência da ação atômica e por consequência o sucesso da recuperação no ambiente assíncrono.

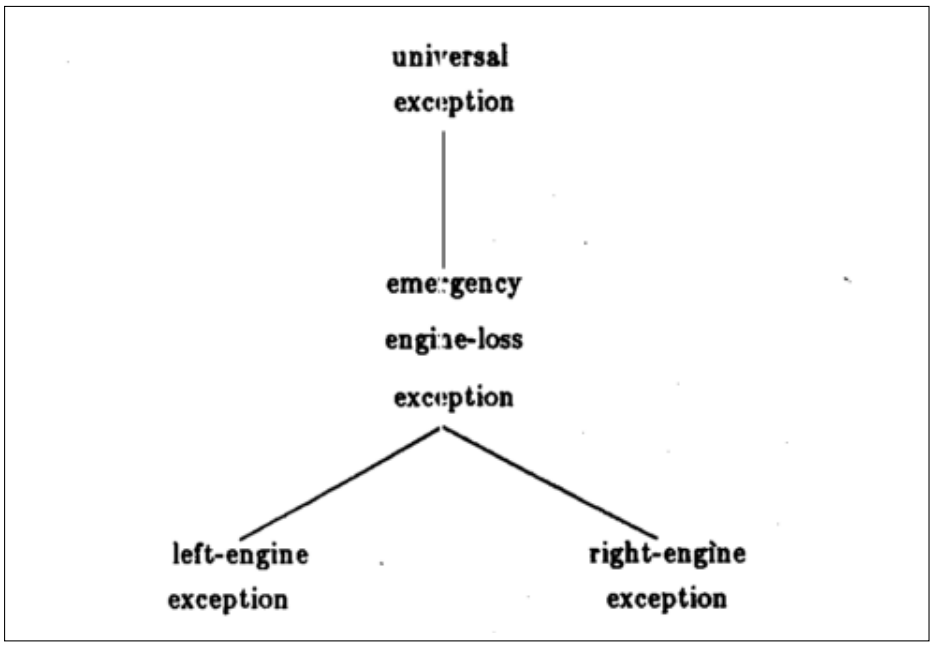
Figura 2.4: Sistema com seus componentes e processos



Fonte: *(CAMPBELL; RANDELL, 1986)*

Segundo Campbell e Randell (1986) quando dois ou mais componentes de uma uma ação atômica levantam exceções de forma concorrente, a resolução dessas exceções deve ser feita com a utilização de uma hierarquia excepcional que pode ser implementada através de uma árvore de exceções. Esse conceito de árvore de exceções consiste de exceções menores relacionadas aos seus tratadores, mas quando levantadas juntas, ou seja de forma concorrente, compõem uma exceção maior que também tem o seu tratador associado. O processo é mostrado na Figura 2.5, em que temos a árvore de exceções para um avião a jato. Nesse cenário temos as exceções nas folhas da árvore que seriam as exceções mais simples, como a exceção de falha no motor esquerdo e exceção de falha no motor direito. O levantamento dessas exceções em conjunto gera a exceção pai das duas que é o caso excepcional de emergência, em que os dois motores foram perdidos.

Figura 2.5: Árvore de Exceções



Fonte: *(CAMPBELL; RANDELL, 1986)*

2.3.3 Tipos de Exceções

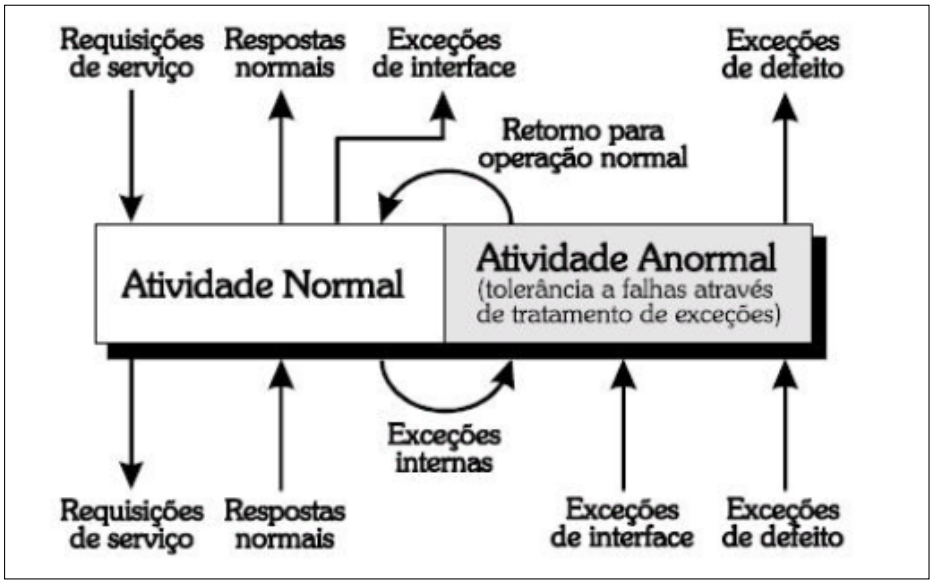
Lee e Anderson (1990) (Figura 2.6) criaram um modelo de tolerância a falhas em que cada componente pode receber requisições enviadas por outros componentes. Quando as respostas das requisições se encontram em conformidade com a especificação, elas são chamadas de respostas normais. Já quando o componente não é capaz de atender a requisição de serviço, então ele retorna respostas anormais ou exceções (LIMA, 2013) (ROCHA, 2013) (LEE; ANDERSON, 1990). Também segundo Lee e Anderson (1990) as exceções podem ser classificadas como:

- **Exceções de interface**: são sinalizadas em resposta a requisição de um serviço não disponível ou serviço invocado com conjunto invalido de entradas;
- **Exceções internas**: são levantadas quando um componente detecta uma situação inesperada durante sua atividade normal e um tratador local é ativado;
- **Exceções de defeito**: são sinalizadas quando por algum motivo, um componente não pode prover o serviço especificado;

Lee e Anderson (1990) explicam que as exceções apresentadas podem ser classificadas em dois tipos: o primeiro são as exceções internas, que podem ser levantadas internamente

pelo componente, e o segundo são as exceções externas (de interface e serviço), que são sinalizadas externamente entre componentes.

Figura 2.6: Componente Tolerante a Falhas Ideal



Fonte: *(LEE; ANDERSON, 1990)*

O componente de tolerância a falhas ideal prevê dois tipos de comportamento, o comportamento de **atividade normal** é aquela em que o componente atende as especificações. Por outro lado o componente que possui **atividade excepcional** deve realizar uma computação adicional para tratar a condição errônea em que se encontra. Dessa forma, se um componente não conseguir tratar a exceção, então o mesmo deve propagar a exceção para o componente de mais alto nível e depois retornar ao seu fluxo normal.

2.3.4 Tratamento de Exceções Sensível ao Contexto

O Tratamento de Exceções Sensível ao Contexto vem do termo inglês *Context-Aware Exception Handling (CAEH)* em que o contexto segundo Dey e Abowd (1999), é um conjunto de informações relevantes sobre o ambiente físico ou lógico, que podem ser os próprios usuários e até mesmo o sistema.

Como já falado anteriormente no Seção 2.1, os dispositivos da *IoT* fazem parte da chamada Computação Móvel e por essa razão tem a necessidade de verificar o contexto que se encontram para tratar suas variações e adequar a execução a elas. Dessa forma, mudanças de contexto como temperatura, umidade e localização podem vir a gerar exceções e eventos excepcionais sendo considerado como contexto excepcional. É dever do

sistema tratar as exceções geradas por esses tipos de eventos, mas essa tarefa não é trivial pelas características como abertura, mobilidade, conexão instável e comunicação assíncrona (DAMASCENO; CACHO, 2006). Tudo isso pode prejudicar o tempo hábil dos programadores que irão se preocupar mais com a forma de se realizar o tratamento de exceções do que com a lógica de negócio em si.

Um exemplo da aplicação de tratamento de exceções sensível ao contexto é o sistema *UbiParking* (ROCHA, 2013; LIMA, 2013; FILHO et al., 2014). Esse sistema possui a funcionalidade de disponibilizar um mapa plotado com todas as vagas de estacionamento disponíveis por região. O mapa de vagas livres é atualizado através de sensores implantados nos acostamentos das vias e nos estacionamentos públicos. Os sensores detectam quando uma vaga está ocupada ou livre e, dessa forma, através de um outro dispositivo *IoT* como um *smartphone/tablet* ou até mesmo no computador de bordo do veículo, as pessoas poderão obter informações sobre a disponibilidade de vagas na região. Além disso, o usuário pode reservar uma vaga e solicitar ao sistema a rota mais apropriada com base em algum critério de sua preferência (e.g., menor distância, maior numero de vagas livres disponíveis ou menor preço) (ROCHA, 2013; LIMA, 2013; FILHO et al., 2014). No momento em que o motorista chega ao estacionamento escolhido, o *UbiParking* conduz o motorista até a vaga reservada ou a vaga livre mais próxima.

O estacionamento ubíquo conta com sensores de monitoramento da temperatura, de modo que quando a temperatura diminui a limiar inferior (e.g., 15 °C) o sistema de aquecimento é acionado. Quando a temperatura se encontra entre os limiares (15 °C $<$ temperatura $<$ 35 °C) apenas a ventilação permanece em funcionamento (ROCHA, 2013; LIMA, 2013; FILHO et al., 2014).

O sistema exemplificado acima possui dois cenários de execeções contextuais que devem ser tratadas.

- Exceção de Incêndio: representa um condição em que há um incêndio dentro do estacionamento. O sistema detecta por meio dos sensores de temperatura e fumaça, e após isso trata a exceção contextual acionando os aspersores, notificando o corpo de bombeiros e conduzindo os motoristas em movimento para fora do estacionamento (ROCHA, 2013; LIMA, 2013; FILHO et al., 2014).

- Exceção de Vaga Indisponível: “Essa exceção contextual modela uma situação em que o veículo está em movimento dentro do pátio de vagas indo em direção a sua vaga reservada. Porém, outro veículo ocupa aquela vaga. Nesse caso, se a vaga for

> a ultima vaga livre disponível, fica caracterizado a situação de anormalidade. Essa exceção contextual é detectada pelo sistema através de informações de contexto sobre as reservas de vagas, a localização do veículo os dados que vem dos sensores de detecção de vaga ocupada. Como forma de tratar essa exceção contextual, o *UbiParking* conduz o veículo até o lado de fora do estacionamento, onde outra vaga livre em outro estacionamento pode ser localizada para ele." (ROCHA, 2013; LIMA, 2013; FILHO et al., 2014)

Na literatura existem vários trabalhos que focam em tentar melhorar o processo de desenvolvimento para tratar esse tipo de exceção, dentre eles estão, os trabalhos de Lopes, Cacho e Batista (2007), Damasceno e Cacho (2006), Filho et al. (2014) dentre outros que serão mostrados junto com suas principais características no capítulo de revisão bibliográfica da literatura.

2.4 Considerações Finais

Neste capítulo foram tratados os conceitos básicos necessários para o entendimento do mecanismo. Todos os conceitos escritos foram baseados no estado da arte dos temas e em seus principais pesquisadores. No próximo capítulo serão apresentados os principais trabalhos que compõem o estado da arte para o tema adotado.

Capítulo 3

Revisão da Literatura e o Estado da Arte

O levantamento bibliográfico a fim de identificar o atual estado da arte foi uma peça chave para a pesquisa realizada. Particularmente, para essa pesquisa foram avaliados os métodos para a realização do tratamento de exceções sensível ao contexto voltado não somente para dispositivos IoT*, mas sim para qualquer tipo de dispositivo. Esta etapa foi apoiada pelo uso da ferramenta* Web Parsifal*, que possibilitou, além da busca em diferentes plataformas de informação, uma forma de definir critérios para a inclusão, ou exclusão, dos trabalhos mais relacionados ao tema escolhido. A seguir é apresentada a descrição do método utilizado para tal tarefa e, em seguida, os trabalhos selecionados a partir de tal método.*

3.1 Método para o Levantamento Bibliográfico

A fim de garantir a condução coesa na etapa de identificação dos estudos relacionados ao tema desta pesquisa, optou-se pela utilização de uma ferramenta que auxilia no processo do levantamento bibliográfico, a *Parsifal* [1]. Foi utilizado também um gerenciador de referências bibliográficas o *Mendeley* [2] que facilitou o processo de extração dos dados. Muito trabalhos encontrados na base Scopus, redirecionam para a base IEEE, mas por serem encontrados a partir da *string* de busca na Scopus, foram marcadas como advindas da mesma. Ao todo, 4 fontes de informação disponíveis na Internet foram consultadas, sem restrições quanto à data de publicação, para o levantamento inicial dos trabalhos correlatos ao tema desta pesquisa de mestrado, são elas: (1) ACM digital library [3], (2) IEEE [4], (3)

[1] https://parsif.al/
[2] https://www.mendeley.com/homepage2/?switchedFrom=
[3] http://dl.acm.org/
[4] http://ieeexplore.ieee.org/Xplore/

Scopus [5] e (4) Google Scholar [6]. Cada uma das 4 fontes de dados listadas foi consultada e devido as diferenças de resultados e dos operadores lógicos de busca para cada base, a *string* de busca teve de ser alterada para se adequar. Durante o levantamento bibliográfico foram encontrados muitos trabalhos relacionados a *Business Process Management (BPM) e workflow*, este trabalho não aborda os conceitos de tratamento de exceções voltado para essas duas linhas de pesquisa. As *strings* para cada base estão listadas a seguir:

- ACM *digital library: context-aware exception handling*
- IEEE: *((exception handling) AND context-aware)*
- Scopus: ((*"Context aware"* OR "Sensibilidade a contexto") AND (*"Exception Handling"* OR "Tratamento de exceções"))
- Google Scholar: ((*"Context aware"* OR "Sensibilidade a contexto") AND (*"Exception Handling"* OR "Tratamento de exceções"))

Tabela 3.1: Mecanismos advindos da Scopus que lidam com tratamento de exceções sensível ao contexto, em conjunto com suas referências e características importantes

Toward Efficient Detection of Semantic Exceptions in Context-Aware Systems (CHO; HELAL, 2012b)
- Linguagem de alto nível com açúcar sintático para facilitar o tratamento de exceções; - Linguagem intermediária (ILSE) que é convertida em um automato finito e permite otimizações; - Permite o tratamento de exceções de erros estáticos, semi-dinâmicos e dinâmicos;
Dynamic Parameter Filling for Semantic Exceptions in Context-Aware Systems (CHO; CHOI; HELAL, 2013)
- Continuação do *Toward Efficient Detection of Semantic Exceptions in Context-Aware Systems*; - Processo de otimização de parâmetros de funções da ILSE como a *toonear()*; - Melhoria dos algoritmos que encontram os valores de taxas de sensibilidades para os sensores

Continua na próxima página

[5] https://www.elsevier.com/solutions/scopus
[6] https://scholar.google.com.br/

Tabela 3.1 – continuação

Expressive Exceptions for Safe Pervasive Spaces (CHO; HELAL, 2012a)
- Conceito de situação ao invés de contexto; - Mecanismo claramente visa a melhoria de expressividade e eficiência de se descrever exceções para os sistemas ubíquos;
Context-oriented Exception Handling (BARBOSA, 2009)
- Modelo baseado no paradigma Holo; - Suporte a composição dinâmica de contextos/eventos; - Utilização de características do paradigma Holo como *"being"* que representa um contexto e é um tipo objeto de dados; - A composição de *"being"* representa a composição de contextos; - Presença de tratadores e de um modelo de propagação de exceções; - Tratamento de exceções pode ser feito em tempo de execução, ou seja, há uma definição dinâmica de tratadores; - Anexo de tratadores pode ser feito em diferentes níveis; - Modelo híbrido de propagação de exceções, em que a propagação pode ser feita de maneira automática ou explícita;
Application-Level Recovery for Context-Aware Pervasive Computing (KULKARNI; TRIPATHI, 2008)
- Mecanismo para recuperação de erros por avanço que faz a integração do modelo de tratamento de exceções com um modelo de comunicação de eventos assíncrono; - Assim como o *middleware* MoCA (DAMASCENO et al., 2006) o mecanismo proposto é focado na construção de aplicações multi-usuários colaborativas com agentes e é do tipo *publish/subscribe*; - Suporte a ações de recuperação dirigidas a eventos; - Suporte a participação do usuário na recuperação de erros que não foram tratados automaticamente; - Define abstrações como papéis, objetos e reações, além de denominar como "atividade" a programação de aplicação sensíveis ao contexto; - Abstração de "agente" em que a sua principal função é manter as informações de contexto e gerar eventos contextuais;

Continua na próxima página

Tabela 3.1 – continuação

- No modelo, a aplicação deve subscrever aos eventos dos agentes e adaptar o seu comportamento; - Aplica conceitos de escopos e propagação de exceções de acordo com regras pré definidas;
A Programming Framework for Implementing Fault-Tolerant Mechanism in IoT Applications (WANG et al., 2015)
- Realiza o tratamento de exceções mas não é o tema principal do mesmo, que acaba focando mais na tolerância a falhas como um todo; - Introduz o conceito de objetos de recursos compartilhados globalmente e de objetos de estado crítico; - Especificação do tratamento de erros através do bloco *catch*; - Cada padrão de regras é feito no estilo Evento-Condição-Ação (ECA), que representa uma estratégia de recuperação a respeito de uma condição de erro especificada; - Proposição de classes como `ContextLogger` e `FailureHandler`, em que o `ContextLogger` é usado para salvar um objeto de estado crítico e o `FailureHandler` para propagar as mensagens de exceção;
Handling Exceptions in Situation-Driven Agent Systems (KIM; KIM, 2009)
- Conceito de situação ao invés de contexto, em que a situação agrega mais informações do que o contexto como o próprio contexto e ações referentes ao problema que se quer resolver; - Propõe um modelo de fluxo para realizar o tratamento de exceções voltado a situações em agentes; - Tratamento de exceções composto por duas formas principais: *self-handling* e *blackbord*;
A Formal Modeling for Exceptions in Context-Aware Systems (YOON et al., 2014)
- Modelo formal para o tratamento de exceções; - Provê detecção automática de exceções independentes da aplicação; - Provê uma forma conveniente de programadores construírem aplicações que tratam exceções semânticas, ou seja, exceções que são específicas a aplicações sensíveis ao contexto;

Continua na próxima página

Tabela 3.1 – continuação

- Trabalha com o conceito de situações, em que agrega mais informações além do contexto; - Novo modelo de tratamento baseado em regras de Situação-Ação (SA);
A method for model checking context-aware exception handling (ROCHA; ANDRADE; GARCIA, 2013)
- Propõe um método para verificação de modelos do tratamento de exceções sensível ao contexto; - Permite a modelagem através do formalismo de uma estrutura de estados Kripke; - Conceito de modelo como proposições contextuais, escopos de tratamento, casos de tratamento;
A Situation-based Exception Detection Mechanism for Safety in Pervasive Systems (ROCHA; ANDRADE; GARCIA, 2013)
- Assim como o mecanismo de Kim e Kim (2009), também trabalha com o conceito de situação; - Conceito de padrão de situação, que é um padrão temporal de um sequência de eventos e outras situações; - Proposição de dois algoritmos, para processamento de eventos e avaliação das situações; - Algoritmo baseado em uma extensão de um automato finito, para representar a transição de estados de situações;
Exception Handling in Pervasive Service Composition Using Normative Agents (GUTIERREZ-GARCIA; RAMOS-CORCHADO, 2011)
- Propõe um mecanismo de composição de serviços baseado em agentes; - Modelo de serviços ubíquos com papeis de agentes em que cada agente tem suas "obrigações" na semântica de interações; - Conceito de organizações virtuais compostas de agentes que tem objetivos compartilhados; - Sempre que uma *trigger* é executada, um conjunto de obrigações é induzido na mesma e dessa forma, um conjunto de obrigações é distribuída para cada membro da organização;

Continua na próxima página

Tabela 3.1 – continuação

Towards the definition of a Context-Aware Exception Handling Mechanism (BEDER; ARAUJO, 2011)
- Incorpora um mecanismo de tratamento de exceções a um *middleware* de serviços multi-camadas *publish/subscribe* chamado MidSensorNet; - Conceito de *Wireless Sensor Networks* (WSN) que monitora dinamicamente as mudanças de contexto; - Arquitetura de camada dupla, em que uma das camadas utiliza de lógica fuzzy e ontologias para interpretar o contexto;
ConExT-U: A context-aware exception handling mechanism for task-based ubiquitous systems (FILHO et al., 2014)
- Modelo robusto baseado em tarefas *Task-Based Development Model* (TBDM), em que as tarefas estão relacionadas ao o que o usuário precisa fazer, enquanto os serviços estão relacionadas a como a tarefa a ser realizada irá estar disponível na infraestrutura computacional; - Assim como o MoCA (DAMASCENO et al., 2006), também desenvolve o tratamento de exceções em cima de um middleware, mas agora unido ao conceito de TBDM; - Conceito de exceções contextuais, propagação de exceções e busca por tratadores adequados;

Tabela 3.2: Mecanismos advindos da ACM *digital library* que lidam com tratamento de exceções sensível ao contexto, em conjunto com suas referências e características importantes

Context-Aware Exception Handling in Mobile Agents Systems: The MoCa Case (DAMASCENO et al., 2006)
- Proposição de um middleware *publish/subscribe* voltado para sistemas de agente móvel; - Suporte explícito para tratamento de exceções sensível ao contexto; - Mecanismo composto por 3 serviços *Context Information Service* (CIS), *Symbolic Region Manager* (SRM) e *Location Inference Service* (LIS); - Modelo em que cada agente móvel deve ter os seus tratadores excepcionais que constituem a parte de atividades excepcionais;

Continua na próxima página

Tabela 3.2 – continuação

- Possui o conceito de escopos e realiza a propagação de exceções quando a mesma é levantada fazendo com que seja levada a outro escopo para ser tratada;
- Três diferentes tipos de escopos: dispositivo, região e grupo;
- Possui o conceito de contextos excepcionais;

Tabela 3.3: Mecanismos advindos do Google Scholar que lidam com tratamento de exceções sensível ao contexto, em conjunto com suas referências e características importantes

Tratamento de exceções sensível ao contexto (DAMASCENO; CACHO, 2006)
- Baseado no trabalho feito em (DAMASCENO et al., 2006); - Propõe um novo modelo e uma arquitetura para tratamento de exceções, identificando os principais aspectos que faltam na literatura para sua aplicação; - Especifica os escopos para o tratamento de exceções como dispositivo, servidor, regiões e grupos; - Detalha o que deve ser uma propagação sensível ao contexto e um tratamento proativo; - Detalha componentes da arquitetura proposta de forma a utilizar um *middleware* como um componente externo que interage com os componentes propostos;
Um Mecanismo de Composição de Eventos para Resolução de Exceções Sensíveis ao Contexto (LOPES; CACHO; BATISTA, 2007)
- Propõe um mecanismo de composição de eventos que utiliza de múltiplos *middlewares publish/subscribe*, dentre ele o MoCa de (DAMASCENO et al., 2006); - Especifica uma linguagem de composição de eventos que conta com operadores *OR, AND, SEQUENTIAL, INTERACTOR, INTERVAL, PARALLEL*; - Propõe uma extensão da arquitetura para dar suporte a resolução de exceções concorrentes;

3.2 Considerações Finais

Neste capítulo foram apresentados os trabalhos da literatura que tinham maior relação com a pesquisa apresentada nesta dissertação. Os trabalhos que foram selecionados apresentam conceitos e abordagens interessantes, que contribuem para a composição das informações sobre tratamento de exceções voltado para sistemas sensíveis ao contexto e deram a base para este trabalho.

Vale destacar que os únicos trabalhos que visam uma abordagem empírica para a realização do tratamento de exceções sensível ao contexto são os trabalhos de Damasceno e Cacho (2006) e Lopes *et al.* (2007), sendo o restante dos outros trabalhos limitados ao maior ponto em comum da realização do tratamento de exceções para agentes móveis. Estes trabalhos muitas vezes não demonstrando exemplos programáticos para a realização do que é proposto. Muitas vezes exibem somente conceitos em alto nível que seriam utilizados junto a algum *middleware* ou até mesmo sem *middleware* como o caso do trabalho de Rocha *et al.* (2013). Os conceitos acabam sendo divergentes entre eles que ao final do levantamento bibliográfico não foram constatados conceitos em comum para que haja uma comparação de tais abstrações.

Ambos os trabalhos de Damasceno e Cacho (2006) e Lopes *et al.* (2007) assim como na pesquisa atual propõem novas abstrações para realizar o tratamento de exceções sobre a base de *middlewares publish-subscribe* e focam nos conceitos de propagação. Além disso eles propõem uma arquitetura em níveis inferiores para tal assim como este trabalho de pesquisa. Os trabalhos citados anteriormente apesar de possuírem todas essas características citadas que foram úteis para a construção dessa pesquisa, não tratam exatamente o conteúdo abordado nesta dissertação pelo fato de não possuírem em sua maioria o foco nos dispositivos da IoT em conjunto com a resolução concorrente de exceções que podem vir a ser levantadas. Além de não realizarem o tratamento explícito de exceções, dessa forma, ficou visível a necessidade de criação de um novo mecanismo para lidar com o tratamento de exceções voltado para esse tipo de dispositivo. No próximo capítulo será apresentada a arquitetura do mecanismo desenvolvido.

Capítulo 4

Mecanismo de Tratamento de Exceções Sensível ao Contexto

Neste capítulo, será apresentada a arquitetura em alto nível do mecanismo, que possibilita realizar o tratamento de exceções sensível ao contexto voltado para dispositivos da internet das coisas. A arquitetura a ser apresentada foi idealizada a partir das estratégias apresentadas pelos trabalhos descritos no Capítulo 3, de forma a suprir as carências que a área selecionada ainda possui e que não foram tratadas pelos trabalhos disponíveis na literatura: realizar o tratamento de exceções sensível ao contexto que até então havia sido pouco explorado no âmbito IoT*, somente no âmbito de agentes móveis.*

4.1 Motivação

Em concordância com a discussão apresentada no Capítulo 1, a real motivação deste trabalho foi encontrar uma forma de melhorar o processo de tratamento de exceções sensível ao contexto voltado para dispositivos da *IoT*. A melhoria foi feita de forma a sistematizar o processo que antes era feito de forma *adhoc* por desenvolvedores de software, através da implementação de novas abstrações, a proposta e validação de uma arquitetura genérica para realização do tratamento de exceções sensível ao contexto. Esses dispositivos estão cada vez mais presentes no nosso cotidiano, dessa forma, uma abordagem para que se possa realizar o tratamento de exceções sensível ao contexto é necessária como já foi mostrado nos capítulos anteriores. Explorar essa lacuna pode melhorar significativamente vários domínios do ambiente humano, pelo fato da *IoT* estar nos mais diversos ambientes e os desenvolvedores utilizarem do mecanismo para tratar as exceções contextuais desses ambientes.

4.2 Objetivo

Este trabalho propõe um Mecanismo de Tratamento de Exceções Sensível ao Contexto[1] aplicado à Internet de Coisas. A partir do levantamento bibliográfico do Capítulo 3 foi constatado que não haviam trabalhos que lidam formalmente e de forma explícita do tratamento de exceções sensível ao contexto no âmbito da Internet das Coisas. Até a escrita do trabalho de Damasceno e Cacho (2006) não haviam sequer trabalhos na literatura que provinham o suporte para tratamento de exceções sensível ao contexto para aplicações móveis robustas (TRIPATHI et al., 2001). O que pode ser encontrado na literatura são abordagens "*adhoc*" de implementação do tratamento de exceções sensível ao contexto por parte dos desenvolvedores de software, que podem não conter todas as abstrações mencionadas anteriormente. Sendo assim o principal desafio desse trabalho é caracterização de novas abstrações que possibilitem:

- A definição e o tratamento de exceções contextuais
- A propagação das exceções contextuais
- A recuperação de erros sensível ao contexto
- A transição (tráfego) das informações contextuais pelos dispositivos

Assim, seguindo esses desafios, este trabalho vêm para agregar ao estado da arte do tratamento de exceções sensível ao contexto em um âmbito mais recente e mais amplo que é o da Internet das Coisas mediante à definição de um mecanismo (arquitetura genérica) para tal. A validação do mecanismo de tratamento de exceções proposto foi realizada por intermédio do desenvolvimento de uma simulação da aplicação prática (estudo de caso) *UbiParking* (ROCHA, 2013) que explora e avalia seu uso na construção de sistemas assíncronos sensíveis ao contexto.

4.3 Middleware

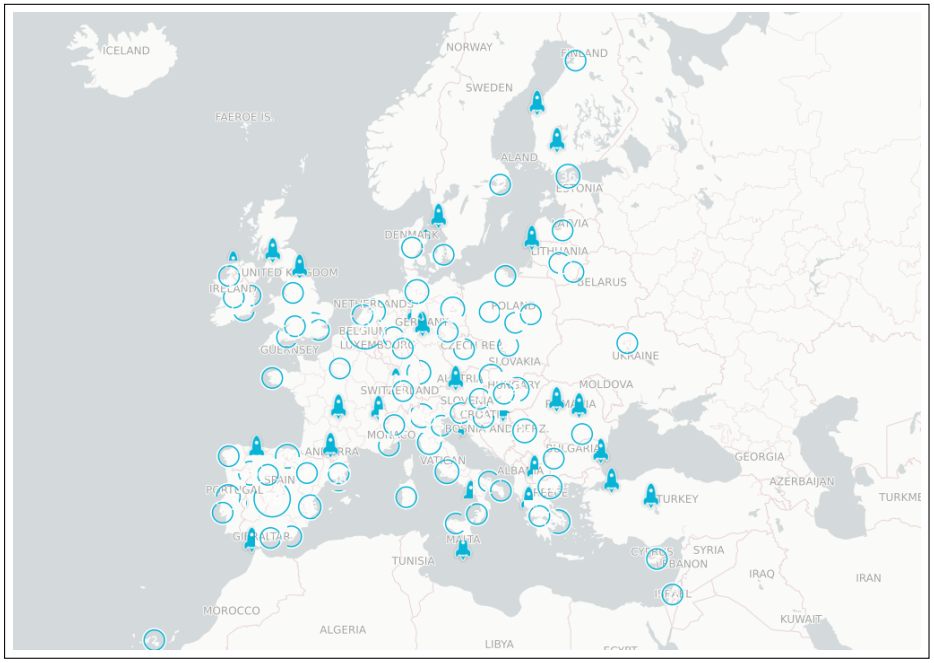
Conforme discutido na Seção 2.2.1, para que haja a comunicação entre os dispositivos da *IoT* e que seja feito o tratamento de exceções sensível ao contexto, a corrente pesquisa fará utilização de um *middleware publish/subscribe*, mais especificamente o *middleware* chamado *FIWARE*. Esse *middleware* faz parte da Comunidade *FIWARE* que é

[1] Em inglês: Context-aware Exception Handling (CAEH).

uma comunidade *open source* os quais membros estão engajados na missão de construir um ecossistema sustentável aberto em torno de padrões de plataforma de softwares públicos, sem *royalties* e orientados à implementação, que facilitarão o desenvolvimento de novos aplicativos inteligentes em vários setores. Para isso foi criado a plataforma *FIWARE* com as seguintes características:

- *APIs* abertas tornando o acesso fácil das aplicações aos recursos da plataforma;
- Faz o uso inteligente dos dados, com *API* padrão para gerenciamento e troca de modelos de dados harmônicos;
- Conexão e coleta de dados de dispositivos;
- Automação de processos em toda a cadeia de valor. Fácil integração *plug and play* com outras soluções e serviços. Parte de um mercado de soluções portáteis e interoperáveis;
- Suporte da comunidade fornecendo recursos compartilhados e validando as tecnologias do FIWARE;
- Mais de 100 cidades (Figura 4.1), 11 *iHubs*, vários programas de aceleração e parcerias estratégicas com a GSMA, TM Forum, CEF e ETSI, entre outros;

Figura 4.1: Localização das pequenas e médias empresas que utilizam o FIWARE



Fonte: *http://map.fiware.org/actors/smes*

- Utilização do *Ultralight* 2.0, que é um protocolo baseado em texto leve destinado a dispositivos e comunicações restritos onde a largura de banda e a memória do dispositivo podem ser recursos limitados;

- Módulos em forma de *docker containers*, que facilitam o início da utilização do *middleware* de forma que a configuração pode ser feita até mesmo através de um comando.

- Utilização de uma especificação de gerenciamento de contexto chamada *NGSI* definida pela *Open Mobile Alliance*(OMA) [2]. Essa especificação diz a respeito da utilização de abstrações como:

  - Entidades, que são a representação virtual de todos tipos de objetos físicos, como exemplo, mesas, cadeiras, salas ou até mesmo pessoas;
  - Atributos, que representam qualquer informação sobre as entidades físicas, como por exemplo, a temperatura corporal de uma pessoa pode ser expressa através de um atributo "temperaturaCorporal";
  - Domínio de atributo, que agrupa um conjunto de atributos, como por exemplo um "statusSaude" que agrupa "temparaturaCorporal" e "pressaoSanguinea";
  - Elemento de contexto, que é a estrutura de dados utilizada para a troca de informações entre as entidades. Ela pode conter informações de vários atributos de uma única entidade;

Por essas características, principalmente a de que as *APIs* são abertas e ser o único *middleware* encontrado que possuí as abstrações claras necessárias para lidar com as informações de contexto, que o *FIWARE* foi adotado como *middleware publish/subscribe* o qual faz a comunicação com os dispositivos da *IoT* na arquitetura do mecanismo. Tais abstrações como entidades, atributos, domínio de atributos e elemento de contexto fazem grande parte das atividades no tratamento de exceções sensível ao contexto. Para realizar a configuração dos dispositivos no FIWARE através dessas abstrações do *middleware*, são realizadas chamadas do *HTTP* do tipo *POST* como é exibido na Figura 4.1. As instruções de como essas abstrações podem ser implementadas fazem parte da diretriz que será apresentada posteriormente na Seção 4.6. Além disso a plataforma possui uma vasta documentação, o que facilitou muito o seu aprendizado e a tarefa de projetar o *CAEH*.

[2] http://www.openmobilealliance.org/wp/

```
curl -iX POST \
  'http://iot-agent:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
"devices": [
        {
         "device_id":   "spacesensor001",
         "entity_name": "urn:ngsi-ld:SpaceSensor:001",
         "entity_type": "Spacesensor",
         "protocol":    "PDI-IoTA-UltraLight",
         "timezone":    "Europe/Berlin",
         "attributes": [
           { "object_id": "s", "name": "statusParkSpace", "type": "Integer" }
         ],
          "static_attributes": [
              { "name":"refPark", "type": "Relationship", "value":
                  ↪ "urn:ngsi-ld:Park:001"}
          ]
        }
    ]
}'
```

Código 4.1: Chamada *POST* para configuração do dispositivo no *middleware* (Elaborado pelo autor)

4.4 Uma Arquitetura de Software para Tratamento de Exceções

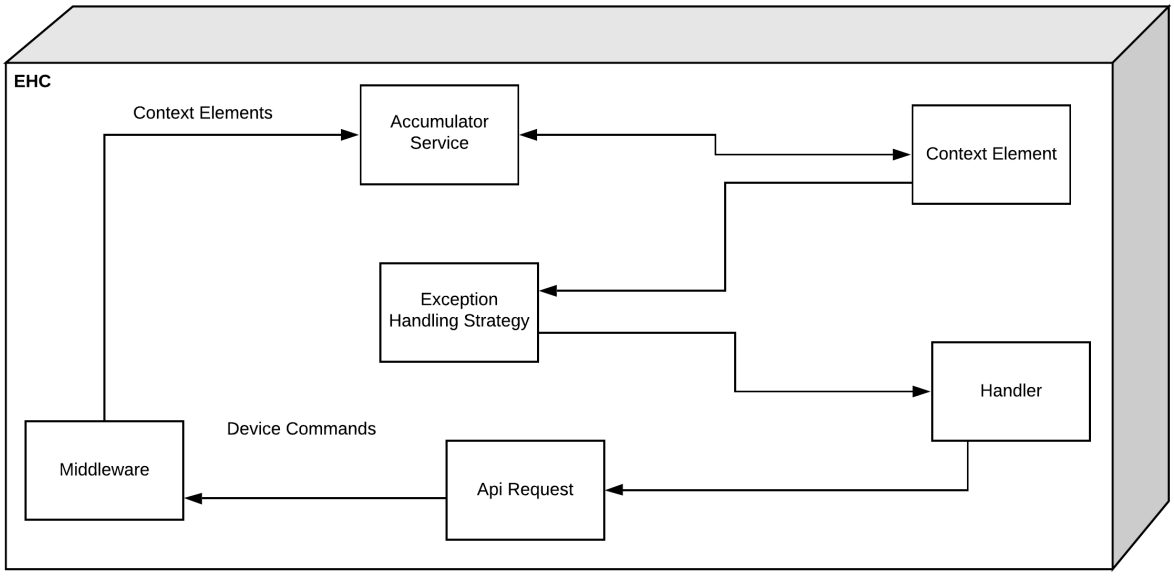
A arquitetura em alto nível do mecanismo proposto é apresentada na Figura 4.2. Essa arquitetura foi elaborada de forma que possa ser implementada utilizando qualquer linguagem de programação que apresente suporte ao paradigma orientado a objetos. Além disso, a arquitetura supõe a utilização de um *middleware* do paradigma *publish/subscribe*. Ela foi baseada nos trabalhos citados no Capítulo 3, que levantam os conceitos e modelos mais importantes para o tema.

A arquitetura genérica, denominada *Context-aware Exception Handling Architecture (CAEHA)*, é composta pelos seguintes componentes:

- O componente denominado *AccumulatorService* é o núcleo do mecanismo, pois é através dele que o *middleware* fará o contato para repassar as informações contextuais advindas dos dispositivos *IoT* independentemente se a informação é de interesse para o dispositivo e que possa causar que seja levantada uma exceção ou não. Esse componente também será capaz de realizar as instanciações das outras classes, para que as chamadas aos métodos possam ser realizadas.

- O componente *ContextElement* é o responsável pela transformação da informação de contexto em objeto, podendo assim ser utilizado de forma encapsulada e mais en-

Figura 4.2: Arquitetura do Mecanismo (Elaborado pelo autor).



xuta. Além disso, cada um dos componentes de elemento de contexto é responsável pela sua própria verificação da exceção associada.

- Os componentes *Handlers* são os responsáveis pela especificação do tratamento de cada uma das exceções levantadas nos escopos local, grupo e região. Além disso ele fará o repasse do status do tratamento naquele determinado escopo. Isso significa que se o tratamento falhar para um determinado escopo, este componente será o responsável por repassar essa informação do escopo em específico.

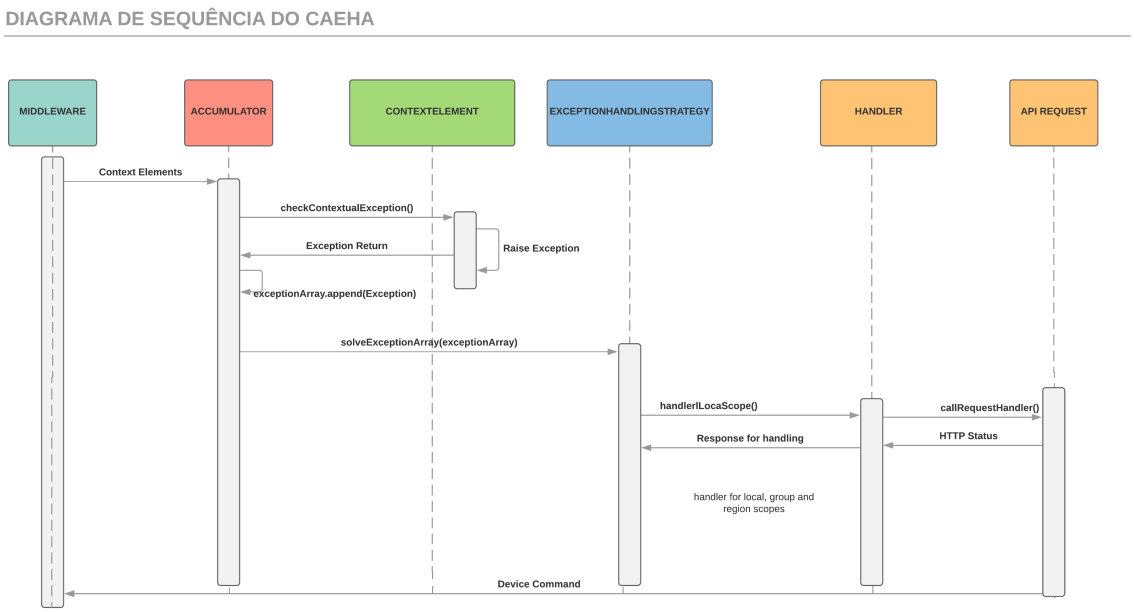
- O componente *ExceptionHandlingStrategy* é o responsável por gerenciar as exceções e seus tratadores e especificar uma estratégia de busca dos tratadores por ordem de prioridade de escopo. Além disso esse componente será responsável também por validar se o tratamento da exceção foi realizado com sucesso. Se a exceção não for tratada, o componente irá verificar através do código de retorno que o tratamento não foi efetuado com sucesso e irá levantar a exceção para um escopo superior.

- O componente *ApiRequest* é o responsável por fazer as chamadas ao *middleware* tanto para consulta de dispositivos, quanto para a comunicação com os dispositivos atuadores para que sejam enviados os comandos.

A sequência em que esses componentes se comunicam é apresentada no diagrama de sequência da Figura 4.3. É possível explicar de forma resumida o fluxo das informações. Primeiramente o componente `Middleware` envia as informações contextuais para o

componente `AccumulatorService` que por sua vez repassa essas informações para o componente `ContextElement` que as verifica e possivelmente levanta as exceções contextuais associadas. O componente `AccumulatorService` captura a exceções levantadas e após um determinado período, as exceções são adicionadas a uma fila e enviadas para o componente `ExceptionHandlingStrategy`, que por sua vez aciona os tratadores (componentes `Handler`) em cada escopo da aplicação. Por fim, os componentes `Handlers` possivelmente fazem requisições para o componente `ApiRequest` passando as informações do dispositivo atuador que será acionado durante a realização do tratamento de exceção contextual.

Figura 4.3: Diagrama de sequência do *CAEHA* (Elaborado pelo autor)



4.5 Implementação do Mecanismo Proposto

Essa seção apresenta a implementação dos componentes discutidos na Seção 4.4 e presentes na arquitetura do modelo de tratamento de exceções proposto (Figura 4.2).

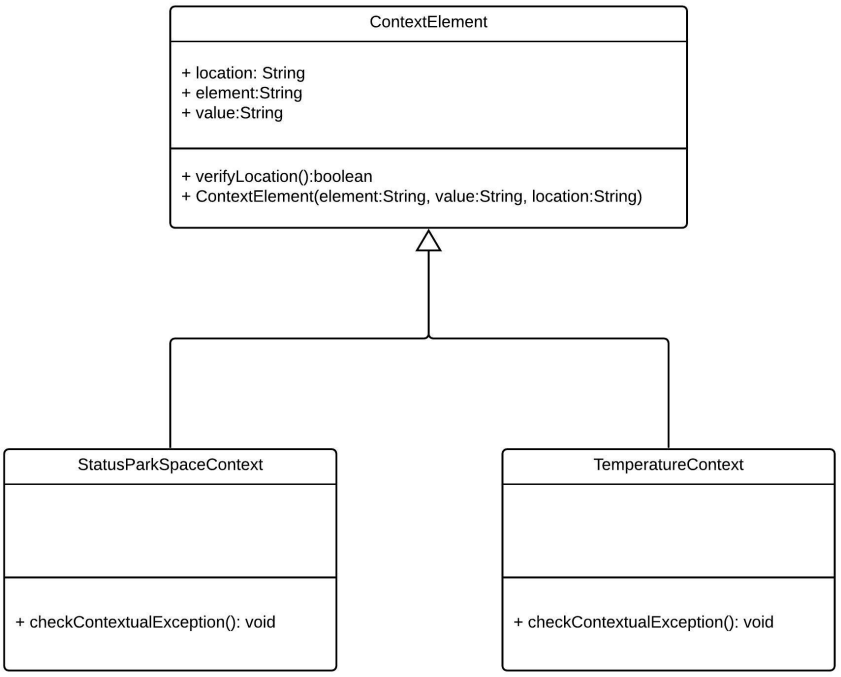
Apesar do modelo de tratamento de exceções proposto ter sido projetado de tal forma a ser possível a implementação utilizando qualquer linguagem de programação orientada a objetos, adotou-se a linguagem Python.

Python oferece suporte para introspecção de objeto. Uma linguagem de programação introspectiva permite consultar atributos e métodos de um objeto e seus tipos e valores (DOUCET; SHUKLA; GUPTA, 2003). Esse conceito é crucial pois foi utilizado, por exemplo, na definição e invocação aos diferentes escopos (local, grupo e região)

de tratamento de exceções contextuais. Por intermédio da classe utilitária denominada `stringToClassHelper`, a aplicação faz a chamada de métodos a partir de uma *String* passada, e isso é crucial para o acesso aos `handlers` conforme será discutido posteriormente.

O componente `Accumulator Service` é um serviço web que foi implementado utilizando *Flask* – pequeno *framework* escrito em Python que provê um modelo simples para o desenvolvimento *web*. O componente `Accumulator Service` disponibiliza o *endpoint* `accumulate` que recebe as requisições enviadas pelo *middleware* com as informações contextuais dos sensores/dispositivos IoT. Esse componente a partir das informações contextuais recebidas realiza a instanciação da classe (subclasse da classe `ContextElement`) mais adequada.

Figura 4.4: Diagrama de classe: `ContextElement` e subclasses (Elaborado pelo autor)

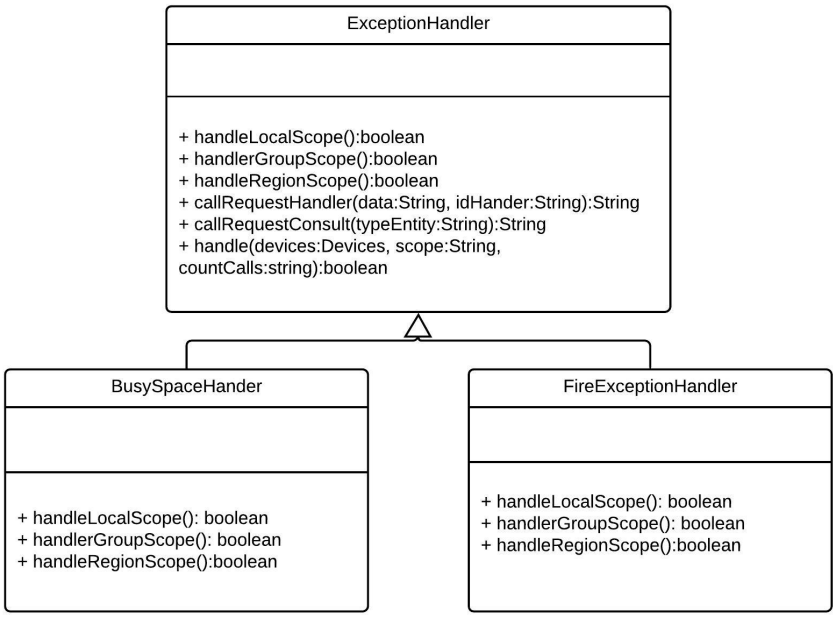


Para exemplificar o funcionamento dessa abordagem, a Figura 4.4 mostra a hierarquia de subclasses da classe `ContextElement`. Por exemplo, para um elemento de contexto cadastrado como temperatura no *middleware*, temos a classe `TemperatureContext` que implementa um método chamado `checkContextualException` que é responsável por verificar se a temperatura (informação contextual) recebida pode ser considerada uma exceção contextual. A classe `TemperatureContext` por herdar da classe `ContextElement` possui os atributos `location`, `element` e `value`. Assim que ocorrer a verificação da infor-

mação contextual e esta for considerada uma situação de contexto excepcional, a exceção equivalente será lançada (Python `raise`). No caso da classe `TemperatureContext` será lançada a exceção `FireException`.

A partir do momento em que as exceções são levantadas, elas irão ser capturadas pelo componente `AccumulatorService` e serão inseridas em uma fila. Esse vetor será enviado a cada `X` segundos para o componente `ExceptionHandlingStrategy`. A configuração da quantidade `X` de segundos utilizada encontra-se em um arquivo de configuração e pode ser facilmente modificada para atender os requisitos dos desenvolvedores do sistema. Após esse processo, o componente `ExceptionHandlingStrategy` irá instanciar o componente `Handler` mais adequado para tratar a exceção. Assim como os elementos de contexto, os `handlers` também herdam de uma classe mãe, conforme é apresentado na Figura 4.5. Assim uma `FireException` irá possuir o seu `FireExceptionHandler` com os métodos de escopo associados. Conforme pode-se observar, os escopos (local, grupo e região) de tratamento das exceções contextuais são métodos implementados pelas subclasses da classe `Handler`.

Figura 4.5: Diagrama de classe: `Handler` e subclasses (Elaborado pelo autor)



Após a instanciação do `handler` de cada exceção levantada, o *ExceptionHandlingStrategy* realiza, por intermédio do conceito de introspecção, o acesso aos métodos de cada um dos escopos do tratador. Primeiramente irá tentar chamar o método `handleLocalScope()`, que se refere ao tratamento local, que será responsável por exemplo, por enviar comandos para o dispositivo atuador responsável por lidar com esse escopo.

Em cada um dos escopos será realizada a instanciação da classe `Devices`, tal classe recebe o tipo dos dispositivos atuadores aos quais deseja-se realizar o tratamento, o `id` desses dispositivos e por último o comando para cada um desses dispositivos aos quais se está chamando.

Após esse processo, cada escopo realiza a chamada para o método `handle` que pertence a classe mãe `ExceptionHandler` passando o objeto criado a partir da classe `Devices`, o escopo atual de tratamento e um contador para registro de chamadas. O método `handle` por sua vez realiza a ação mediante a invocação do método denominado `callRequestHandler` passando a ação do dispositivo atuador e o `id` do dispositivo, que por sua vez irá realizar uma invocação ao método `send` do componente `Api Request`. Esse método retorna um *status HTTP*. Desse modo, se o dispositivo retornar qualquer `status` diferente de 200 que representa o *status* OK, o `ExceptionHandlingStrategy` irá acionar os outros escopos que se encontram implementados no tratador por intermédio dos métodos `handleGroupScope` e `handleRegionScope` responsáveis pelo tratamento do escopo de grupo e região respectivamente. Esse processo de chamada dos escopos será realizado até o momento que a exceção contextual seja tratada com sucesso ou até que o escopo mais genérico seja atingido. A chamada que o componente `Api Request` realiza para acionar os dispositivos atuadores também é por intermédio dos *endpoints* que o *middleware* disponibiliza, isso demonstra a importância do mesmo para o modelo e a necessidade da pesquisa dos melhores *middlewares* para tal implementação.

Finalizando a discussão apresentada nessa seção, a Tabela 4.1 apresenta uma visão geral da implementação de cada um dos componentes do modelo proposto.

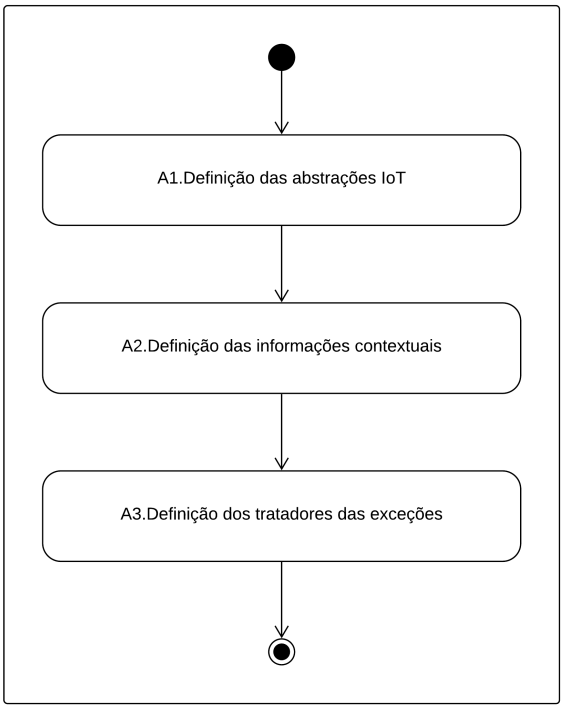
Tabela 4.1: Componentes do modelo e suas implementações (Elaborado pelo Autor)

Componente genérico	Implementação
`Middleware Publish/Subscribe`	*Middleware* FIWARE
`AccumulatorService`	Flask *web service*
`ContextElement`	Classe mãe de todos os elementos de contexto
`ExceptionHandlingStrategy`	Classe com a lógica de priorização de escopos de tratamento de exceções
`ExceptionHandler`	Classe mãe de todos os tratadores de exceções contextuais
`ApiRequest`	Classe auxiliar com a logica de invocação/chamada dos *endpoints*

4.6 Diretrizes - Visão Geral

Foi definido um método para a apresentação das diretrizes. Tal método consiste de atividades que podem ser formadas por um conjunto de tarefas interconectadas, onde as saídas de uma tarefa funcionam como entradas das tarefas seguintes. Assim para ter uma visão geral do método que funciona como um sumário para as diretrizes, apresentamos a sequência das atividades (Figura 4.6) representadas por um diagrama de atividades. As tarefas para cada uma das atividades serão apresentadas a seguir.

Figura 4.6: Atividades do Método (Elaborado pelo autor)



4.6.1 Atividade A1. Definição das abstrações *IoT*

O objetivo desta atividade é definir as abstrações do cenário de tratamento de exceções. Essas abstrações consistem das entidades, atributos, domínio de atributos, elementos de contexto para que sejam definidos os dispositivos *IoT* e as informações contextuais serão transitadas na comunicação entre os dispositivos. Além disso, irá ser realizada a definição das abstrações no arquivo que será utilizado pelo *middleware* para a configuração dos dispositivos. O arquivo de configuração é do tipo `BASH`, que é um tipo de *Shell* sendo assim considerada uma linguagem de *scripting*. As configurações serão realizadas através de novos comandos no arquivo de configuração que serão chamadas `curl` (ferramenta que realiza chamadas para *APIs*) do tipo `HTTP POST` ao *orion*, essas chamadas para os *endpoints* podem ser acessadas através de qualquer cliente ou diretamente através de comandos, apesar disso os *endpoints* possuem características especificas do *middleware FIWARE*. O *orion*, como explicado nos capítulos anteriores faz o gerenciamento de todos os elementos do FIWARE inclusive das entidades. Todas as configurações irão conter três *headers* representados na chamada *POST* através do -H:

- "Context-Type: application/json": Representa que o tipo de conteúdo que está sendo enviado e encontra-se no formato *JavaScript Object Notation* (JSON).
- "fiware-service: openiot": Representa o nome do serviço usado pelo internamente pelo *middleware*.
- "fiware-servicepath:" Representa um caminho de *urls* de chamadas para as entidades.

4.6.1.1 Tarefa T1. Definir a entidade de localização

Para que seja realizado o tratamento, é necessário especificar uma entidade que será o local físico em que os dispositivos estão inseridos. Essa é a primeira definição a ser feita, pois os dispositivos terão um relacionamento com o local. Por exemplo, para o cenário em que temos a automação de uma academia inteligente, o local pode ser a própria academia em si. O Código 4.2 mostra a configuração de uma localização genérica.

O JSON de conteúdo da localização deverá conter:

- "id": "valor único atribuído a localização".
- "type": "tipo da localização".

```
curl -X POST \
'http://orion:1026/v2/op/update' \
-H 'Context-Type: application/json' \
-H 'fiware-service: openiot' \
-H 'fiware-servicepath: /' \
-g -d '{
  "actionType": "APPEND",
  "entities": [
    {
      "id": "urn:ngsi-ld:GENERICLOCATION:001", "type": "GENERICLOCATION"
    }
  ]
}'
```

Código 4.2: Definição da configuração da localização (Elaborado pelo autor)

4.6.1.2 Tarefa T2. Definir o grupo de dispositivos

Outra abstração a ser definida é o grupo de dispositivos. Um grupo de dispositivos como o próprio nome já explica, se refere a um conjunto de dispositivos com características similares. Nesse exemplo citado na tarefa anterior da academia poderíamos ter um grupo chamado de "aparelhos", contendo todos aparelhos *smart* da academia. O Código 4.3 mostra a configuração de um grupo de dispositivos.

```
curl -iX POST \
  'http://iot-agent:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    "/iot/d"
   }
 ]
}'
```

Código 4.3: Definição da configuração do *service* de grupo de dispositivos (Elaborado pelo autor)

O JSON deve conter um vetor chamado *services* as seguintes chaves/valores:

- "apikey": "chave utilizada pelos dispositivos que irão pertencer ao grupo"
- "cbroker": "define o *context broker* que será utilizado, no caso o *orion* que fará o gerenciamento das entidades"
- "entity_type": "tipo de entidade utilizada"
- "resource": "*url* caminho para o service"

4.6.1.3 Tarefa T3. Definir os dispositivos e as informações contextuais

A abstração mais importante é a definição do dispositivos *IoT* que serão os responsáveis pela captação das informações contextuais. No exemplo da academia os dispositivos podem ser esteiras, elípticos e até mesmo estações de musculação. Sendo que as esteiras podem enviar a informação da velocidade atual do dispositivo, e a estação pode enviar a informação contextual da força aplicada pelo usuário do aparelho em suas barras. O Código 4.4 mostra a configuração do dispositivo e das informações contextuais.

```
curl -iX POST \
  'http://iot-agent:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
"devices": [
    {
     "device_id":   "genericdevice001",
     "entity_name": "urn:ngsi-ld:Genericdevice:001",
     "entity_type": "Genericdevice",
     "protocol":    "PDI-IoTA-UltraLight",
     "timezone":    "Europe/Berlin",
     "attributes": [
       { "object_id": "s", "name": "genericattribute", "type": "Float" }
     ],
      "static_attributes": [
          { "name":"refPark", "type": "Relationship", "value":
              ↪ "urn:ngsi-ld:GENERICLOCATION:001"}
      ]
    }
  ]
}'
```

Código 4.4: Definição da configuração da localização (Elaborado pelo autor)

4.6.1.4 Tarefa T4. Configurar a inscrição das informações contextuais

Para que os dispositivos façam o envio das informações contextuais é necessário realizar a inscrição de que atributos pertencentes as entidades serão enviados para o *endpoint* do `AcumulatorService`. O Código 4.5 mostra a configuração de inscrição das informações contextuais.

```
curl -X POST \
  --url 'http://orion:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  --header 'fiware-service: openiot' \
  --header 'fiware-servicepath: /' \
  --data '{
  "description": "Subcription of generic device",
  "subject": {
    "entities": [
      {
        "idPattern": ".*",
        "type": "GenericDevice"
      }
    ],
    "condition": {
      "attrs": [
        "genericattribute"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://IP:1028/accumulate"
    },
    "attrs": [
      "genericattribute",
      "refLocation"
    ]
  },
  "throttling": 5
}'
```

Código 4.5: Definição da configuração de inscrição das informações contextuais (Elaborado pelo autor)

- "description": "descrição da inscrição"
- "subject": "entities": ["idPattern": "padrão de devices", "type": "tipo de *devices*"], "condition": "attrs": "atributo do dispositivo"
- "notification": "http": "url": "ip ao qual o accumulator está rodando", "attrs": ["os atributos que serão enviados"]

4.6.1.5 Tarefa T5. Definir as constantes utilizadas pelo *CAEHA*

O *CAEHA* exige a definição de constantes que funcionam como uma configuração central. Cada constante será responsável por uma funcionalidade específica e muito importante para o recebimento de informações contextuais, definição de uma exceção, tratamento da exceção e a propagação da exceção.

Um exemplo genérico do arquivo de constantes pode ser visto no Código 4.6.

```
CONTEXTELEMENTS = ['genericattribute']
GENERICEXCEPTIONVALUE = 5
IDLOCATION1 = 'urn:ngsi-ld:GENERICLOCATION:001'
IDLOCATION2 = 'urn:ngsi-ld:GENERICLOCATION:001'
NAMELOCATION = 'refLocation'
# The header to connect the middleware API
HEADER = {
    'Content-Type': 'application/json',
    'fiware-service': 'openiot',
    'fiware-servicepath': '/'
}
HEADERGET = {
    'fiware-service': 'openiot',
    'fiware-servicepath': '/'
}

TIMEWAITINGEXCEPTIONS = 5
```

Código 4.6: Definição das constantes utilizadas pelo CAEHA (Elaborado pelo autor)

- "CONTEXTELEMENTS": ["atributos do *middleware* que serão traduzidos como elementos contextuais"]
- "GENERICEXCEPTIONVALUE": "valor de constante para que um determinado atributo seja considerado uma exceção, no caso o atributo genericattribute"
- "header e headerget": "headers de configuração do *middleware*"
- "TIMEWAITINGEXCEPTIONS": "tempo de espera por exceções"

4.6.2 Definição das informações contextuais (exceções contextuais)

Como mostra o Código 4.7, o elemento de contexto tem que estender da classe `ContextElement` se tornando assim filha dela. Cada elemento de contexto será uma classe que irá realizar a checagem do valor comparado com o valor da constante declarada. Além dessa checagem, será verificado na classe `ContextElement` o valor da localização em que a informação contextual foi recebida é igual ao valor da localização configurada. Outras verificações pode ser realizadas para que seja levantada a exceção, por exemplo pode ser feito o consumo de outro serviço *web* que possui uma informação específica e crucial para que seja levantada a exceção desejada.

```
class GenericContext(ContextElement):
  def checkContextualException(self):
      print("Checking Contextual Exception...")
      if self.value == const.STATUSPARKSPACE and self.verifyLocation:
          print('Raising GenericException')
          raise ContextualException("Generic")
```

Código 4.7: Definição de elemento de contexto utilizado pelo CAEHA (Elaborado pelo autor)

4.6.3 Definição dos tratadores das exceções

O tratador é uma classe que deve estender da classe `ExceptionHandler`. Ele deve conter três métodos, o `handlerLocalScope` que irá lidar com o tratamento do escopo local como exibido no Código 4.8, o `handlerGroupScope` que irá lidar com o tratamento do escopo de grupo como é exibido no Código 4.9. Por último o `handleRegionScope` irá lidar com o tratamento do escopo de região, como mostra o Código 4.10. Na implementação de um método de escopo deve conter as chamadas para os dispositivos que irão realizar o tratamento e o método deve retornar um *boolean* se o tratamento foi realizada com sucesso ou não. Cada escopo deve realizar a chamada para o dispositivo tratador correspondente, sendo assim, essa parte do código é a que o desenvolvedor possui uma liberdade maior para realizar o tratamento efetivo.

```
    def handleLocalScope(self, countCalls):
      self.countCalls = countCalls
      devices = []
      genericDevice = Devices('Genericdevice', ['001'], ['right'])
      devices.append(genericDevice)

      return self.handle(devices, 'Local', self.countCalls)
```

Código 4.8: Definição de tratador genérico de escopo local utilizado pelo CAEHA (Elaborado pelo autor)

```
    def handleGroupScope(self):
        genericSensors = self.callRequestConsult(
            'Genericsensor', const.IDLOCATION1)

        for sensor in genericSensors:
            if(sensor['statusParkSpace'] != const.STATUSPARKSPACE):
                print(
                    'Call the microservice to the get the shortest path
                        to the sensor with id = ' + sensor['id'])

        devices = []
        genericDevices = Devices(
            'Genericdevice', ['002', '003'], ['right', 'left'])

        devices.append(genericDevices)

        return self.handle(devices, 'Group', self.countCalls)
```

Código 4.9: Definição de tratador genérico de escopo de grupo utilizado pelo CAEHA (Elaborado pelo autor)

```
    def handleRegionScope(self):
        locations = [const.IDLOCATION1, const.IDLOCATION2]
        for location in locations:
            genericSensors = self.callRequestConsult('Genericsensor',
                location)
            for sensor in genericSensors:
                if(sensor['statusParkSpace'] != const.STATUSPARKSPACE):
                    print(
                        'Call the microservice to the get the shortest path
                            to the sensor with id = ' + sensor['id'])

        devices = []
        genericDevices = Devices('Genericdeivce', ['004', '005', '006'], [
                                 'right', 'left', 'right'])
        devices.append(genericDevices)
        return self.handle(devices, 'Region', self.countCalls)
```

Código 4.10: Definição de tratador genérico de escopo de região utilizado pelo CAEHA (Elaborado pelo autor)

4.6.4 Considerações Finais

O *FIWARE* foi utilizado na implementação do mecanismo para o sistema apresentado na Seção 2.3.4, o estacionamento ubíquo chamado *UbiParking*. Tal sistema, é utilizado como estudo de caso para a avaliação do mecanismo proposto, utilizando os critérios mostrados no objetivo da corrente pesquisa. Foram avaliados as facilidades de se implementar o mecanismo no caso do estacionamento, e também as principais dificuldades de sua utilização durante as etapas de implementação. Ao final foi obtido o resultado para a viabilidade de se utilizar o mecanismo para a implementação do *CAEH* voltado para dispositivos da *IoT*. Além disso, foi disponibilizado o código fonte[3] do mecanismo na plataforma GitLab junto ao *middleware* e seus procedimentos de forma que outros desenvolvedores possam validar a utilização do mesmo em outros cenários de estudo de caso.

[3] <https://gitlab.com/gabrielhfgomes/mecanismo_de_tratamento_de_excecoes_iot>

Capítulo 5

Estudo de caso

Este capítulo apresenta a aplicação sensível ao contexto utilizada como estudo de caso para avaliação do mecanismo proposto. Ubiparking foi desenvolvida por Rocha (2013), tal estudo de caso modela um estacionamento ubíquo.

5.1 UbiParking

O Ubiparking é a aplicação que foi apresentada no Capítulo 2 e será utilizada para exemplificar o uso da arquitetura em conjunto com o *middleware FIWARE*. A aplicação realiza o controle de um estacionamento e tem como objectivo controlar as exceções contextuais de incêndio e vaga preenchida do estacionamento que será modelado.

5.1.1 Tratadores e Propagação de exceções

Com intuito de demonstrar um exemplo prático das exceções contextuais, foram criadas as exceções de incêndio e exceção de vaga ocupada. Para cada uma dessas exceções foi projetado um tratador respectivo. Assim, para que a exceção seja capturada é necessário que algumas etapas sejam cumpridas, como por exemplo o registro do dispositivo IoT no *middleware* e a subscrição aos atributos do mesmo. No caso das exceções contextuais de incêndio (`FireException`) e vaga ocupada (`BusySpaceException`), os dispositivos IoT relacionados são:

- Sensor de temperatura (`TemperatureSensor`)
- Sensor de presença (`ParkSpaceSensor`)
- Aspersor (`Sprinkler`)

- Alarme sonoro (`SoundAlarm`)
- Painel de informações (`LightPanel`)
- Telefone (`Phone`)

Podemos classificar os dispositivos *IoT* em duas categorias, a primeira é relacionada a dispositivos sensores que captam dados como o `ParkSpaceSensor` e `TemperatureSensor`. E a outra seria relacionada a dispositivos atuadores que fazem um tratamento efetivo como o `SoundAlarm`, `LightPanel` e o `Sprinkler`. Para que uma exceção contextual de incêndio seja levantada, uma condição do atributo do dispositivo tem que ser satisfeita, neste caso a expressão para isso ocorra seria "temperatura > 40" relativa ao `TemperatureSensor`. Essa informação é enviada do dispositivo associado ao *middleware*, que através dos seus processos redireciona a informação contextual ao *EHC* que por sua vez faz a verificação através da expressão. Essa expressão apesar de parecer simples, tem uma grande importância para o Ubiparking, pois dessa forma a exceção de incêndio será tratada pelo `FireExceptionHandler` no primeiro escopo chamado local, este por sua vez irá localizar o `Sprinkler` mais próximo do `TemperatureSensor` e irá realizar uma chamada para sua *API* disponibilizada através do *middleware*. Se o status associado a resposta for o *status* 201 significa que o dispositivo foi contactado e realizou o tratamento adequadamente. Caso a execução do mecanismo não tenha sucesso, o mecanismo irá levantar a exceção para o escopo superior e continuar o tratamento até que o escopo mais genérico seja atingido.

A exceção intitulada de `BusySpaceException` é mais complexa que a anterior e modela o caso em que um usuário se encontra dentro de um veículo e reservou uma vaga dentro do estacionamento através de um dispositivo celular ou tablet. A partir dessa ação ele foi direcionado para a vaga mais próxima dentro do estacionamento.

O sistema do dispositivo que possui uma integração com o *EHC*, faz o processo de reserva da vaga 301, por exemplo, para o usuário que estava logado no sistema e dessa forma, irá indicar no dispositivo o caminho para o estacionamento e a vaga. No cenário criado, o usuário estava a caminho da vaga quando outro usuário ocupou a mesma. Quando o sensor de presença `ParkSpaceSensor` que se encontra fisicamente naquela vaga detectar que ouve uma mudança de seu status, o *middleware* irá receber a alteração do atributo `statusParkSpace` que modela os dois status da vaga, ocupada e desocupada. Neste momento o sistema checa a expressão "statusParkSpace == 1", a qual significa que a vaga está ocupada e faz uma chamada para o sistema do dispositivo do usuário

para verificar quais usuários estão associados à aquela vaga. Se o sistema constatar que existe mais de um usuário associado a vaga, o sistema irá retornar que o tratamento deve ser efetivado e dessa forma junto da informação que a vaga já foi ocupada a exceção `BusySpaceException` será levantada.

5.1.2 Tarefas e atividades do Ubiparking

Na Seção 4.6 foi apresentado o conjunto de diretrizes genéricas para a implementação do mecanismo proposto. Para essa seção serão seguidos os mesmos passos que consiste de tarefas e atividades para que seja realizado o tratamento do estudo de caso do *Ubiparking* para a exceção de vaga ocupada e para a exceção de incêndio.

5.1.3 Atividade A1. Definição das abstrações IoT

5.1.3.1 Tarefa T1. Definir a entidade de localização

Os *HEADERS* da chamada que será feita, fazem parte das configurações citadas do *middleware* para o cenário. A entidade que será utilizada é chamada de estacionamento ("Park"), essa entidade irá ser referenciada posteriormente durante a criação dos outros dispositivos como um atributo de localização, e vai ser criado com "id:urn:ngsi-ld:Park:001". Essa referência, tem como objetivo identificar o estacionamento utilizando do padrão *NGSI* como explicado anteriormente. A configuração no arquivo pode ser vista no Código 5.1. A seção de diretrizes a qual essa implementação foi baseada é a Seção 4.6.1.1.

```
curl -X POST \
  'http://orion:1026/v2/op/update' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
  "actionType": "APPEND",
  "entities": [
    {
        "id":"urn:ngsi-ld:Park:001","type":"Park"
    }
  ]
}'
```

Código 5.1: Definição da configuração da localização (Elaborado pelo autor)

5.1.3.2 Tarefa T2. Definir o grupo de dispositivos

O grupo será do tipo “Thing”. O nome é somente uma forma de identificar o grupo. A seção de diretrizes a qual essa implementação foi baseada é a Seção 4.6.1.2.

O Código 5.2 exibe o *service* padrão que irá conter o tipo de entidade denominado "Thing".

```
curl -iX POST \
  'http://iot-agent:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    "/iot/d"
   }
 ]
}'
```

Código 5.2: Definição da configuração do service do grupo de dispositivos (Elaborado pelo autor)

5.1.3.3 Tarefa T3. Definir os dispositivos e as informações contextuais

Os dispositivos que serão utilizados foram citados anteriormente na subseção 5.1.1. Cada dispositivo possui como principal característica os atributos, que no caso dos dispositivos sensores são atributos que seguem o protocolo *Ultralight* contendo um identificador (“object_id”), um nome (“name”) e um tipo (“type”). No caso dos dispositivos atuadores, possui um nome (“name”) e um comando (“command”). A seção de diretrizes a qual essa implementação foi baseada é a Seção 4.6.1.3.

O Código 5.3 mostra a criação do sensor de temperatura e os atributos necessários para tal. O principal atributo é o atributo *Ultralight* “t” com nome de “temperature” do tipo “Float”, que irá representar o envio do dado de temperatura. Aqui será utilizado como atributo estático referente a localização que foi criado anteriormente chamada “refPark” com o valor de `id` criado anteriormente no Código 5.1.

```
curl -iX POST \
  'http://iot-agent:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
 "devices": [
    {
      "device_id":   "thermometer001",
      "entity_name": "urn:ngsi-ld:Thermometer:001",
      "entity_type": "Thermometer",
      "protocol":    "PDI-IoTA-UltraLight",
      "timezone":    "Europe/Berlin",
      "attributes": [
        { "object_id": "t", "name": "temperature", "type": "Float" }
      ],
      "static_attributes": [
        { "name":"refPark", "type": "Relationship", "value": "urn:ngsi-ld:Park:001"}
      ]
    }
 ]
}'
```

Código 5.3: Definição da configuração dispositivo sensor de temperatura (Elaborado pelo autor)

O Código 5.4 mostra a criação do sensor de presença e os atributos necessários para tal, tendo como principal o atributo *Ultralight* "s" que com nome de "statusParkSpace" do tipo "Integer" com o valor de status se a vaga está preenchida ou vazia. Assim como no Código 5.3, a criação do dispositivo atual faz referência a localização do estacionamento através do `id`.

```
curl -iX POST \
  'http://iot-agent:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
 "devices": [
    {
     "device_id":   "spacesensor001",
     "entity_name": "urn:ngsi-ld:SpaceSensor:001",
     "entity_type": "Spacesensor",
     "protocol":    "PDI-IoTA-UltraLight",
     "timezone":    "Europe/Berlin",
     "attributes": [
       { "object_id": "s", "name": "statusParkSpace", "type": "Integer" }
     ],
      "static_attributes": [
          { "name":"refPark", "type": "Relationship", "value":
              ↪ "urn:ngsi-ld:Park:001"}
      ]
    }
 ]
}'
```

Código 5.4: Definição da configuração dispositivo sensor de presença (Elaborado pelo autor)

O Código 5.5 mostra a criação do painel de informações e os atributos necessários para tal, tendo como principal o atributo *"side"* representando o comando com o lado pelo qual o dispositivo irá acender a seta luminosa. Assim como os outros, este dispositivo faz referencia do estacionamento através do `id`.

```
curl -iX POST \
  'http://iot-agent:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
 "devices": [
    {
      "device_id": "lightpanel001",
      "entity_name": "urn:ngsi-ld:LightPanel:001",
      "entity_type": "LightPanel",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://192.168.15.14:1029/lightpanel/001",
      "timezone": "America/Sao_Paulo",
      "commands": [
        { "name": "right", "type": "command" },
        { "name": "left", "type": "command" }

      ],
       "static_attributes": [
         {"name":"refPark", "type": "Relationship","value": "urn:ngsi-ld:Park:001"}
      ]
    }
 ]
}'
```

Código 5.5: Definição da configuração do *service* do painel de informações (Elaborado pelo autor)

O Código 5.6 mostra a criação do aspersor e dos atributos necessários para tal, tendo como principal o atributo de comandos chamados *"start"* e *"stop"* que representa os comandos para o dispositivo ligar ou desligar para exercer sua função de liberar água. Assim como os outros, este dispositivo faz referencia do estacionamento através do `id`.

```
curl -iX POST \
  'http://iot-agent:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
 "devices": [
    {
      "device_id": "sprinkler001",
      "entity_name": "urn:ngsi-ld:Sprinkler:001",
      "entity_type": "Sprinkler",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://192.168.15.14:1029/sprinkler/001",
      "timezone": "America/Sao_Paulo",
      "commands": [
        { "name": "on", "type": "command" },
        { "name": "off", "type": "command" }

      ],
       "static_attributes": [
         {"name":"refPark", "type": "Relationship","value": "urn:ngsi-ld:Park:001"}
      ]
    }
 ]
}'
```

Código 5.6: Definição da configuração do dispositivo aspersor (Elaborado pelo autor)

O Código 5.7 mostra a criação do alarme sonoro e os atributos necessários para tal, tendo como principal os comandos *"on"* e *"off'* para ligar e desligar o som do alarme. Assim como os outros, este dispositivo faz referencia do estacionamento através do `id`.

```
curl -iX POST \
  'http://iot-agent:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -g -d '{
 "devices": [
    {
      "device_id": "soundAlarm001",
      "entity_name": "urn:ngsi-ld:SoundAlarm:001",
      "entity_type": "SoundAlarm",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://192.168.15.14:1029/soundAlarm/001",
      "timezone": "America/Sao_Paulo",
      "commands": [
        { "name": "on", "type": "command" },
        { "name": "off", "type": "command" }

      ],
       "static_attributes": [
         {"name":"refPark", "type": "Relationship","value": "urn:ngsi-ld:Park:001"}
      ]
    }
 ]
}'
```

Código 5.7: Definição da configuração do dispositivo de alarme sonoro (Elaborado pelo autor)

5.1.3.4 Tarefa T4. Definir a inscrição das informações contextuais

Conforme explicado na Seção 4.6.1.4, um passo importante é realizar a inscrição das informações contextuais que serão enviadas ao `AccumulatorService`.

O Código 5.8 mostra essa inscrição, para o dispositivo de termômetro que fará o envio da temperatura. A chave "description" representa a descrição da inscrição, que serve somente para identificação da regra para o desenvolvedor. Dentro de "subject" que é o assunto, temos as "entities" que são as entidades e a "condition" condição. Os tipos de entidades que podem ser inseridos são aqueles cadastrados anteriormente que contém a chave "`entity_type`" como é exibido no Código 5.3. Podemos ter a subscrição de múltiplas entidades, sendo assim é um vetor, que contém o "idPattern" com o padrão que no caso vai ser geral representado pelo "*" para todos os dispositivos do tipo "Thermometer". A condição por sua vez possui "attrs" atributos, no caso será o atributo "temperature" que representa a temperatura. A chave "notification" é para onde será enviado o dado quando o evento for disparado com aquela condição explicada anteriormente, no caso será enviado por meio do protocolo "http", com a "url" identificada. Os "attrs" atributos que serão enviados para a "url" vão ser os "temperatura" e "refPark".

```
curl -X POST \
  --url 'http://orion:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  --header 'fiware-service: openiot' \
  --header 'fiware-servicepath: /' \
  --data '{
  "description": "One subscription to rule them all",
  "subject": {
    "entities": [
      {
        "idPattern": ".*",
        "type": "Thermometer"
      }
    ],
    "condition": {
      "attrs": [
        "temperature"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://192.168.15.14:1028/accumulate"
    },
    "attrs": [
      "temperature",
      "refPark"
    ]
  },
  "throttling": 5
}'
```

Código 5.8: Definição da inscrição do sensor de temperatura (Elaborado pelo autor)

O Código 5.9 mostra a inscrição para o dispositivo de sensor de presença que fará o envio se existe algo naquele espaço, ou não, representado pela variável `statusParkSpace`. O processo de inscrição é mesmo feito anteriormente só que para outro tipo de dado de contexto e dispositivo, no caso o dado de status da vaga e que será captado através do sensor de presença.

```
curl -X POST \
  --url 'http://orion:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  --header 'fiware-service: openiot' \
  --header 'fiware-servicepath: /' \
  --data '{
  "description": "One subscription to rule them all",
  "subject": {
    "entities": [
      {
        "idPattern": ".*",
        "type": "Spacesensor"
      }
    ],
    "condition": {
      "attrs": [
        "statusParkSpace"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://192.168.15.14:1028/accumulate"
    },
    "attrs": [
      "statusParkSpace",
      "refPark"
    ]
  },
  "throttling": 5
}'
```

Código 5.9: Definição da inscrição do sensor de presença (Elaborado pelo autor)

O Código 5.10 mostra a inscrição para o dispositivo painel que informará o lado a ser tomado através dos atributos `left, right`.

```
curl -iX POST \
  'http://orion:1026/v2/registrations' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "description": "Commands Registrations",
  "dataProvided": {
    "entities": [
      {
        "id": "urn:ngsi-ld:LightPanel:001","type": "LightPanel"
      }
    ],
    "attrs": ["right", "left"]
  },
  "provider": {
    "http": {"url": "http://orion:1026/v1"},
    "legacyForwarding": true
  }
}'
```

Código 5.10: Definição da inscrição do painel de informações (Elaborado pelo autor)

O Código 5.11 mostra a inscrição para o dispositivo aspersor que possui os comandos iniciar e parar que são os atributos `on, off`.

```
curl -iX POST \
  'http://orion:1026/v2/registrations' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "description": "Commands Registrations",
  "dataProvided": {
    "entities": [
      {
        "id": "urn:ngsi-ld:Splinkler:001","type": "Splinker"
      }
    ],
    "attrs": ["on", "off"]
  },
  "provider": {
    "http": {"url": "http://orion:1026/v1"},
    "legacyForwarding": true
  }
}'
```

Código 5.11: Definição da inscrição do aspersor (Elaborado pelo autor)

Por último, o Código 5.12 mostra a inscrição para alarme sonoro que pode estar ligado ou desligado, o qual é representado através dos atributos `on, off`.

5.1.3.5 Tarefa T5. Definir as constantes utilizadas pelo CAEHA

Conforme explicado na Seção 4.6.1.5 as constantes do Código 5.13 foram definidas com base nos elementos de contexto planejados anteriormente. Para o cenário atual existe um vetor contendo os elementos de contexto (*"CONTEXTELEMENTS"*) de temperatura

```
curl -iX POST \
  'http://orion:1026/v2/registrations' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "description": "Commands Registrations",
  "dataProvided": {
    "entities": [
        {
            "id": "urn:ngsi-ld:SoundAlarm:001","type": "SoundAlarm"
        }
    ],
    "attrs": ["on", "off"]
  },
  "provider": {
    "http": {"url": "http://orion:1026/v1"},
    "legacyForwarding": true
  }
}'
```

Código 5.12: Definição da inscrição do alarme sonoro (Elaborado pelo autor)

(*"temperature"*) e estado da vaga (*"statusParkSpace"*). Os limiares para que esses contextos sejam considerados com excepcionais é visto através da constante de valor de exceção de incêndio (*"FIREEXCEPTIONVALUE"*) e estado da vaga (*"STATUSPARKSPACE"*). A constante *"IDLOCATION"* é utilizada para definir o identificador (*id*) da localização e a *"NAMELOCATION"* o nome da localização. Já as constante *"HEADER"* e *"HEADER-GET"* são constantes de configuração do *middleware* para o cenário atual. Por último a *"TIMEWAITINGEXCEPTIONS"* representa o tempo do componente aguardando as exceções para que sejam tratadas.

```
CONTEXTELEMENTS = ['temperature', 'statusParkSpace']
FIREEXCEPTIONVALUE = 50
STATUSPARKSPACE = '1'
IDLOCATION1 = 'urn:ngsi-ld:Park:001'
IDLOCATION2 = 'urn:ngsi-ld:Park:002'
NAMELOCATION = 'refPark'
# The header to connect the middleware API
HEADER = {
    'Content-Type': 'application/json',
    'fiware-service': 'openiot',
    'fiware-servicepath': '/'
}
HEADERGET = {
    'fiware-service': 'openiot',
    'fiware-servicepath': '/'
}

TIMEWAITINGEXCEPTIONS = 0
```

Código 5.13: Definição das constantes do Ubiparking (Elaborado pelo autor)

5.1.4 Atividade A2. Definição das informações contextuais

Os elementos de contexto do Código 5.14 foram definidos com base nas abstrações definidas anteriormente e conforme a Seção 4.6.2 de diretrizes.

```
class StatusParkSpaceContext(ContextElement):
    def checkContextualException(self):
        print("Checking Contextual Exception")
        if (self.value == const.STATUSPARKSPACE) and self.
            verifyLocation and (not self.hasAvailableParkSpaces()):
            raise ContextualException("BusySpace")

    def hasAvailableParkSpaces(self):
        parkSpaces = self.callRequestConsult(
            'Spacesensor', const.IDLOCATION1)

        for parkSpace in parkSpaces:
            if(parkSpace['statusParkSpace'] != const.STATUSPARKSPACE):
                return True

        return False
```

Código 5.14: Definição do elemento de contexto de status de vaga (Elaborado pelo autor)

5.1.5 Atividade A3. Definição dos tratadores das exceções

5.1.5.1 Tratadores da exceção de vaga ocupada

Os métodos que são apresentação nessa seção fazem parte da classe responsável pelo tratamento da exceção de vaga ocupada, a classe se chama `BusySpaceHandler`.Essa classe

deve estender da class `ExceptionHandler` como pode ser visto no Código 5.15.

```
class BusySpaceHandler(ExceptionHandler):
```

Código 5.15: Declaração da classe BusySpaceHandler (Elaborado pelo autor)

O Código 5.16 mostra o tratamento no escopo local onde deve ser feita uma ação somente em um dispositivo, que no caso poderia ser um local onde sempre irão haver vagas disponíveis. É realizada a chamada para o *light panel* com a direção já pré inserida e avaliada a sua resposta. Essa atividade foi baseada na Seção 4.6.3 de diretrizes.

```
def handleLocalScope(self, countCalls):
    self.countCalls = countCalls
    devices = []
    lightPanels = Devices('LightPanel', ['001'], ['right'])
    devices.append(lightPanels)

    return self.handle(devices, 'Local', self.countCalls)
```

Código 5.16: Definição do tratador da exceção de vaga ocupada para o escopo local (Elaborado pelo autor)

O Código 5.17 mostra como são encontradas as vagas que são representadas pelo dispositivo "SpaceSensor". Após encontradas as vagas, elas são checadas se possuem status como desocupadas, isso baseado na constante "STATUSPARKSPACE". Se a vaga não está ocupada, então poderá ser realizada uma chamada para outro serviço que encontra os "light panels", este por sua vez encontra o que pode compor o menor caminho para que o veículo chegue a vaga e as direções adequadas para tal. É realizada uma iteração sobre o retorno e as devidas chamadas são realizadas para esses "light panels" mostrem o caminho até a vaga livre.

```
    # Search in the park if there is a empty park space(this could be a
        call for the Web Service logic of the parking.
    spaceSensors = self.callRequestConsult(
        'Spacesensor', const.IDLOCATION1)
    # Take the sensors and check the one that is avaiable
    # Call webservice for the smallest routes that return the panels
    for sensor in spaceSensors:
        if(sensor['statusParkSpace'] != const.STATUSPARKSPACE):
            print(
                'Call the microservice to the get the shortest path to the
                    sensor with id = ' + sensor['id'])

    devices = []
    lightPanels = Devices('LightPanel', ['002', '003'], ['right', 'left'])

    devices.append(lightPanels)

    return self.handle(devices, 'Group', self.countCalls)
    # Call various lightpanels inside the park to the closest exit to go to
        the other park
```

Código 5.17: Definição do tratador da exceção de vaga ocupada para o escopo de grupo (Elaborado pelo autor)

O último escopo de tratamento mostrado no Código 5.18, o de região verifica as localizações que são o “Park1” e o “Park2” e baseado nessas localizações checa os sensores que possuem vaga livre. Dessa forma se um estacionamento tiver todas as vagas ocupadas, o veiculo será levado para o outro estacionamento que se encontra na região.

```
    # Search in all region1 parks for a empty park space
    # Call various lightpanels inside the park to the closest exit to go to
        the other park
    locations = [const.IDLOCATION1, const.IDLOCATION2]

    for location in locations:
        spaceSensors = self.callRequestConsult('Spacesensor', location)
        for sensor in spaceSensors:
            if(sensor['statusParkSpace'] != const.STATUSPARKSPACE):
                print(
                    'Call the microservice to the get the shortest path to
                       the sensor with id = ' + sensor['id'])

    devices = []
    lightPanels = Devices('LightPanel', ['004', '005', '006'], [
                          'right', 'left', 'right'])
    devices.append(lightPanels)

    return self.handle(devices, 'Region', self.countCalls)
```

Código 5.18: Definição do tratador da exceção de vaga ocupada para o escopo de região (Elaborado pelo autor)

5.1.5.2 Tratadores da exceção de incêndio

Os métodos que são apresentação nessa seção fazem parte da classe responsável pelo tratamento da exceção de incêndio, a classe se chama `FireHandler`. Essa classe deve estender da class `ExceptionHandler` como pode ser visto no Código 5.19.

```
    class FireHandler(ExceptionHandler):
```

Código 5.19: Declaração da classe FireHandler (Elaborado pelo autor)

O tratamento do escopo local pode ser visto no Código 5.20. Ele é realizado através da chamada de somente um único dispositivo específico aonde foi detectado ou aonde se encontra um ponto de risco. Nesse caso é feita a chamada para o dispositivo aspersor que prontamente é acionado para jogar aguá no possível incêndio.

```
    def handleLocalScope(self, countCalls):
        self.countCalls = countCalls
        devices = []
        sprinklers = Devices('Sprinkler', ['001'], ['on'])
        devices.append(sprinklers)
        return self.handle(devices, 'Local', self.countCalls)
```

Código 5.20: Definição do tratador da exceção de incêndio para o escopo local (Elaborado pelo autor)

O Código 5.21 mostra o tratamento do escopo do grupo de forma a chamar múltiplos dispositivos aonde foi detectado ou aonde se encontra um ponto de risco. Nesse caso é feita a chamada para os dispositivos desse grupo que são os vários aspersores e realizada um iteração sobre a resposta desses dispositivos para verificar se o tratamento em todos foi realizado com sucesso.

```
    def handleGroupScope(self):
        devices = []
        sprinklers = Devices('Sprinkler', ['001', '002'], ['on', 'on'])
        soundAlarm = Devices('SoundAlarm', ['001'], ['on'])
        devices.append(sprinklers)

        return self.handle(devices, 'Group', self.countCalls)
```

Código 5.21: Definição do tratador da exceção de incêndio para o escopo de grupo (Elaborado pelo autor)

O tratamento do escopo de região que é exibido no Código 5.22 é realizado de forma a chamar múltiplos dispositivo da região do Park1, esses dispositivos são os alarmes sonoros, que avisam as pessoas para saírem do estacionamento. Após isso é feita a iteração para avaliar a resposta de todos os dispositivos e verificar se o tratamento foi bem sucedido.

```
    def handleRegionScope(self):
        idsSoundAlarms = self.callRequestConsult(
            'SoundAlarm', const.IDLOCATION1, 'id')
        idPhone = self.callRequestConsult(
            'SoundAlarm', const.IDLOCATION1, 'id')

        devices = []
        soundAlarm = Devices('SoundAlarm', idsSoundAlarms, ['on'])
        phone = Devices('Phone', idPhone, ['call'])

        devices.append(soundAlarm)
        devices.append(phone)
        return self.handle(devices, 'Region', self.countCalls)
```

Código 5.22: Definição do tratador da exceção de incêndio para o escopo de região (Elaborado pelo autor)

5.1.6 Simulação do Estudo de Caso

Para realizar a simulação do estudo de caso, foi criada uma aplicação `python` utilizando o `framework Flask` que simula tanto os dispositivos sensores, quanto os dispositivos atuadores. A aplicação possui dois *endpoints* um para cada tipo de dispositivo, sendo que ambos são *endpoints* que aceitam requisições do tipo `POST`.

O *endpoint* que simula os dispositivos atuadores se chama `simulateActuators` e recebe dois parâmetros como mostra o Código 5.23. O primeiro equivale ao tipo de dispositivo e o segundo ao `id` do dispositivo que foi criado no *middleware*. Dessa forma, é gerado um booleano aleatório (verdadeiro ou falso) e se o valor dele for verdadeiro, então o *endpoint* irá ter uma resposta `HTTP 200`, já se o valor for falso, irá retornar `HTTP 500`. Isso simula a resposta dos dispositivos atuadores e se foram capazes ou não de efetuar a ação desejada para que o tratamento seja efetuado.

```
@app.route("/<device>/<id>", methods=['POST'])
def simulateActuators(device, id):
    rand = bool(random.getrandbits(1))
    now = datetime.now()
    file = open(device+":"+id+".txt", "a+")

    if rand:
        stringToWrite = str(now)+' '+device+' SUCCESS '+'\n'
        file.write(stringToWrite)
        file.close()
        return Response(status=200)
    else:
        stringToWrite = str(now)+' '+device+' ERROR '+'\n'
        file.write(stringToWrite)
        file.close()
        return Response(status=500)
```

Código 5.23: Definição do *endpoint* que simula os dispositivos atuadores (Elaborado pelo autor)

O segundo *endpoint* se chama `simulateSensors` e simula os dispositivos sensores. Tal *endpoint* recebe três parâmetros, o primeiro é o `id` do dispositivo sensor, o segundo é o parâmetro `ultralight` e o terceiro é o número de requisições que serão simuladas. O *endpoint* realiza as chamadas para o *middleware* através do parâmetro `ultralight` simulando dados de temperatura para o caso do dispositivo termômetro e dados *booleanos* para o caso do dispositivo da vaga, como mostra o Código 5.24.

```
    @app.route("/sensors/<idSensor>/<ultraLightAttribute>/<
        numberOfSubmissions>", methods=['POST'])
def simulateSensors(idSensor, ultraLightAttribute, numberOfSubmissions):
    urlDevice1 = 'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&
        i='+idSensor
    header = {
      'Content-Type':'text/plain'
    }

    numberOfSubmissions = int(numberOfSubmissions)
    while numberOfSubmissions > 0:
        sleep(12)
        rand = random.getrandbits(1)
        if(ultraLightAttribute == 't'):
            if(str(rand) == '0'):
                data = ultraLightAttribute+'|51'
            else:
                data = ultraLightAttribute+'|40'
        else:
            data = ultraLightAttribute+'|'+str(rand)
        print(data)
        requests.post(urlDevice1, headers=header, data=data)
        numberOfSubmissions -= 1

    return Response(status=200)

def initialize():
    app.run(host=host, port=port)

if __name__ == '__main__':
    initialize()
```

Código 5.24: Definição do *endpoint* que simula os dispositivos sensores (Elaborado pelo autor)

5.1.6.1 Contexto Excepcional de Incêndio

Para realizar a simulação do contexto excepcional de incêndio, será realizada a chamada `POST` para os dispositivos sensores com a URL <http://localhost:1029/sensors/thermometer001/t/100>. Essa *URL* possui os parâmetros que foram citados anteriormente e são essenciais para que a requisição seja feita com sucesso. Com essa chamada, serão realizadas 100 requisições `POST` com temperaturas geradas aleatoriamente dentro do método e será simulado o fluxo do *middleware* até o tratamento especificado nas seções anteriores através dos dispositivos atuadores. O mecanismo gera um conjunto de arquivos mostrando o escopo o qual cada contexto excepcional foi tratado. A partir desses arquivos é possível contar quantas requisições que simulam os sensores geraram contexto excepcionais, e dessas que geraram quantas foram tratadas em cada um dos escopos como é mostrado na Tabela 5.1.

Tabela 5.1: Requisições e escopos de tratamento das exceções do contexto de incêndio o qual as requisições foram tratadas ou não

Requisições e Escopos					
Requisições	**Requisições Excepcionais**	**Local**	**Grupo**	**Região**	**Não Tratadas**
100	47	30	3	4	10
1000	544	398	74	43	29

A tabela mostra que o maior número de casos são tratados no escopo local. Isso se deve pelo fato de o escopo local dos cenário excepcional de incêndio possuir somente um dispositivo, dessa forma, quando a exceção for levantada haverá uma chance de 50% da exceção conseguir ser tratada no escopo e 50% do tratamento não conseguir ser tratado no escopo local. Para os escopos de grupo e região acontece justamente o contrário, pela fato de que esses dispositivos possuem múltiplos dispositivos, todos os dispositivos necessários para a realização do tratamento necessitam possuir um retorno `HTTP 200` significando que o dispositivo conseguiu atuar. Se algum dispositivo retornar `HTTP 500` o escopo não irá conseguir tratar e a exceção será levantada para o próximo escopo. Ao final, se a exceção não puder ser tratada nos escopos mencionados, poderá ser criado um escopo genérico dependente da regra de negócio da aplicação como medida paliativa para tentar tratar a exceção.

5.1.6.2 Contexto Excepcional Vaga Ocupada

Para realizar a simulação do contexto excepcional de vaga ocupada no estacionamento, é necessário que todas a vagas que estavam livres no estacionamento estejam preenchidas, assim, como foram simulados 3 dispositivos sensores de preenchimento de vaga no estacionamento, temos que preencher até a ultima vaga para que o contexto seja considerado excepcional e dessa forma desencadear um possível tratamento. As URL de requisição para cada um dos dispositivos será <http://localhost:1029/sensors/spacesensor001/s/2>, <http://localhost:1029/sensors/spacesensor002/s/2> e <http://localhost:1029/sensors/spacesensor003/s/10>. A Tabela 5.2 mostra a quantidade de requisições excepcionais e o tratamento delas em cada escopo.

A quantidade de dispositivos empregada pela exceção de vaga ocupada foi a mesma da quantidade de dispositivos da exceção de incêndio, por esse fato, como já explicado anteriormente as quantidades de exceções tratadas em cada escopo serão similares a exceção de incêndio.

Tabela 5.2: Requisições e escopos de tratamento das exceções do contexto de vagas ocupadas o qual as requisições foram tratadas ou não

Requisições e Escopos					
Requisições	**Requisições Excepcionais**	**Local**	**Grupo**	**Região**	**Não Tratadas**
100	54	29	7	3	15
1000	512	397	69	19	27

5.1.7 Considerações Finais

Com a utilização das diretrizes e com os conhecimentos passados pelo estudo de caso, é possível realizar o tratamento de qualquer cenário excepcional que for necessário. Os conhecimentos passados possibilitam realizar a configuração do *middleware* de forma a atender os requisitos da pesquisa, para que as informações contextuais sejam enviadas e tratadas pelo mecanismo de acordo as implementações dos desenvolvedores.

Capítulo 6

Conclusões

Neste capítulo são apresentadas as conclusões do trabalho. Primeiramente serão apresentadas as principais discussões do trabalho. Posteriormente serão apresentados as contribuições do trabalho além das propostas para continuação do mesmo em trabalhos futuros.

6.1 Discussões finais

Mecanismos especializados de tratamento de exceções tem sido consistentemente uma questão central em Engenharia de Software de tal forma a promover melhor confiabilidade no desenvolvimento de sistemas de software (PERRY; ROMANOVSKY; TRIPATHI, 2000). Com a crescente utilização de dispositivos IoT e aplicações ubíquas, uma série de desafios em desenvolvimento de software tem emergido. Tais sistemas precisam satisfazer requisitos rigorosos relacionados à tolerância a falhas, integridade e disponibilidade. Nesse contexto, com o objetivo de superar os desafios inerentes, este trabalho apresenta a proposta de um Mecanismo de Tratamento de Exceções Sensível ao Contexto aplicado à Internet de Coisas que caracteriza novas abstrações que possibilitam: (i) a definição e o tratamento de exceções contextuais; (ii) a propagação das exceções contextuais; (iii) a recuperação de erros sensível ao contexto e (iv) a transição (tráfego) das informações contextuais pelos dispositivos.

Similarmente ao trabalho de Damasceno e Cacho (2006) o mecanismo proposto, por intermédio da definição de uma arquitetura genérica, auxilia no processo de desenvolvimento de sistemas confiáveis por abstrair questões inerentes ao tratamento de exceções contextuais. Por exemplo, utilizando o mecanismo proposto, desenvolvedores não necessitam se preocupar em definir uma estratégia para propagação do erros, ou mesmo pensar

quais seriam os diferentes escopos adequados para sua aplicação. Porém, em contraste ao trabalho de Damasceno e Cacho (2006), este trabalho estende o conceito de tratamento de exceções sensível ao contexto para o âmbito da Internet das Coisas, que era antes tratado apenas no contexto de agentes móveis.

Com a utilização de um *middleware* que pode ser utilizado para negócios de grande escala possuindo grande numero de requisições advindas de muitos dispositivos IoT, o tratamento de exceções pode ser efetuado em diversas áreas do domínio humano, podendo assim contribuir muito para a melhoria dos mais diversos ambientes onde os dispositivos IoT se encontram.

Como trabalho futuro, pretendemos evoluir o mecanismo de exceções proposto, provendo suporte à resolução de exceções contextuais concorrentes. Várias exceções lançadas concomitantemente podem ocorrer em uma aplicação sensível ao contexto, o que realmente pode significar a ocorrência de uma situação excepcional mais grave. Assim, o mecanismo de tratamento de exceções deve ser capaz de recolher todas as ocorrências de exceções concorrentes e resolvê-las de modo que uma exceção mais adequada seja sinalizada e tratada. Além disso, este trabalho foi validado com um estudo de caso com dispositivos *IoT* simulados, e como trabalho futuro seria interessante também validar em um estudo de caso com dispositivos *IoT* reais. Por fim, consideramos que a utilização do paradigma *publish-subscribe* e do *middleware FIWARE* se mostrou como uma boa alternativa para a implementação do mecanismo de tratamento de exceções. Entretanto, pode ser interessante utilizar outros paradigmas e sistemas de *middleware* especialmente para ampliar o modelo de tratamento de exceções.

Referências

ASHTON, K. That 'internet of things' thing. *RFID journal*, v. 22, n. 7, p. 11–97, 2009.

ASQUITH, B. j. Supporting Fault Tolerance in the Internet of Things. p. 228, 2017. Disponível em: <https://cloudfront.escholarship.org/dist/prd/content/qt39s3k7bw/qt39s3k7bw.pdf>.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, v. 1, n. 1, p. 11–33, 2004. ISSN 1545-5971. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1335465&tag=1>.

BARBOSA, J. L. V. D. F. C. Context-oriented exception handling Jorge Luis Victória Barbosa. v. 2, n. 1, p. 16–25, 2009.

BATISTA, T. V. Aplicações para Cidades Inteligentes Requisitos e Plataformas de Middleware para Cidades Inteligentes. 2016.

BAZZANI, M.; CONZON, D.; SCALERA, A.; SPIRITO, M. A.; TRAINITO, C. I. Enabling the IoT paradigm in E-health solutions through the VIRTUS middleware. *Proc. of the 11th IEEE Int. Conference on Trust, Security and Privacy in Computing and Communications, TrustCom-2012 - 11th IEEE Int. Conference on Ubiquitous Computing and Communications, IUCC-2012*, p. 1954–1959, 2012. ISSN 2324-898X.

BEDER, D. M.; ARAUJO, R. B. D. Towards the definition of a Context-Aware Exception Handling Mechanism. In: *Proceedings - 5th Latin-American Symposium on Dependable Computing Workshops, LADCW 2011*. São Carlos, SP: IEE Computer Society, 2011. p. 25–28. ISBN 9780769543949.

CAMPBELL, R. H.; RANDELL, B. Error Recovery in Asynchronous Systems. *IEEE Transactions on Software Engineering*, SE-12, n. 8, p. 811–826, 1986. ISSN 00985589.

CAPRA, L.; EMMERICH, W.; MASCOLO, C. CARISMA: Context-Aware Reflective middleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, v. 29, n. 10, p. 929–945, 2003. ISSN 00985589.

CHANG, T.; DUAS, S.; MELING, H. P2S : A Fault-Tolerant Publish / Subscribe Infrastructure Categories and Subject Descriptors. *Debs*, p. 189–197, 2014.

CHO, E. S.; CHOI, J. H.; HELAL, S. Dynamic parameter filling for semantic exceptions in context-aware systems. *Proceedings - IEEE 10th International Conference on Ubiquitous Intelligence and Computing, UIC 2013 and IEEE 10th International Conference on Autonomic and Trusted Computing, ATC 2013*, p. 293–300, 2013.

CHO, E. S.; HELAL, S. Expressive exceptions for safe pervasive spaces. *Journal of Information Processing Systems*, v. 8, n. 2, p. 279–300, 2012. ISSN 1976913X.

CHO, E. S.; HELAL, S. Toward efficient detection of semantic exceptions in context-aware systems. *Proceedings - IEEE 9th International Conference on Ubiquitous Intelligence and Computing and IEEE 9th International Conference on Autonomic and Trusted Computing, UIC-ATC 2012*, p. 826–831, 2012.

DAMASCENO, K.; CACHO, N. Tratamento de exceções sensível ao contexto. *Anais do XX Simpósio ...*, n. July, p. 49–64, 2006. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbes/2006/004.pdf>.

DAMASCENO, K.; CACHO, N.; GARCIA, A.; ROMANOVSKY, A.; LUCENA, C. Context-aware exception handling in mobile agent systems: The MoCA Case. *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems - SELMAS '06*, p. 37, 2006. Disponível em: <http://portal.acm.org/citation.cfm?doid=1138063.1138071>.

DEY, A. K.; ABOWD, G. D. Towards a Better Understanding of Context and Context-Awareness. *Computing Systems*, v. 40, n. 3, p. 304–307, 1999. ISSN 00219266.

DOUCET, F.; SHUKLA, S.; GUPTA, R. Introspection in system-level language frameworks: Meta-level vs. integrated. *Proceedings -Design, Automation and Test in Europe, DATE*, p. 382–387, 2003. ISSN 15301591.

DUMITRU, R. L. IoT Platforms: Analysis for Building Projects. *Informatica Economica*, v. 21, n. 2/2017, p. 44–53, 2017. ISSN 14531305. Disponível em: <http://revistaie.ase.ro/content/82/04rusu.pdf>.

FILHO, C. A. Q.; ANDRADE, R. M. C.; ROCHA, L. S.; BRAGA, R. B.; OLIVEIRA, C. T. ConExT-U: A context-aware exception handling mechanism for task-based ubiquitous systems. *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014*, p. 127–132, 2014.

GADDAH, A.; KUNZ, T. A survey of middleware paradigms for mobile computing. *Techical Report*, n. July, 2003. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.7633&rep=rep1&type=pdf>.

GUTIERREZ-GARCIA, J. O.; RAMOS-CORCHADO, F. F. Exception Handling in Pervasive Service Composition Using Normative Agents. *Journal of Web Engineering*, v. 10, n. 3, p. 175–196, 2011. ISSN 15409589. Disponível em: <http://dl.acm.org/citation.cfm?id=2230837>.

KIM, M.; KIM, M. Handling exceptions in situation-driven agent systems. *NCM 2009 - 5th International Joint Conference on INC, IMS, and IDC*, p. 870–875, 2009.

KNUDSEN, J. L. Better Exception- Handling in Block- Structured Systems. n. 40, p. 40–49, 1987.

KRISHNAMURTHY, J.; MAHESWARAN, M. *Internet of Things Principles and Paradigms*. [S.l.: s.n.], 2016. 79–102 p. ISSN 10745351. ISBN 9780128093474.

KULKARNI, D.; TRIPATHI, A. Application-level recovery mechanisms for context-aware pervasive computing. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, p. 13–22, 2008. ISSN 10609857.

LEE, P.; ANDERSON, T. *Fault Tolerance*. [S.l.]: Springer, 1990. ISBN 9783709189924.

LIMA, R. UMA LINGUAGEM PARA MODELAGEM E VERIFICAÇÃO DO TRATAMENTO DE EXCEÇÃO SENSÍVEL AO CONTEXTO EM SISTEMAS UBÍQUOS. 2013.

LOPES, F.; CACHO, N.; BATISTA, T. Um Mecanismo de Composição de Eventos para Resolução de Exceções Sensíveis ao Contexto. *... de Engenharia de Software (SBES'07 ...*, p. 182–198, 2007. Disponível em: <http://www.lbd.dcc.ufmg.br:8080/colecoes/sbes/2007/SBES11.pdf>.

LOPES, F. A. d. S. *CES – Um Mecanismo Genérico de Composição de Eventos para Sistemas Sensíveis ao Contexto*. 80 p. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, 2008. Disponível em: <http://ftp.ufrn.br/pub/biblioteca/ext/bdtd/FredericoASL.pdf>.

MEYER, B. Object-Oriented Software Construction. 1988.

NUGROHO, L. E. Context-awareness: Connecting computing with its environment. *ICITACEE 2015 - 2nd International Conference on Information Technology, Computer, and Electrical Engineering: Green Technology Strengthening in Information Technology, Electrical and Computer Engineering Implementation, Proceedings*, p. 3–7, 2016.

PERRY, D. E.; ROMANOVSKY, A. B.; TRIPATHI, A. Current Trends In Exception Handling. *Tse*, v. 26, n. 10, p. 921–922, 2000.

ROCHA, L. S. CAEH: Um Método para Verificação de Modelos do Tratamento de Exceção Sensível ao Contexto em Sistemas Ubíquos. 2013.

ROCHA, L. S.; ANDRADE, R. M. C.; GARCIA, A. F. A Method for Model Checking Context-Aware Exception Handling. *2013 27th Brazilian Symposium on Software Engineering*, n. i, p. 59–68, 2013. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6800182>.

TRIPATHI, A.; KNUDSEN, J. L.; DONY, C.; ROMANOVSKY, A. *Advances in Exception Handling Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. 242–242 p. (Lecture Notes in Computer Science, 3). ISSN 03029743. ISBN 978-3-540-41952-5. Disponível em: <http://link.springer.com/10.1007/3-540-45407-1>.

VITERBO, J.; SACRAMENTO, V.; ROCHA, R.; BAPTISTA, G.; MALCHER, M.; ENDLER, M. A Middleware Architecture for Context-Aware and Location-Based Mobile Applications. In: *2008 32nd Annual IEEE Software Engineering Workshop*. Kassandra, Greece: IEEE, 2008. p. 52–61. ISBN 978-0-7695-3617-0. Disponível em: <https://ieeexplore.ieee.org/document/5328450http://ieeexplore.ieee.org/document/5328450/>.

WANG, G.; ZOMAYA, A.; PEREZ, G. M.; LI, K. A Programming Framework for Implementing Fault-Tolerant Mechanism in IoT Applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 9530, p. 771–784, 2015. ISSN 16113349.

WEBER, T. S. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. 2002. Disponível em: <http://www.inf.ufrgs.br/~taisy/disciplinas/textos/Dependabilidade.pdf>.

WEISER, M. The computer for the 21 st century. *ACM SIGMOBILE Mobile Computing and Communications Review*, v. 3, n. 3, p. 3–11, 1991. ISSN 15591662. Disponível em: <http://portal.acm.org/citation.cfm?doid=329124.329126>.

YOON, T. S.; CHOI, J. H.; CHO, E. S.; HELAL, S. A formal modeling for exceptions in context-aware systems. *Proceedings - IEEE 38th Annual International Computers, Software and Applications Conference Workshops, COMPSACW 2014*, p. 734–739, 2014.