UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

BELMONDO RODRIGUES ARAGÃO JUNIOR

SUCCEED: UM *FRAMEWORK* DE SUPORTE À CRIAÇÃO E EXECUÇÃO DE *WORKFLOWS* PARA SAS EM AMBIENTES IOT

FORTALEZA

2018

BELMONDO RODRIGUES ARAGÃO JUNIOR

SUCCEED: UM *FRAMEWORK* DE SUPORTE À CRIAÇÃO E EXECUÇÃO DE *WORKFLOWS* PARA SAS EM AMBIENTES IOT

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Engenharia de Software.

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade.

Co-Orientador: Prof. Dr. Marcio Espíndola Freire Maia.

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A671s Aragão Junior, Belmondo Rodrigues.
SUCCEED : Um framework de suporte à criação e execução de workflows para SAS em ambientes IoT / Belmondo Rodrigues Aragão Junior. – 2018.
97 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2018.
Orientação: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientação: Prof. Dr. Marcio Espíndola Freire Maia.

1. Internet of Things. 2. Auto-adaptação. 3. Workflows. 4. Framework. I. Título.

CDD 005

BELMONDO RODRIGUES ARAGÃO JUNIOR

SUCCEED: UM *FRAMEWORK* DE SUPORTE À CRIAÇÃO E EXECUÇÃO DE *WORKFLOWS* PARA SAS EM AMBIENTES IOT

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Engenharia de Software.

Aprovada em: 31 de Agosto de 2018

BANCA EXAMINADORA

Profa. Dra. Rossana Maria de Castro Andrade. (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Marcio Espíndola Freire Maia. (Co-Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Carlos Raniery Paula dos Santos
Universidade Federal de Santa Maria (UFSM)

Prof. Dr. Everton Ranielly de Sousa Cavalcante
Universidade Federal do Rio Grande do Norte (UFRN)

Prof. Dr. Windson Viana de Carvalho
Universidade Federal do Ceará (UFC)

Dr. Tales Paiva Nogueira
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim e investir em mim, os seus cuidados e dedicação foi que deram a oportunidade de galgar meu caminho atrás dos meus sonhos.

AGRADECIMENTOS

Ter a oportunidade de ser um aluno de mestrado é algo único. Foi um momento de descoberta, ou melhor, redescoberta. Ao remexer nos registros de minha memória, percebo que eu não conseguiria expressar a gratidão que tenho por aqueles que compartilharam um pouco dessa minha caminhada. Entretanto, farei meu melhor para deixar registrado a importância daqueles que me fizeram alcançar essa conquista.

Partindo do princípio, agradeço ao Grande Arquiteto do Universo, por me guiar na busca dos meus objetivos e por sempre encher meu coração de Luz, me concebendo a oportunidade de embelezar minhas ações.

À Profa. Dra. Rossana Maria de Castro Andrade por me orientar em minha dissertação de mestrado. Mais do que isso, pela oportunidade de aprender todos os dias e por cada esforço empenhado nesta minha evolução acadêmica. A minha gratidão por cada conselho, momento de ajuda, puxão de orelha e claro, pelo seu incentivo. Em meu âmago, acredito que quando um professor desencoraja um aluno, ele já falhou como professor. Nesse sentido, lhe agradeço por nunca ter falhado como professora.

Ao Prof. Dr. Marcio Espíndola Freire Maia, pela coorientação e por ter sempre me dado clareza com relação à minha pesquisa. Agradeço por ter me mostrado o "caminho das pedras" em temas que eu ainda tenho muito a aprender.

Ao Dr. Tales Paiva Nogueira, pela paciência nos momentos de ajuda em artigos, bem como por cada dica de escrita científica e por cada correção em meus erros de inglês.

Agradeço à minha família por toda a atenção, amor e carinho. Obrigado por construírem as partes do que eu sou hoje.

Aos amigos que fiz nesse período de pós-graduação, em especial e em ordem alfabética, Bruno Sabóia, Erick Barros, Nayana Carneiro, Rute Castro, e Ticianne Darin. Sem dúvidas, boa parte da força necessária para que eu seguisse em frente veio de simples momentos compartilhados em conversas na Cantina da Química (nosso GigaByte). À todos os funcionários "do administrativo" que sempre ajudam a todos os alunos do GREat. O apoio e o suporte de vocês "nos bastidores" é fundamental no prosseguimento do trabalho de todos.

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

Ainda, o presente trabalho foi parcialmente realizado com o apoio da Fundação Cearense de Pesquisa e Cultura (FCPC).

“Transmita o que aprendeu. Força, maestria. Mas fraqueza, insensatez, fracasso também. Sim, fracasso acima de tudo. O maior professor, o fracasso é. Luke, nós somos o que eles crescem além. Esse é o verdadeiro fardo de todos os mestres.”

(Yoda)

RESUMO

O desenvolvimento de sistemas para o ambiente da Internet das Coisas (IoT) gera a necessidade de se lidar com características como a auto-adaptabilidade e o dinamismo do ambiente. Com isso, a abordagem tradicional de desenvolvimento, baseada em decisões em tempo de *design*, pode ser contraproducente. Considerando ainda o número de dispositivos e tecnologias envolvidos em IoT, isso pode ocasionar uma incapacidade de se lidar com cenários imprevistos em tempo de *design*. Portanto, o desenvolvimento de sistemas para IoT requer o uso de práticas que permitam adaptar o comportamento e a estrutura do sistema de maneira mais flexível. Abordagens baseadas em modelos e estruturas que descrevem o comportamento, como políticas e *workflows*, têm sido propostas na literatura para atingir a capacidade de auto-adaptação. No entanto, as abordagens existentes ou não suportam totalmente os principais requisitos de um ambiente de IoT ou estão acopladas a um ambiente específico e às suas tecnologias. Assim, este trabalho propõe o SUCCEEd, um framework de suporte ao desenvolvimento de aplicações auto-adaptativas, cujo objetivo é prover a criação e execução de sequências de adaptações em um ambiente IoT. Para ilustrar o funcionamento do SUCCEEd, uma prova de conceito é implementada na plataforma Android. Além disso, são feitos experimentos simulados relacionados ao acoplamento e à qualidade da auto-adaptação. Os resultados mostram que o SUCCEEd possui alto grau de reuso, que o tempo usado pelo SUCCEEd para prover a adaptação é menor do que o tempo usado para executar a finalidade da adaptação, e que o número de passos necessários para se prover a adaptação é mínimo em relação ao tempo.

Palavras-chave: *Internet of Things*; auto-adaptação; *Workflows*; *Framework*.

ABSTRACT

The development of systems for the Internet of Things (IoT) environment generates the need to deal with characteristics such as self-adaptability and the dynamism of the environment. Hence, the traditional development approach, based on design-time decisions, can be counterproductive. Considering also the number of devices and technologies involved in IoT, this can cause an inability to deal with unforeseen scenarios at design-time. Therefore, the development of systems for IoT requires the use of practices that allow adapting the behavior and structure of the system more flexibly. Approaches based on models and structures that describe behavior, such as policies and workflows, have been proposed in the literature to achieve the capacity for self-adaptation. However, existing approaches do not fully support the core requirements of an IoT environment or are coupled to a specific environment and its technologies. Thus, this work proposes SUCCEEd, a support framework for the development of self-adaptive applications, in order to provide the creation and execution of sequences of adaptations in an IoT environment. To illustrate how SUCCEEd works, a proof of concept is implemented in the Android platform. In addition, simulated experiments related to coupling and the quality of self-adaptation are performed. The results indicate that (i) SUCCEEd has a high degree of reuse, (ii) the time used by SUCCEEd to provide the adaptation is less than the time used to perform the purpose of adaptation, and (iii) the number of steps required to provide adaptation is minimal in relation to time.

Keywords: Internet of Things; self-Adaptation; Workflow; Framework.

LISTA DE FIGURAS

Figura 1 – Perspectiva de crescimento do número de dispositivos (EVANS, 2011) . . . 19
Figura 2 – Metodologia desta dissertação 21
Figura 3 – O paradigma IoT 23
Figura 4 – Elementos que constituem o cenário IoT 24
Figura 5 – Arquitetura de Referência da WSO2 26
Figura 6 – Elementos de um sistema auto-adaptativo 28
Figura 7 – MAPE-K loop 29
Figura 8 – Exemplo de workflow - processo de publicação de um artigo 30
Figura 9 – Relações entre a definição do *workflow* e o sistema de gerenciamento 31
Figura 10 – Arquitetura do LoCCAM 36
Figura 11 – Metodologia da revisão de literatura aplicada nesta dissertação 39
Figura 12 – Representação de um *workflow* pervasivo segundo a arquitetura de Montagut e Molva (2005) 41
Figura 13 – Visão geral da arquitetura do Presto, descrita por Giner *et al.* (2010) 43
Figura 14 – Exemplificação dos elementos envolvidos na arquitetura do Presto 44
Figura 15 – Elementos envolvidos no tempo de desenvolvimento 45
Figura 16 – Visão geral da arquitetura do PROtEUS 47
Figura 17 – Visão geral da arquitetura do SitOPT 49
Figura 18 – Elementos envolvidos no tempo de desenvolvimento 50
Figura 19 – Exemplo de execução do IFTTT 52
Figura 20 – Exemplo de execução do Node-Red 52
Figura 21 – Tela do FRED 53
Figura 22 – Exemplos de tipos de Nodes 54
Figura 23 – Arquitetura de Referência de SAS descrita por Li *et al.* (2017) 55
Figura 24 – Arquitetura do Adapt definido por Li *et al.* (2017) 56
Figura 25 – Visão geral da abordagem proposta por Gerostathopoulos *et al.* (2018) 57
Figura 26 – Visão geral do SUCCEEd 63
Figura 27 – Arquitetura do SUCCEEd 65
Figura 28 – Primeiro Momento - Nenhuma adaptação é requerida 73
Figura 29 – Segundo Momento - Regra 1 satisfeita 73
Figura 30 – Segundo Momento - Regra 5 satisfeita 74

Figura 31 – Terceiro Momento - Regra 2 satisfeita 75
Figura 32 – Quarto Momento - Regra 3 satisfeita 76
Figura 33 – Quinto Momento - Regra 4 satisfeita 77
Figura 34 – Localização do SUCCEEd na Arquitetura de Referência da WSO2 78
Figura 35 – Localização do SUCCEEd no MAPE-K Loop 78
Figura 36 – Aplicação e Hardware utilizados na prova de conceito..... 81
Figura 37 – Representação do *workflow* da PoC 82
Figura 38 – Avaliação da métrica *Time for Adaptation* do SUCCEEd 86
Figura 39 – Análise de Aleatoriedade de Dados da Biblioteca Utilizada para a Simulação 88

LISTA DE TABELAS

Tabela 1 – Lista com os critérios de inclusão e exclusão dos trabalhos relacionados . . 40
Tabela 2 – Comparativo entre os Trabalhos Relacionados 60
Tabela 3 – Resultado das Métricas 85
Tabela 4 – Comparativo entre os Trabalhos Relacionados e o SUCCEEd 91
Tabela 5 – Lista dos artigos científicos produzidos 92

LISTA DE ALGORITMOS

Algoritmo 1 – Pseudocódigo de uma regra para temperatura menor do que 17^{o} C . . . 68
Algoritmo 2 – Pseudocódigo da Criação de uma Task - Adaptação 69
Algoritmo 3 – Pseudocódigo da instanciação de um workflow sequencial no sistema . 71
Algoritmo 4 – Pseudocódigo da instanciação de um workflow baseado em regras no sistema 71

LISTA DE ABREVIATURAS E SIGLAS

CoAP-CTX	*CoAP Contextual*
COF	*Coupling Factor*
GREat	Grupo de Redes, Engenharia de Software e Sistemas
ICSE	*International Conference on Software Engineering*
IoT	*Internet of Things*
LoCCAM	*Loosely Coupled Context Acquisition Middleware*
PoC	*Proof of Concept*
SAC	*Sensor and Actuator Component*
SAS	*Self-Adaptive System*
SEAMS	*International Symposium on Software Engineering for Adaptive and Self-Managing Systems*
SUCCEEd	*decoupled SUpport meChanism for CrEating and Executing workflows*
SysSU	Sistema de Suporte para Computação Ubíqua
SysSU-DTS	Sistema de Suporte para Computação Ubíqua Distribuída
TA	*Time for Adaptation*
WAT	*Working vs. Adaptivity Time*
WFMS	*Workflow Management System*

SUMÁRIO

1 INTRODUÇÃO 18
1.1 Contextualização 18
1.2 Motivação 19
1.3 Objetivos e Contribuições 20
1.4 Metodologia 21
1.5 Organização da Dissertação 22
2 FUNDAMENTAÇÃO TEÓRICA 23
2.1 Internet das Coisas 23
2.1.1 Elementos e Requisitos 24
2.1.2 Arquiteturas de Referência 25
2.1.3 Desafios no Desenvolvimento de Aplicações IoT 27
2.2 Sistemas Auto-Adaptativos 27
2.2.1 Loop de Controle 28
2.3 Workflow 30
2.3.1 Sistemas de Gerenciamento de Workflow 30
2.3.2 Engines de Workflow 31
2.4 Plataforma de *Middleware* 32
2.4.1 Requisitos de uma Plataforma de Middleware para IoT 32
2.4.1.1 Requisitos de Serviço *33*
2.4.1.2 Requisitos de Arquitetura *34*
2.4.2 LoCCAM 35
2.5 Conclusão 37
3 TRABALHOS RELACIONADOS 38
3.1 Metodologia de Busca 38
3.2 Trabalhos Encontrados 39
3.2.1 Enabling Pervasive Execution of Workflows 41
3.2.2 Presto 42
3.2.3 FESAS 44
3.2.4 SimSota 46
3.2.5 PROtEUS 46

3.2.6 ***SitOPT - A General Purpose Situation-Aware Workflow Management System*** 48
3.2.7 ***A Dynamic Verification Mechanism for Real-Time Self-Adaptive Systems*** . 50
3.2.8 ***IFTTT*** 51
3.2.9 ***Node-Red*** 52
3.2.10 ***ADAPT*** 54
3.2.11 ***Adapting a System with Noisy Outputs with Statistical Guarantees*** 56
3.3 Comparação e Discussão 58
3.4 Conclusão 60
4 SUCCEED 62
4.1 Visão Geral 62
4.2 Arquitetura 64
4.3 Elementos do *Framework* 66
4.3.1 Elementos Modeláveis 66
4.3.1.1 Regras *67*
4.3.1.2 Task *68*
4.3.1.3 Gateways *69*
4.3.2 Elementos Semi-Modeláveis 70
4.3.2.1 Conexão Externa *70*
4.3.2.2 Workflow *70*
4.3.3 Elemento Não-Modelável 71
4.4 Processo de Adaptação 72
4.5 SUCCEEd, as Arquiteturas de Referência e o MAPE-K Loop 77
4.6 Conclusão 79
5 AVALIAÇÃO 80
5.1 Prova de Conceito 80
5.2 Acoplamento 82
5.3 Qualidade da Auto-Adaptação 84
5.4 Ameaças à Validade 87
5.5 Conclusão 89
6 CONCLUSÃO 90
6.1 Resultados Alcançados 90
6.2 Limitações 92

6.3 Trabalhos Futuros 93
REFERÊNCIAS 95

1 INTRODUÇÃO

Esta dissertação apresenta um *framework* de suporte ao desenvolvimento de sistemas auto-adaptativos em ambientes de Internet das Coisas. O referido *framework* se baseia na descrição de adaptações e execução destas adaptações através de *workflows*. A parte central do *framework* é a orquestração das adaptações baseando-se em regras, o que permite que as adaptações sejam executadas de maneira mais flexível, utilizando como referência o estado atual do sistema.

Este capítulo está organizado da seguinte forma: a Seção 1.1 contextualiza e caracteriza o problema que este trabalho aborda. Na Seção 1.2 é apresentada a motivação para o desenvolvimento deste trabalho. Na Seção 1.3 são expostos os objetivos e a contribuição deste trabalho. Na Seção 1.4 é descrita a metodologia utilizada e, por fim, na Seção 1.5 é descrita a organização desta dissertação.

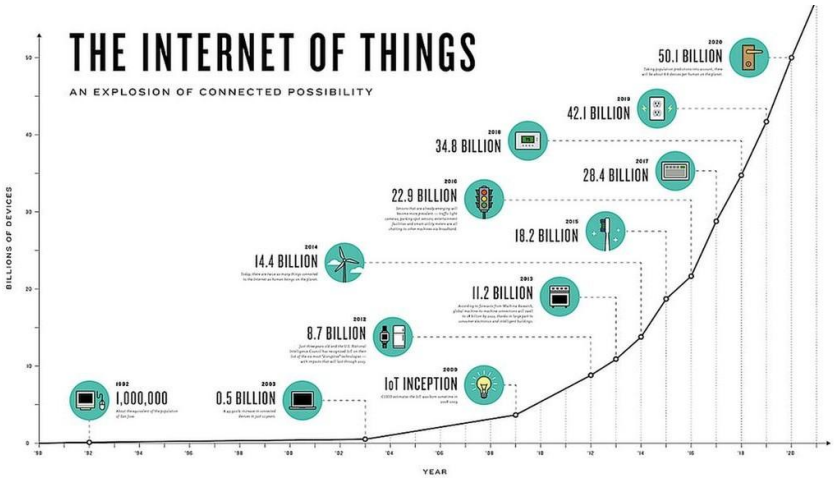
1.1 Contextualização

A Internet das Coisas (do inglês, *Internet of Things* (IoT)) pode ser definida como uma infraestrutura de rede global, dinâmica, autoconfigurável e interoperável, onde objetos do cotidiano estão integrados e possuem identificação e capacidades físicas (SUNDMAEKER *et al.*, 2010). Ainda de acordo com Sundmaeker et al., em um cenário IoT, estes objetos do cotidiano são capazes de interagir entre si e comunicarem-se com o ambiente, trocando dados e informações. Deste modo, influenciam o mundo físico, executando procedimentos que demandam serviços, ações e adaptações no ambiente e no sistema.

O volume mundial destes objetos cresceu rapidamente e a previsão é de que até 2020, o número destes objetos alcance o número de 50 bilhões, como explicitado na Figura 1. Esse cenário de objetos heterogêneos, com suas próprias limitações e capacidades, contribui com a dinamicidade do ambiente. Deste modo, é necessário adaptar o comportamento dos dispositivos de IoT para otimizar: (i) os estados e condições do cenário, e (ii) os requisitos da aplicação. Entretanto, seguindo o modelo tradicional de desenvolvimento, o software retorna aos estágios iniciais do ciclo de produção de um software sempre que há a necessidade de uma alteração ou evolução (BARESI; GHEZZI, 2010). Como mostrado por Baresi e Ghezzi (2010), esse regresso ao ciclo de produção não atende aos requisitos de cenários dinâmicos, que são sujeitos a alterações frequentes, tanto em requisitos quanto no ambiente, e que demandam uma adaptação

rápida a essas alterações. Portanto, a abordagem tradicional de desenvolvimento não é indicada para IoT, dada a complexidade desse cenário e de seus recursos. Sendo assim, é necessário que os sistemas desenvolvidos ajam de maneira autonômica (HUGHES, 2018), i.e. capazes de auto-gerenciamento e de adaptar-se a mudanças imprevisíveis (ZHAO *et al.*, 2009).

Figura 1 – Perspectiva de crescimento do número de dispositivos (EVANS, 2011)



Fonte: Disponível em: https://www.semiwiki.com/forum/content/5559-quick-history-internet-things.html. Acesso em 04/09/2017

Para esse atingir esse objetivo, o processo autonômico baseia-se na criação de modelos que descrevem o comportamento, bem como a estrutura do sistema, que são posteriormente traduzidos em módulos de execução que implementam os requisitos do sistema (KRAMER; MAGEE, 2007). A título de exemplo, para alterar o comportamento de um determinado sistema, os modelos que descrevem os seus estados são analisados. Em seguida, regras de adaptação são geradas em tempo de execução e, a partir destas regras, são implantadas ações de otimização ou correção, através da substituição de módulos inadequados, ou sintonizando parâmetros de execução.

1.2 Motivação

A abordagem baseada em decisões durante o tempo de desenvolvimento, mencionada anteriormente, é contraproducente, basicamente por dois motivos (BARESI; GHEZZI, 2010): i) grande quantidade de dispositivos, tecnologias de suporte e estados da aplicação, o que aumenta a complexidade de especificar, em tempo de desenvolvimento, todas as ações a serem executadas e as respectivas regras de adaptação; e ii) incapacidade de tratar em tempo de execução os cenários não previstos durante o tempo de desenvolvimento. Assim, o desenvolvimento de aplicações e

sistemas para a IoT requer o uso de práticas que permitam adaptar o comportamento e a estrutura da aplicação de forma mais flexível.

Na literatura existem vários trabalhos que suportam algum tipo de adaptação, seja para a aplicação em si ou para o ambiente em que estam inseridos (Giner et al, Wieland et al., Montagut and Molva). Existem ainda, trabalhos que operam com a ideia de situações ou contextos que demandam adaptação (Seiger et al., IFTTT, NodeRed). Em geral, esses trabalhos adotam uma estrutura para geração e execução de *workflows* a fim de permitir alguma adaptação. Contudo, não foi encontrada nenhuma abordagem específica que implemente os requisitos de um bom arcabouço (i.e. *framework*) para o desenvolvimento de um sistema auto-adaptativo (e.g. baixo acoplamento, permissão de detecção de eventos/estados e a capacidade de atuação considerando os estados do sistema). Deste modo, percebe-se a necessidade do desenvolvimento de um mecanismo de apoio ao processo de auto-adaptação voltado especificamente para o desenvolvimento de sistemas IoT.

1.3 Objetivos e Contribuições

Este trabalho tem como objetivo propor um mecanismo de suporte à criação e execução de adaptações para ambientes de Internet das Coisas. Este mecanismo possibilita a auto-adaptação a partir de regras pré-definidas, descritas através do próprio mecanismo e orquestra, em tempo de execução, o processo de auto-adaptação. Para isso, os seguintes objetivos específicos devem ser atingidos:

- Potencializar a execução das adaptações, orquestrando de maneira flexível as possibilidades de adaptação tendo como base para estas adaptações os estados do sistema.
- Ser desenvolvido de modo que possua um baixo acoplamento com demais estruturas (e.g. plataformas de *middleware*) e considerando seus próprios componentes, a fim de permitir uma melhor extensão por parte dos desenvolvedores. Desse modo, extensões de sua estrutura ou modificações podem ocorrer com alterações simples ou sem alterações em módulos existentes.

A contribuição principal deste trabalho de dissertação é um *framework* para apoiar o desenvolvimento de sistemas auto-adaptativos, oferecendo uma estrutura de suporte para descrição e execução de estratégias de adaptação.

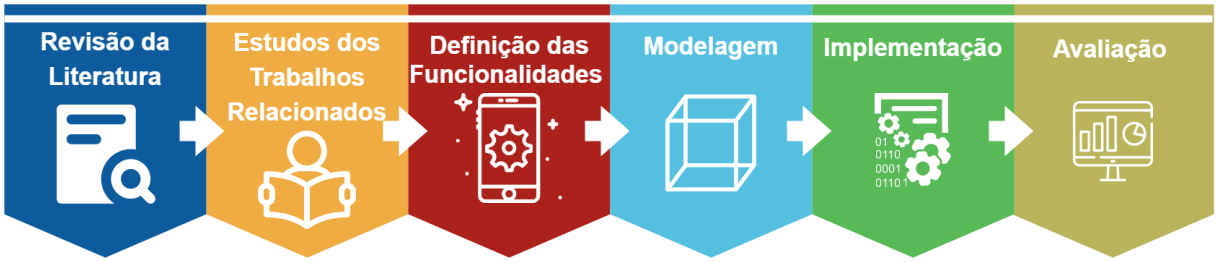
É importante destacar que o escopo deste trabalho restringe-se aos procedimentos de descrição e orquestração do processo de adaptação. Portanto, não está no escopo deste *framework*

lidar com os outros estágios do processo de auto-adaptação, a saber, monitoramento do ambiente, análise de ocorrências e planejamento das adaptações.

1.4 Metodologia

A Figura 2 apresenta a metodologia adotada por este trabalho. A metodologia possui as seguintes fases:

Figura 2 – Metodologia desta dissertação



Fonte: Do autor

- **Revisão da Literatura:** Foi realizada uma revisão bibliográfica envolvendo os conceitos de Internet das Coisas, Sistemas Auto-Adaptativas e *Workflow*. Também foi realizada uma revisão da literatura de trabalhos que descreviam a utilização de *workflows* ou mecanismos desta natureza em ambientes inteligentes.
- **Estudo dos Trabalhos Relacionados:** Com base na revisão da literatura, trabalhos relacionados foram escolhidos fundamentados em dois critérios: i) se apresentavam algum mecanismo de descrição e execução de *workflows*; ii) que sua aplicação fosse em um ambiente IoT ou com a presença de sensores e atuadores. Os trabalhos encontrados foram classificados levando em consideração requisitos necessários para o desenvolvimento de um arcabouço que propicie a implementação de um sistema auto-adaptativo.
- **Definição das funcionalidades do SUCCEEd:** Com a identificação dos trabalhos relacionados, notou-se que os serviços existentes ainda eram vagos ou não contemplavam requisitos de estrutura com capacidadade de auxiliar na engenharia de um sistema auto-adaptativo. Deste modo, foram definidas as funcionalidades que deveriam estar presentes na solução apresentada por este trabalho.
- **Modelagem:** Nesta fase, a arquitetura geral do SUCCEEd foi definida, bem como as interações entre seus componentes.
- **Implementação:** Conforme a modelagem de algum componente do sistema era finalizada, era iniciado sua implementação, adotando um modelo iterativo e incremental de

desenvolvimento (SOMMERVILLE, 2010). Cada módulo do SUCCEEd foi testado individualmente ao final de sua implementação e testes de integração foram feitos à medida que seus componentes eram finalizados, a fim de garantir o bom funcionamento da solução.

- **Avaliação:** Após a implementação do SUCCEEd, foi criada uma Prova de Conceito com o objetivo de constatar a sua viabilidade. Além disso, foram avaliados o acoplamento e a qualidade da adaptação do SUCCEEd através de simulações e métricas.

1.5 Organização da Dissertação

Esta dissertação está organizada em seis capítulos. O capítulo atual descreveu uma breve introdução ao tema, contextualizando o assunto abordado, a motivação, a metodologia, os objetivos e as contribuições deste trabalho.

No **Capítulo 2** são abordados os principais conceitos relacionados a Internet das Coisas, Sistemas Auto-Adaptativos e *Workflows*. Nesse capítulo também são destacados os desafios relacionados à auto adaptação no contexto de Internet das Coisas.

No **Capítulo 3** são descritos os trabalhos relacionados a esta pesquisa. Este capítulo descreve também a metodologia de busca e bases de pequisa utilizadas neste trabalho. Este capítulo descreve também a revisão da literatura utilizada neste trabalho.

No **Capítulo 4** é apresentado o SUCCEEd, um *framework* para descrição, execução e orquestração de auto-adaptação em ambientes de Internet das Coisas. Ainda neste capítulo, é descrita a arquitetura do *framework* proposto e seus componentes, bem como uma visão detalhada do modo de execução e orquestração de adaptações.

No **Capítulo 5** é descrito o processo de avaliação do *framework* proposto. A avaliação ocorreu utilizando uma Prova de Conceito, Avaliações de Acoplamento e da Qualidade da Auto-Adaptação provida pelo *framework*. Ainda nesse capítulo são descritas as ameaças à validade, bem como as suas estratégias de mitigação.

Por fim, o **Capítulo 6** descreve os resultados alcançados, bem como as conclusões deste trabalho e apresenta sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

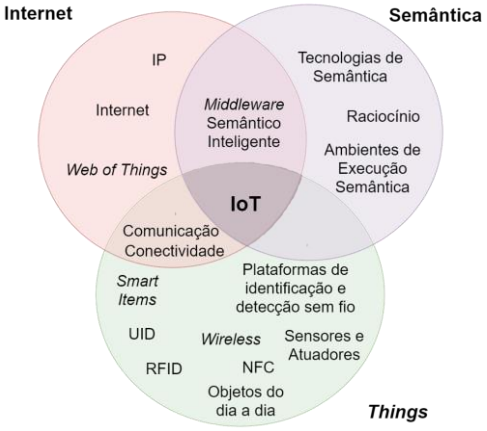
Neste capítulo, os conceitos que constituem a base teórica deste trabalho são apresentados. A Seção 2.1 apresenta conceitos sobre Internet das Coisas, incluindo arquitetura de referência e requisitos encontrados na literatura. Na Seção 2.2 são introduzidas as definições acerca de Sistemas Auto-Adaptativos. A Seção 2.3 apresenta definições de *Workflow*. A Seção 2.4 discorre sobre plataformas de *middleware* e seus requisitos, e apresenta o LoCCAM, a plataforma de *middleware* utilizada neste trabalho. Por fim, a Seção 2.5 expõe as considerações finais sobre os temas debatidos neste capítulo.

2.1 Internet das Coisas

Internet das Coisas (do inglês, *Internet of Things*) não apresenta uma definição consensual, contudo, as definições existentes descrevem que dispositivos com capacidade de sensoriamento e atuação, denominados dispositivos inteligentes, se relacionam mutuamente e com o ambiente, permitindo a troca de dados – através de padrões e protocolos de comunicação – ao passo que reagem a estímulos (ATZORI *et al.*, 2010). Avanços na indústria de hardware, permitem que dispositivos como *notebooks*, *smartphones* e ares-condicionados sejam caracterizados como dispositivos inteligentes (i.e. *smart-objects*), e, até 2020 a quantidade desses objetos deve chegar a 50 bilhões (EVANS, 2011).

A Figura 3 sumariza os principais conceitos e tecnologias destacados por (ATZORI *et al.*, 2010) e que servem de referência para a evolução do paradigma de IoT.

Figura 3 – O paradigma IoT

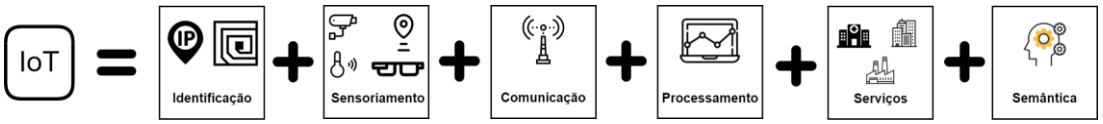


Fonte: Adaptado de Atzori *et al.* (2010)

2.1.1 Elementos e Requisitos

Compreender os elementos que compõem esse paradigma ajuda a entender a sua capacidade, bem como os desafios que lhe são inerentes. A Figura 4 apresenta os seis elementos principais de IoT, descritos por Al-Fuqaha *et al.* (2015).

Figura 4 – Elementos que constituem o cenário IoT



Fonte: Adaptado de Al-Fuqaha *et al.* (2015)

I **Identificação**: A identificação é uma condição importante para esse cenário, tanto para dispositivos quanto para serviços. Ainda, é necessário diferenciar entre a identificação (Id) do objeto e seu endereço. A Id do objeto está relacionado ao seu nome, como “Sensor L” para um sensor de luminosidade. Já o endereço especifica o IP deste objeto dentro de uma determinada rede.

II **Sensoriamento**: O sensoriamento no âmbito de IoT refere-se à capacidade de coletar dados dos dispositivos inteligentes distribuídos pela rede e enviar aos serviços, como banco de dados ou *cloud*. Os dados recebidos são analisados tendo por base os próprios serviços e servem de estímulo para tomada de ações específicas. Para Al-Fuqaha *et al.* (2015), os sensores IoT englobam dispositivos com capacidade de sensoriamento e atuação.

III **Comunicação**: O objetivo das tecnologias de comunicação no âmbito de IoT é possibilitar a conexão de objetos com múltiplas capacidades e serviços específicos. A literatura define exemplos de protocolos: Wi-Fi, *Bluetooth*, IEEE 802.15.4, *Z-wave* e *LTE-Advanced*. Dentre as tecnologias de comunicação, pode-se citar: RFID, *Near Field Communication* e *Ultra-Wide Bandwidth*.

IV **Processamento**: Várias plataformas (e.g., microcontroladores e microprocessadores), como Arduino, Raspberry PI, BeagleBone e T-Mote Sky, são exemplos que contribuem para o cenário IoT. Além destas plataformas, é importante salientar os sistemas operacionais, que se mostram alternativas interessantes para o desenvolvimento de aplicações IoT, como o Contiki. Ainda, o Google com suas alianças "*Open Handset Alliance*"[1] e a "*Open Auto Alliance*"[2] tem envidado esforços para agregar recursos na plataforma Android. A

[1] https://www.openhandsetalliance.com
[2] https://www.openautoalliance.net/about

computação em nuvem forma um outro componente integrante para IoT, considerando os benefícios do processamento de dados em tempo real e *big data*.

V **Serviços**: Os serviços IoT são classificados em (i) serviços relacionados à identidade, (ii) serviço de agregação de informações, (iii) serviços de colaboração, e (iv) serviços ubíquos (XIAOJIANG *et al.*, 2010; GIGLI; KOO, 2011). Os serviços relacionados à identidade são os alicerces de outros serviços, visto que toda aplicação que desejar interagir de algum modo com objetos do mundo real deve identificá-los. Os serviços de agregação de informações são aqueles que coletam e agregam os dados vindos dos sensores que precisam ser processados nas aplicações IoT. Os serviços de colaboração usam os dados obtidos pelos serviços de agregação para tomar decisões e agir. Por fim, os serviços ubíquos visam fornecer serviços de colaboração sempre que forem necessários para quem precisa deles em qualquer lugar.

VI **Semântica**: A semântica corresponde à extração de conhecimento por diferentes objetos a fim de fornecer os serviços necessários. A extração de conhecimento inclui ainda descobrir e usar recursos e analisar dados à título de decisão de serviço. A extração de conhecimento inclui descobrir e usar recursos e informações de modelagem. Além disso, inclui reconhecer e analisar dados para dar sentido à decisão certa de fornecer o serviço exato (BARNAGHI *et al.*, 2012). Com tais características, a semântica representa a inteligência do cenário, responsável por enviar as demandas corretas para os recursos pertinentes.

2.1.2 Arquiteturas de Referência

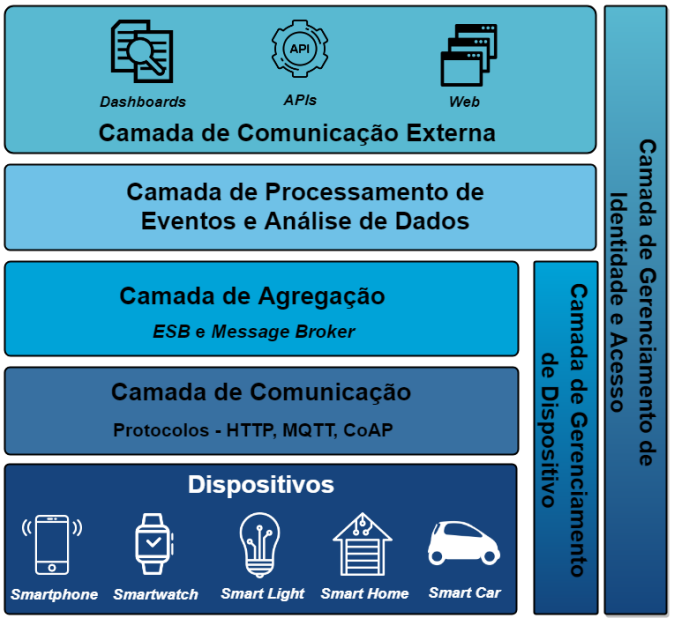
Uma arquitetura de referência pode ser definida como uma abstração que engloba conhecimentos de um determinado domínio de aplicação, podendo simplificar e guiar o desenvolvimento e a evolução de sistemas nesse domínio (NAKAGAWA *et al.*, 2014). Ainda, segundo Cavalcante *et al.* (2015) estabelecer uma arquitetura de referência para IoT pode significar uma questão importante, uma vez que questões como escalabilidade e complexidade crescente se apresentam como um desafio. Desse modo, uma arquitetura de referência pode contribuir com o desenvolvimento de metodologias de desenvolvimento que permitam lidar com os requisitos funcionais e não funcionais desse cenário, como a interoperabilidade (CAVALCANTE *et al.*, 2015).

Dentre as arquiteturas descritas na literatura, duas das mais conhecidas são: *IoT*

Architectural Reference Model (*IoT ARM*) (BASSI *et al.*, 2016) e o modelo proposto pela empresa WSO2 (FREMANTLE, 2015). Cavalcante *et al.* (2015) comparou ambas as arquiteturas e concluiu que a arquitetura da WSO2 é a que oferece uma abordagem mais promissora.

A arquitetura de referência proposta pela WSO2 é baseada em sua experiência no desenvolvimento de soluções IoT. Como descrito por Fremantle (2015), essa arquitetura prevê elementos fundamentais, como: dispositivos inteligentes, serviços e *cloud*. Apesar disso, ela não está vinculada a um conjunto específico de tecnologias e não descreve características particulares, como o funcionamento da *cloud*, por exemplo. Deste modo, cada uma das camadas pode ser instanciada e implementada usando tecnologias que melhor se adequem ao sistema em desenvolvimento.

Figura 5 – Arquitetura de Referência da WSO2



Fonte: Adaptado de Fremantle (2015)

Essa arquitetura de referência é composta por cinco camadas, como demonstrada na Figura 5. Cada camada é descrita a seguir.

A **Camada de Dispositivos** prevê que cada dispositivo deve possuir uma identidade única e comunicação com a Internet; a **Camada de Comunicação** prevê a conexão de cada dispositivo através de protocolos de comunicação; a **Camada de Agregação** prevê o recebimento e a integração de dados provindos dos dispositivos; a **Camada de Processamento de Eventos e Análise de Dados** prevê o processamento e a reação de possíveis eventos provenientes da camada anterior, podendo ainda realizar o armazenamento de dados e a **Camada de Comunicação**

Externa prevê o acesso aos dados e a interação do sistema por parte dos usuários.

A arquitetura da WSO2 fornece ainda duas camadas transversais: a **Camada de Gerenciamento de Dispositivo** e a **Camada de Gerenciamento de Identidade e Acesso**. A primeira prevê a gerência remota e a comunicação dos dispositivos através de protocolos diferentes e a segunda trata da segurança.

2.1.3 Desafios no Desenvolvimento de Aplicações IoT

A natureza dinâmica de um sistema IoT implica em desafios durante as fases de desenvolvimento e pode exigir equipes de desenvolvedores com competências desde eletrônica até a mineração de dados. Aliado a isso, as características de IoT criam um conjunto de desafios no desenvolvimento de aplicações, como descrito por Udoh e Kotonya (2018). Segundo esses autores, os principais desafios no desenvolvimento de aplicações IoT estão no fato de que essas aplicações são (i) distribuídas, (ii) se envolvem em um cenário heterogêneo, (iii) necessitam de uma gerência dos dados, (iv) demandam um processo de manutenção flexível, (v) a maioria são centradas no usuário, (vi) são interdependentes, e (vii) o seu desenvolvimento lida com múltiplos *stakeholders*.

Portanto, o modelo de desenvolvimento de aplicações voltados para IoT deve ser diferente do modelo seguido para as aplicações tradicionais (MORIN *et al.*, 2017). No trabalho de Taivalsaari e Mikkonen (2017) são elencados os sete pontos de distinção entre esses modelos de desenvolvimento: (i) os dispositivos IoT podem integrar um grupo, fazendo parte de uma arquitetura maior; (ii) as aplicações IoT devem manter o funcionamento mesmo diante da indisponibilidade de algum recurso integrante; (iii) o número de unidades de computação é grande, demandando um gerenciamento em massa; (iv) os dispositivos de IoT são incorporados de maneira pervasiva e dependendo das condições físicas do local pode ser de difícil manutenção; (v) é um ambiente heterogêneo, tanto em *hardware* quanto em *software*; (vi) o ambiente requer que as aplicações estejam preparadas para falhas; e (vi) as topologias aplicadas podem ser dinâmicas e momentâneas.

2.2 Sistemas Auto-Adaptativos

Um Sistema Auto-Adaptativo ou *Self-Adaptive System* (SAS) é aquele que possui a capacidade de modificar autonomicamente o seu comportamento, em resposta a estímulos

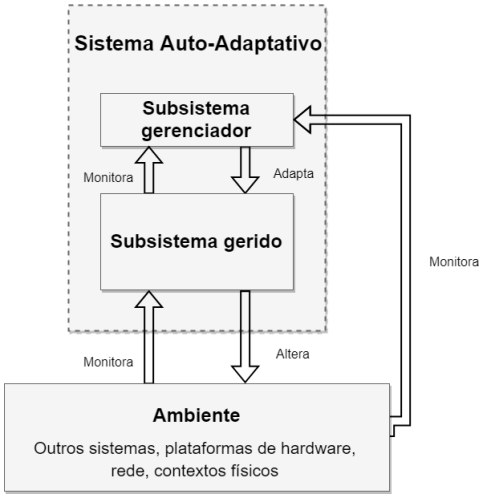
(BRUN *et al.*, 2009). Para tal, deve haver um monitoramento do próprio sistema e do ambiente no qual ele está executando. Deste modo, é possível detectar estímulos, tomar decisões e adaptar-se.

A fim de atingir a auto-adaptação, é necessário que um SAS possua as seguintes propriedades implementadas: autoconfiguração, auto-otimização, autocura e autoproteção (KEPHART; CHESS, 2003). Propriedades estas conhecidas como propriedades *self* -* e que são discutidas a seguir.

A **autoconfiguração** diz respeito à capacidade de um sistema configurar automaticamente o seu comportamento e de seus componentes, baseado em políticas de alto nível. **A auto-otimização** é a capacidade de monitorar e coordenar os recursos do sistema de maneira automática a fim de permitir a evolução do seu comportamento. A **autocura** consiste da capacidade de perceber, diagnosticar e restaurar o sistema na presença de uma falha de software ou hardware. A **autoproteção** é a capacidade de preservar o funcionamento do sistema perante ataques ou falhas e ainda perante mudanças exercidas pelo próprio usuário.

Um sistema auto-adaptativo é composto por um subsistema gerenciado e um subsistema gerenciador (Figura 6). O subsistema gerenciado é composto pela lógica da aplicação que fornece a funcionalidade do sistema. O subsistema gerenciador contém a lógica de adaptação (WEYNS *et al.*, 2013).

Figura 6 – Elementos de um sistema auto-adaptativo



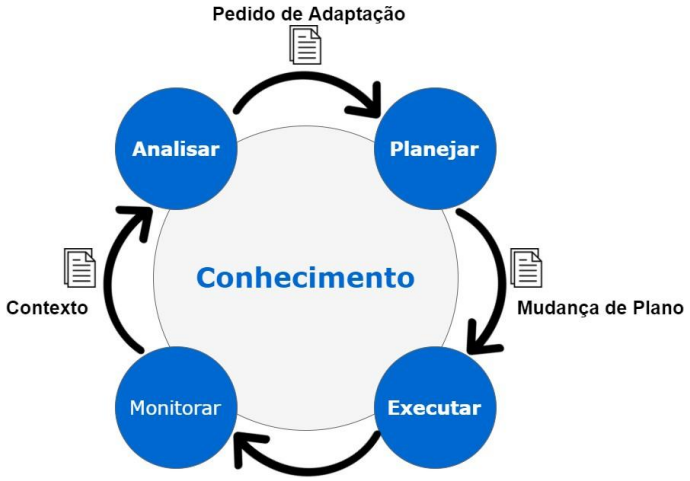
Fonte: Adaptado de Weyns *et al.* (2013)

2.2.1 Loop de Controle

O processo de adaptação segue um loop de controle, conhecido como MAPE-K Loop (Figura 7) e através desse modelo é possível coordenar, manter e evoluir o sistema em

tempo de execução (IBM, 2005). O MAPE-K Loop é composto de 4 fases: Monitorar (M), Analisar (A), Planejar (P) e Executar (E). Circundado por estas 4 fases está o Conhecimento (K), que representa as descrições dos objetivos, requisitos e restrições do sistema.

Figura 7 – MAPE-K loop



Fonte: Adaptado de IBM (2005)

A fase Monitorar (M) é responsável por perceber e coletar os estímulos e mudanças dos recursos do sistema, internos ou externos. Estes estímulos são agregados, correlacionados e filtrados até que sejam caracterizados como um fenômeno que necessite ser analisado. Em caso positivo, essa informação é transmitida para a fase Analisar (A).

Ao receber o modelo do estado atual do sistema, a fase Analisar é responsável por comparar com o estado desejado, previamente conhecido. Esta comparação permite inferir se o estado atual requer qualquer tipo de adaptação. Em caso positivo, uma solicitação de adaptação é passada para a fase Planejar (P).

A fase Planejar tem como função, estruturar um plano de execução a fim de colocar o sistema de volta em um estado desejado. Este plano de execução consiste de uma sequência de ações praticadas de modo a corrigir o fenômeno detectado, podendo ocorrer sob o sistema e seus recursos. Caso não seja possível definir um modo de atingir o estado desejado, o sistema deve receber informações externas sobre como reagir, de um gerenciador ou de outro sistema.

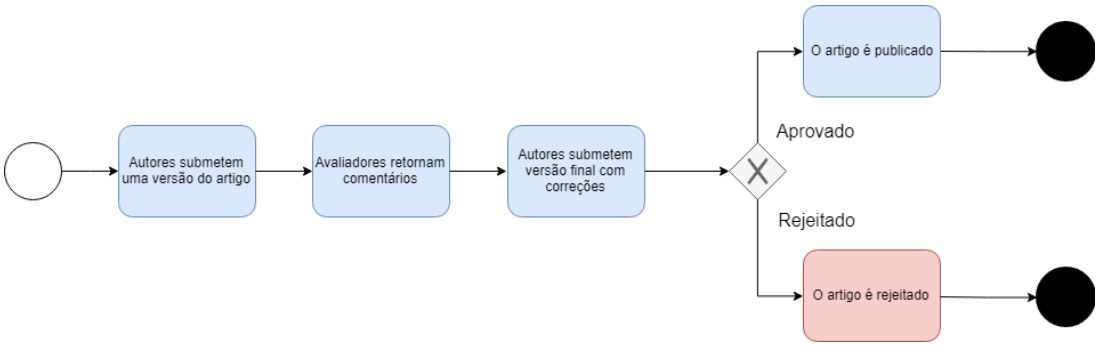
A fase Executar (E) é responsável por executar as ações planejadas na fase anterior.

Já a fase de Conhecimento (K) consiste de uma coleção que contêm as características do sistema, sua política, requisitos, estados, objetivos e ações. Ainda, sempre que MAPE-K Loop é executado, essas informações podem ser atualizadas.

2.3 Workflow

Workflow é um conceito inicialmente associado a automação de processos de informação em uma organização e pode ser definido como uma coleção de tarefas (*tasks*) organizadas, de modo a satisfazer um processo, de acordo com um conjunto de regras definidas (GEORGAKOPOULOS *et al.*, 1995). Ainda segundo Georgakopoulos *et al.* (1995), a estrutura provida pelo *workflow* descreve as tarefas do processo em um nível conceitual, os requisitos do processo e habilidades envolvidas.

Figura 8 – Exemplo de workflow - processo de publicação de um artigo



Fonte: Do autor

A Figura 8 apresenta um exemplo de *workflow*. Neste exemplo, estão presentes cinco etapas da publicação de um artigo. Cada etapa realiza uma processo a partir dos dados fornecidos pela etapa anterior até que o resultado desejado seja produzido por uma das duas últimas etapas.

2.3.1 Sistemas de Gerenciamento de Workflow

O gerenciamento de um *workflow* corresponde a conceder os dados adequados a atividade pertinente, organizando os recursos adequados nos momentos necessários. Assim, um *Workflow Management System* (WFMS) é um sistema que permite definir, gerenciar e executar as atividades de um *workflow* de acordo com uma lógica de domínio, como descrito por Hollingsworth e Hampshire (1995). A Figura 9 mostra a relação entre a definição de um *workflow* e o sistema de gerenciamento.

Um WFMS é, portanto, uma estrutura que deve fornecer as seguintes funcionalidades (HOLLINGSWORTH; HAMPSHIRE, 1995):

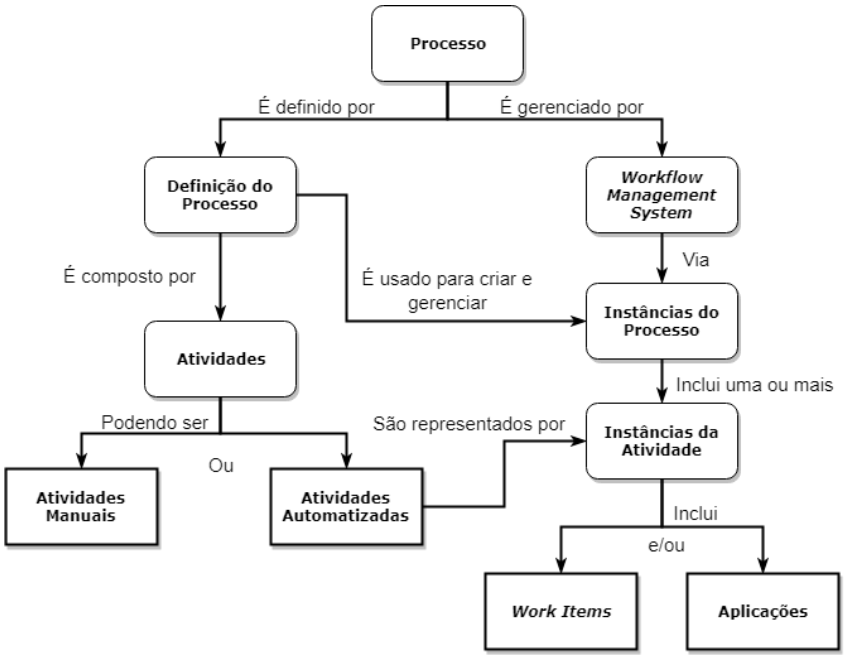
I **Definição do *workflow***: Deve permitir definir e modelar o processo e suas atividades, possuindo a capacidade de adaptar o mundo real para uma representação computacional.

II **Re-*design***: Deve prover um modo que permita o rápido re-*design* e alguma possível

reimplementação do processo.

III **Funções e Interações *Run-time***: Deve gerenciar os processos do *workflow* e deve permitir a interação com usuários e aplicações a fim de processar as atividades.

Figura 9 – Relações entre a definição do *workflow* e o sistema de gerenciamento



Fonte: Adaptado de Hollingsworth e Hampshire (1995)

2.3.2 Engines de Workflow

A *engine* de um *workflow* tem como responsabilidade a gerência e o monitoramento do estado da atividades em execução dentro de um determinado fluxo (HOLLINGSWORTH; HAMPSHIRE, 1995). Além disso, lida com o as informações que transitam dentro de um fluxo. As funções principais de uma engine são: (i) Verificação do Estado, (ii) Verificação de Autorização e (iii) Verificação de Condições. Essas funções são descritas a seguir:

I A **Verificação do Estado** diz respeito à responsabilidade de verificar se é adequado executar uma determinada tarefa, com base no *status* atual.

II A **Verificação de Autorização** diz respeito à responsabilidade de verificar se o usuário atual possui permissão adequada para a execução de determinada tarefa.

III A **Verificação de Condições** diz respeito à responsabilidade de verificar sobre a completude da execução de uma tarefa. Caso a execução seja concluída com êxito, a *engine* retornará com sucesso e caso contrário, a *engine* relata o erro para acionar e reverter a ação

da tarefa executada.

Nesse sentido, algumas *engines* de *workflow* como Copper[3], Taverna[4], jBPM[5], Activiti[6] e MARPLE (PRYSS *et al.*, 2010) mostram-se boas opções para uso. Essas *engines* foram avaliadas sob a perspectiva de execução *mobile*, capacidade de execução e linguagem de notação. Por fim, as *engines* Copper e jBPM foram escolhidas como base para esta dissertação. Entretanto, após um estudo da estrutura e da documentação dessas *engines*, foi possível verificar algumas limitações ao associar essas *engines* com um ambiente IoT. Além disso, estender essas soluções demandaria esforço, considerando a documentação escassa. Ainda, essas *engines* mostram percorrer fluxos de execução de modo sequencial, i.e., sem flexibilidade na execução do percurso das execuções e percebeu-se que essa flexibilidade pode trazer um diferencial para o que é proposto neste trabalho. Por essas razões, foi decidido o desenvolvimento de uma estrutura de *engine* mais flexível para contribuir com a continuidade deste trabalho de dissertação.

2.4 Plataforma de *Middleware*

Uma plataforma de *middleware* atua como uma camada intermediária, responsável por abstrair a complexidade do sistema ou hardware, permitindo que o desenvolvedor envide seus esforços no desenvolvimento em si, sem a necessidade de se preocupar com as particularidades do sistema ou do hardware (e.g. comunicação de rede, coordenação de processos) (GRIGORAS, 2006; NEELY *et al.*, 2006). Em adição, no paradigma IoT, é necessário considerar a característica da heterogeneidade, tanto em comunicação quanto em tecnologia, e uma plataforma de *middleware* deve prover esse suporte (RAZZAQUE *et al.*, 2016).

2.4.1 Requisitos de uma Plataforma de Middleware para IoT

Com base nas características de IoT, Razzaque *et al.* (2016) elicitam um conjunto de requisitos para uma plataforma de *middleware* e os agrupam em dois conjuntos: (i) de serviços e (ii) de arquitetura.

[3] http://copper-engine.org/
[4] https://taverna.incubator.apache.org/
[5] https://www.jbpm.org/
[6] https://www.activiti.org/quick-start

2.4.1.1 Requisitos de Serviço

Dentro desse conjunto de requisitos, Razzaque *et al.* (2016) descrevem outras duas categorias de requisitos, sendo: funcional e não-funcional. Os requisitos funcionais são aqueles relacionados a serviço ou a funções específicas de atuação da plataforma de *middleware* (e.g. gerenciamento de recursos). Em complemento, os requisitos não-funcionais estão relacionados à qualidade do serviço ou ao desempenho (e.g. confiabilidade e disponibilidade). Os requisitos funcionais são:

I **Descoberta de recursos**: Os recursos de IoT envolvem dispositivos heterogêneos (e. g. sensores e *smartphones*), energia e memória dos dispositivos, módulos de comunicação disponíveis nesses dispositivos, infraestrutura de rede e os serviços fornecidos por esses dispositivos.

II **Gerenciamento de recursos**: Os recursos devem ser monitorados e alocados de maneira igualitária, bem como devem ser resolvidos os conflitos associados ao uso desses recursos.

III **Gerenciamento de dados**: Os dados têm papel importante nas aplicações IoT. Por esse motivo, uma plataforma de *middleware* IoT precisa prover o gerenciamento de dados, garantindo a aquisição, processamento e armazenamento desses dados.

IV **Gerenciamento de eventos**: O gerenciamento de eventos tem por objetivo transformar os eventos simples que foram observados, em eventos significativos. Para isso, deve analisar os dados em tempo real para transmitir informações precisas às aplicações.

V **Gerenciamento de código**: A alocação de código seleciona o conjunto de dispositivos para realizar uma tarefa enquanto que a migração de código transfere o código de um dispositivo para outro, possibilitando a reprogramação de nós na rede, permitindo a realocação de processos computacionais.

Os requisitos não funcionais levantados por Razzaque *et al.* (2016) são:

I **Escalabilidade**: Considerando a capacidade de crescimento da rede, a plataforma de *middleware* precisa ser escalável.

II **Tempo Real**: A plataforma de *middleware* deve fornecer serviços em tempo real considerando que as aplicações IoT demandam informações, ao passo que o próprio ambiente e suas condições já são alteradas.

III **Confiabilidade**: A confiabilidade na plataforma de *middleware* ajuda a acrescer a confiabilidade do sistema.

IV **Disponibilidade**: Uma plataforma de *middleware* deve estar disponível sempre que soli-

citado. Mesmo com a ocorrência de falhas no sistema, o tempo de recuperação deve ser ínfimo. Nesse sentido, os requisitos de confiabilidade e disponibilidade juntos melhoram a tolerância a falhas.

V **Segurança e privacidade**: A segurança e a privacidade precisam ser consideradas em todos os blocos, incluindo à nível de aplicação. Isso, pois as informações contextuais pessoais do usuários estão registradas.

VI **Facilidade de implantação**: A implantação de uma plataforma de *middleware* não deve exigir conhecimento ou suporte especializado.

VII **Popularidade**: Um plataforma de *middleware* deve ser continuamente suportada.

2.4.1.2 Requisitos de Arquitetura

Os requisitos de arquitetura tem como objetivo dar suportar aos desenvolvedores das aplicações IoT e serão descritos a seguir.

I **Abstração de Programação**: A plataforma de *middleware* deve fornecer uma API para desenvolvedores de aplicativos ou serviços.

II **Interoperável**: Uma plataforma de *middleware* deve funcionar com dispositivos, tecnologias e serviços heterogêneos.

III **Baseada em serviços**: A arquitetura de uma plataforma de *middleware* deve ser baseada em serviços para garantir a flexibilidade quando surgir a necessidade de adição de novas funções.

IV **Adaptativo**: A plataforma de *middleware*, por mudanças no ambiente, pode precisar adaptar-se, a fim de se ajustar à mudanças.

V **Ciente de Contexto**: A ciência do contexto é um requisito importante, ao considerar a dinamicidade do ambiente. A arquitetura da plataforma de *middleware* precisa garantir a ciência contextual de usuários, dos dispositivos, e do ambiente para oferecer serviços de modo eficaz.

VI **Autônomo**: Os participantes ativos em um ambiente IoT (e.g. dispositivos e tecnologias) devem conseguir manter interação e comunicação entre si, sem uma intervenção humana direta e a plataforma de middleware deve fornecer essa capacidade.

VII **Distribuído**: A arquitetura da plataforma de middleware deve ser pensada de modo a permitir funções distribuídas à nível de infraestrutura. Isso, pois tanto aplicações, dispositivos e usuários trocam informações e colaboram entre si e de modo geograficamente

distribuído.

Nesse seguimento, algumas plataformas de *middleware* como Fiware[7], Google Awareness[8], LoCCAM (MAIA *et al.*, 2013) e ContextNet (ENDLER *et al.*, 2011; ENDLER; SILVA, 2018) mostram-se boas opções para uso. Para este trabalho, a plataforma de *middleware* escolhida foi o LoCCAM (MAIA *et al.*, 2013), plataforma esta que foi desenvolvida no laboratório do Grupo de Redes, Engenharia de Software e Sistemas (GREat)[9] e possui como foco a plataforma Android.

O principal diferencial do LoCCAM ao compará-la às demais plataformas de *middleware* mencionadas é o seu foco em plataformas Android, bem como o fato da documentação acerca de seu funcionamento e modo de implementação ser vasta e mais acessível [10]. Além disso, facilita os testes associados à sua execução, considerando a possibilidade de executar em um *smart-object*, como um *smartphone* ou um *smartwatch*.

2.4.2 LoCCAM

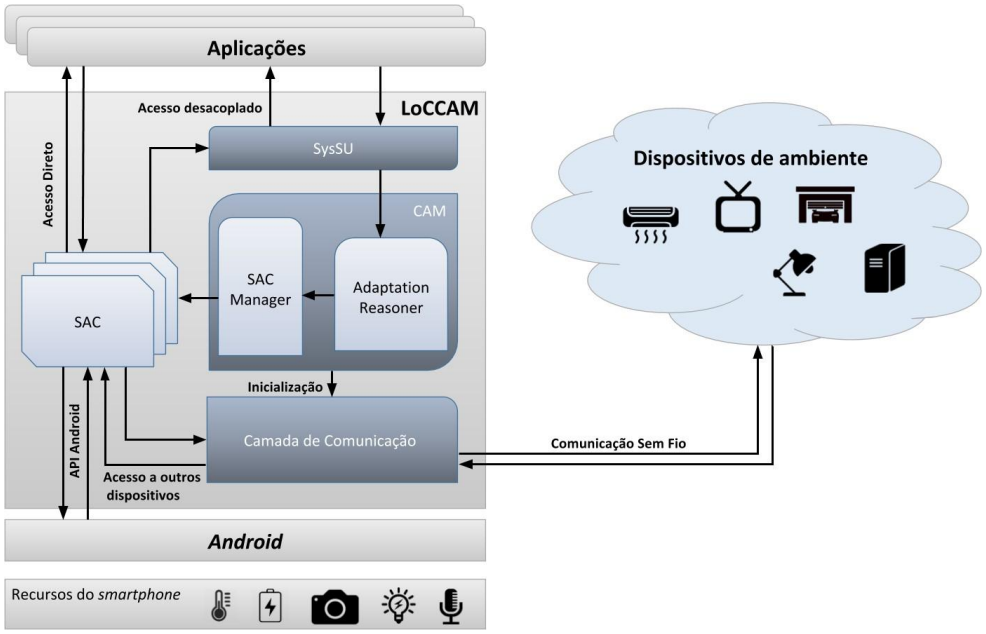
A primeira estrutura que contribuiu para o desenvolvimento do LoCCAM foi o Sistema de Suporte para Computação Ubíqua (SysSU) (LIMA, 2011). O SysSU é um sistema de suporte com o objetivo de facilitar a coordenação entre os dispositivo envolvidos e a descrição/descoberta/invocação de serviços considerando as características dos sistemas ubíquos. Posteriormente, o SysSU evoluiu para o Sistema de Suporte para Computação Ubíqua Distribuída (SysSU-DTS) (NETO, 2013), e o seu modelo de coordenação passou a ser baseado em espaços de tuplas distribuídas. O *Loosely Coupled Context Acquisition Middleware* (LoCCAM) (MAIA *et al.*, 2013) utiliza como base esses trabalhos e o acresce de uma camada de aquisição de contextos e outra camada que gerencia o processo. Em seguida, o trabalho desenvolvido por Barreto (2017) acrescentou mais uma funcionalidade à essa plataforma, contribuindo com o acréscimo do *CoAP Contextual* (CoAP-CTX), uma extensão do protocolo CoAP, permitindo o processo de descoberta de *smart objects* de modo mais adequado ao interesse do usuário e do contexto atual.

Na sua arquitetura atual, o LoCCAM possui cinco componentes principais, como demonstrado na Figura 10. Os componentes da arquitetura são: (i) *Sensor and Actuator Component*

[7] https://www.fiware.org/
[8] https://developers.google.com/awareness/
[9] http://www.great.ufc.br/
[10] http://loccam.great.ufc.br/

(SAC); (ii) *Sensor and Actuator Component Manager*; (iii) *Adaptation Reasoner*; (iv) *System Support for Ubiquity* (SysSU), e (v) Camada de Comunicação. Cada um desses componentes, demonstrados na Figura 10, são descritos a seguir.

Figura 10 – Arquitetura do LoCCAM



Fonte: Retirado de Barreto (2017)

I ***Sensor and Actuator Component* (SAC)**: São as estruturas responsáveis pelo encapsulamento do acesso aos sensores e atuadores, estando esses sensores ou atuadores no dispositivo ou no ambiente. Além disso, se caracterizam como componentes, o que garante o reuso de um determinado SAC por todas as aplicações que necessitarem.

II ***SAC Manager***: É a estrutura responsável por gerenciar o ciclo de vida dos SACs, garantindo também a instalação e ativação dos SACs em tempo de execução.

III ***Adaptation Reasoner***: É a estrutura responsável por indicar quando determinado SAC deve ser iniciado ou retirado da execução, baseado no interesse das aplicações.

IV ***System Support for Ubiquity (SysSU)***: É a estrutura que coordena a interação entre as aplicações e os SACs e utiliza como representação destes, um espaço de tuplas.

V **Camada de Comunicação**: Essa estrutura é responsável por abstrair a troca de mensagens entre os SACs e os dispositivos presentes no ambiente. Além disso, abstrai as particularidades de cada comunicação e deste modo a comunicação pode ocorrer através de *Bluetooth* ou *Wifi*, por exemplo.

2.5 Conclusão

Este capítulo apresentou os conceitos básicos de Internet das Coisas, Sistemas Auto-Adaptativos, *Workflows* e Plataformas de *Middleware*, que constituem a base teórica desta dissertação. Com relação a Internet das Coisas, foi apresentada a definição de IoT, seus elementos e requisitos, bem como as arquiteturas de referência. Também foram apresentados os desafios presentes no desenvolvimento de aplicações para o cenário IoT.

Este capítulo também apresentou uma visão geral sobre Sistemas Auto-Adaptativos, sua definição, suas propriedades e elementos constituintes. Também foi discutido o MAPE-K Loop, o *loop* de controle que guia o processo de adaptação.

Este capítulo ainda discorre sobre uma das estruturas que podem ser utilizadas para atingir a auto-adaptação, os *workflows*. Essa estrutura contribui na definição do processo de determinado domínio e permite, a partir de regras de negócio, executar um determinado processo. Já os sistemas de gerenciamento de *workflow* cooperam e coordenam as atividades do processo para oferecer um comportamento global desejado. Com tal recurso, o desempenho de uma aplicação pode ser melhorado, dado que permite a evolução do sistema mediante a interação dos diversos módulos que o formam. Também foi descrita a importância de uma *engine* e suas características.

Por fim, este capítulo apresentou os conceitos de plataforma de *middleware* e discutiu os requisitos necessários para o seu funcionamento em um ambiente IoT. Após a análise das plataformas de *middleware* para IoT, houve a escolha da plataforma utilizada para o desenvolvimento deste trabalho, o LoCCAM. O LoCCAM permite a aquisição de informação contextual e o envio de ações para os dispositivos móveis. A sua última versão permite ainda o processo de descoberta sensível ao contexto, através do protocolo CoAP. A escolha dessa plataforma de *middleware*, aconteceu principalmente devido a documentação vasta acerca de seu funcionamento e modo de implementação. Além disso, facilita os testes associados à sua execução, considerando a sua associação direta a um *smart-object*, como um *smartphone*.

3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados e discutidos os trabalhos relacionados ao tema desta dissertação. A Seção 3.1 apresenta a metodologia de busca utilizada para identificar os trabalhos relacionados e os critérios levantados para comparação. Na Seção 3.2 são apresentados os trabalhos relacionados encontrados. Na Seção 3.3 é apresentado um comparativo entre esses trabalhos. Por fim, a Seção 3.4 conclui este capítulo.

3.1 Metodologia de Busca

Com o objetivo de encontrar os trabalhos relacionados, foi realizada uma busca na literatura por publicações que estivessem relacionadas a um dos seguintes tópicos: i) Especificação de adaptações em ambientes que possuam sensoriamento ou atuação, a fim de identificar abordagens que possam ser utilizadas em ambientes IoT, mesmo sem ter sido desenvolvido para tal objetivo; ii) Adaptações baseadas em planos de execução ou *workflows*, para identificar as abordagens existentes que utilizam como cerne a ideia de sequência de ações ou adaptações em um ambiente dinâmico; e iii) Mecanismos ou *Frameworks* que permitem a especificação de adaptações, em tempo de desenvolvimento, e a execução de adaptações, em tempo de execução da aplicação.

É preciso mencionar que esses tópicos foram escolhidos em razão do *framework* proposto neste trabalho utilizar o conceito de fluxo de execuções ou *workflows* para permitir o processo de adaptação. Além disso, foi definido que seria necessário verificar abordagens que pudessem ser utilizadas em IoT, baseado no seu uso em cenários com características semelhantes, como sistemas ubíquos.

Ainda, a respeito das buscas, apesar de não ter sido elaborada uma revisão sistemática, como descrito por Kitchenham (2004), alguns elementos desse processo de pesquisa foram aplicados neste trabalho, a saber, definição de *strings* de buscas e de bases de dados. Dessa maneira, para os tópicos supracitados, foram definidas *strings* de busca, em inglês, com um conjunto de permutações entre as seguintes palavras-chave:

(IoT OR Internet of Thing OR cyber-physical-system* OR Ubiquitous Computing) AND (Workflow* OR Smart-Workflow OR Adaptive-Workflow* OR Process Execution).*

Como bases de dados, foram utilizados, inicialmente, o IEEE [1] e o SCOPUS[2].

[1] https://ieeexplore.ieee.org/Xplore/home.jsp
[2] https://www.scopus.com/search/form.uri

Entretanto, percebeu-se vantagens ao utilizar apenas a SCOPUS como meio de busca, dado que possui agregado em si uma grande quantidade de fontes de busca [3], o que tornou possível, inclusive, encontrar os mesmos trabalhos encontrados antes na IEEE e na ACM. Além disso, a sua ferramenta de busca é robusta, como descrito por Meho e Yang (2007), o que permite uma busca com mais possibilidade de combinações. Como descrito por Meho e Yang (2007), a busca básica oferecida pela SCOPUS permite palavras chaves e buscas à partir de frases, enquanto que na busca avançada permite buscas por ano, nome do autor, título, ano e número de páginas, por exemplo. Além disso, oferece classificação por autor, ano de publicação, fonte, assunto, categoria e tipo de documento.

Por fim, fez-se a busca de trabalhos em anais do *International Conference on Software Engineering* (ICSE) e do *International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (SEAMS), mais especificamente entre os anos de 2017 e 2018. Esses eventos, com *workshops* específicos das áreas de SAS, foram escolhidos devido a possibilidade de apresentarem trabalhos mais recentes e que poderiam ser relacionados a esse trabalho de dissertação. A Figura 11 resume esse processo.

Figura 11 – Metodologia da revisão de literatura aplicada nesta dissertação

Fonte: Do Autor

O processo de seleção dos estudos utiliza como base os critérios de inclusão e exclusão dos estudos primários. A saber, esses critérios foram definidos de modo a estarem alinhados com os objetivos da pesquisa. Esses critérios estão sumarizados na tabela 1.

Após a leitura dos artigos encontrados e dos anais do ICSE e do SEAMS, tendo como base as leituras de título, resumo e artigo completo, foram selecionados 11 artigos.

3.2 Trabalhos Encontrados

Como mencionado anteriormente, foi realizada a busca por publicações relacionadas a um dos seguintes tópicos: i) Especificação de adaptações em ambientes que possuam sensoriamento ou atuação; ii) Adaptações baseadas em planos de execução ou *workflows* e iii) Mecanismos ou *Frameworks* que permitem a especificação e execução de adaptações.

[3] https://www.scopus.com/sources.uri?zone=TopNavBarorigin=searchbasic

Tabela 1 – Lista com os critérios de inclusão e exclusão dos trabalhos relacionados

	Critérios de Inclusão	Critérios de Exclusão
I	Deve ser escrito em Inglês	Não estar escrito em língua inglesa
II	Deve estar publicado em *workshop*, conferência, revista ou jornal entre os anos de 2003 e 2018.	Estar disponível somente em forma de *abstract*, apresentações, resumo expandido
III	apresentar um *framework* para prover adaptação em IoT	Não se tratar de um trabalho primário.
IV	Apresentar uma solução para prover adaptação em ambientes IoT	Não apresentar uma solução voltada para IoT
V	Apresentar uma solução para prover adaptação em ambientes com características semelhantes ao de IoT, como em sistemas ciber-físicos	Não apresentar uma solução focada no processo de adaptação do sistema
VI	Apresentar uma solução para prover adaptação, de modo que possa ser estendida ou utilizada em IoT	Não apresentar uma solução que possa ser estendida para IoT
VII	Apresentar uma proposta ou metodologia que permita prover algum modo de adaptação em um sistema	

Fonte: Do autor

Nesse sentido, foram encontrados 11 trabalhos relacionados, a saber: *Enabling Pervasive Execution of Workflows* (MONTAGUT; MOLVA, 2005), Presto (GINER *et al.*, 2010), FESAS (KRUPITZER *et al.*, 2013), SimSota (ABEYWICKRAMA *et al.*, 2014), PROtEUS (SEIGER *et al.*, 2015), SitOPT (WIELAND *et al.*, 2015), *A Dynamic Verification Mechanism for Real-Time Self-Adaptive Systems* (TSUDA *et al.*, 2016), IFTTT[4], Node-Red[5], *ADAPT* (LI *et al.*, 2017) e *Adapting a System with Noisy Outputs with Statistical Guarantees* (GEROSTATHOPOULOS *et al.*, 2018).

O Enabling Pervasive Execution of Workflows (MONTAGUT; MOLVA, 2005) propõe uma arquitetura de suporte à execução distribuída em ambientes pervasivos; o Presto (GINER *et al.*, 2010) provê estruturas que são utilizadas para descrever a conexão física-virtual dos dispositivos; o FESAS (KRUPITZER *et al.*, 2013) fornece uma estrutura baseada em processos para troca de componentes em tempo de execução; o SimSota (ABEYWICKRAMA *et al.*, 2014) é um *plugin* que adota o desenvolvimento dirigido a modelos a fim de permitir a modelagem do sistema e simular padrões arquiteturais auto-adaptativos; o PROtEUS (SEIGER *et al.*, 2015) é um ambiente de execução integrado baseado em metamodelos para atingir a auto-adaptação; o SitOPT (WIELAND *et al.*, 2015) promove a adaptação através do reconhecimento

[4] https://ifttt.com
[5] https://nodered.org

de uma determinada situação; o *Dynamic Verification Mechanism for Real-Time Self-Adaptive Systems* (TSUDA *et al.*, 2016) propõe um mecanismo de verificação dinâmica de restrições em tempo de execução; o IFTTT[6] fornece uma estrutura para descrição de automações baseada em condições; o Node-Red[7] fornece uma estrutura para automação de atividades com base em fluxo; o ADAPT (LI *et al.*, 2017) fornece uma estrutura de modelagem e estruturação do sistema e gera partes específicas de código. Por fim, o *Adapting a System with Noisy Outputs with Statistical Guarantees* (GEROSTATHOPOULOS *et al.*, 2018) propõe um *framework* que combina otimização de tempo de execução com dados obtidos de testes estatísticos.

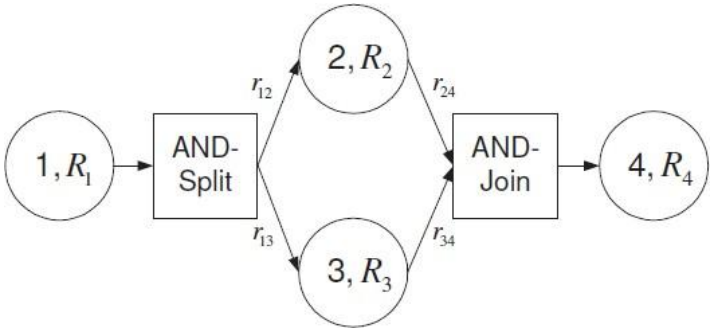
Nas Subseções seguintes, esses trabalhos são melhor apresentados, através da discussão de sua arquitetura, modo de funcionamento e exposição de características relevantes aos seus processos de adaptação.

3.2.1 Enabling Pervasive Execution of Workflows

O trabalho de Montagut e Molva (2005) propõe uma arquitetura de suporte à execução distribuída de *workflows* em ambientes pervasivos, baseando-se na capacidade de controle descentralizado e na atribuição dinâmica de funções aos dispositivos. Essa arquitetura parte do princípio de que a execução distribuída de *workflows* em ambientes pervasivos deve ser independente de controle centralizado, considerando a natureza dinâmica desse ambiente.

Nesse trabalho, um *workflow* pervasivo W é representado usando um grafo direcional. Cada vértice representa as ações do *workflow* e as arestas representam a seqüência de passos do *workflow*. A Figura 12 apresenta um exemplo de *workflow* segundo essa proposta.

Figura 12 – Representação de um *workflow* pervasivo segundo a arquitetura de Montagut e Molva (2005)



Fonte: Retirado de Montagut e Molva (2005)

[6] https://ifttt.com
[7] https://nodered.org

- t_{in}: representam atividades compostas por ações executadas pelos dispositivos que participam do *workflow*. Cada atividade é executada localmente por um determinado dispositivo e um vértice *n* representa um conjunto de atividades executadas localmente no mesmo dispositivo. Desse modo, cada vértice é vinculado a um único dispositivo e dois vértices vizinhos são vinculados a dois dispositivos diferentes. Cada atividade é identificada por um índice *i* e pelo índice *n* do vértice onde é executado.
- r_{ij}: representam um processo de roteamento usado para transferência de dados e controle entre os dispositivos. No grafo, são as arestas simbolizando os dados trocados entre os dispositivos. Nessa representação, *i* e *j* são os índices dos vértices ligados por essa aresta.
- f: representam o modo de execução, bem como as possíveis dependências existentes entre o processo de roteamento e as tarefas. As dependências podem ser (i) Sequenciais, (ii) AND, representando atividades executadas em paralelo e (iii) OR, representando condicionais.

Seguindo o processo demonstrado na Figura 12, o dispositivo executando a atividade do vértice 1 envia dados para os dispositivos responsáveis pelos vértices 2 e 3 (r_{12} e r_{13}). Esses vértices são executados em paralelo e enviam para o dispositivo executando no vértice 4. Essa abordagem considera cenários de execução do *workflow* no qual não são tratados problemas de sincronização e as atividades são executadas em paralelo sem considerar restrições. Além disso, não há a implantação de tratamentos de falhas.

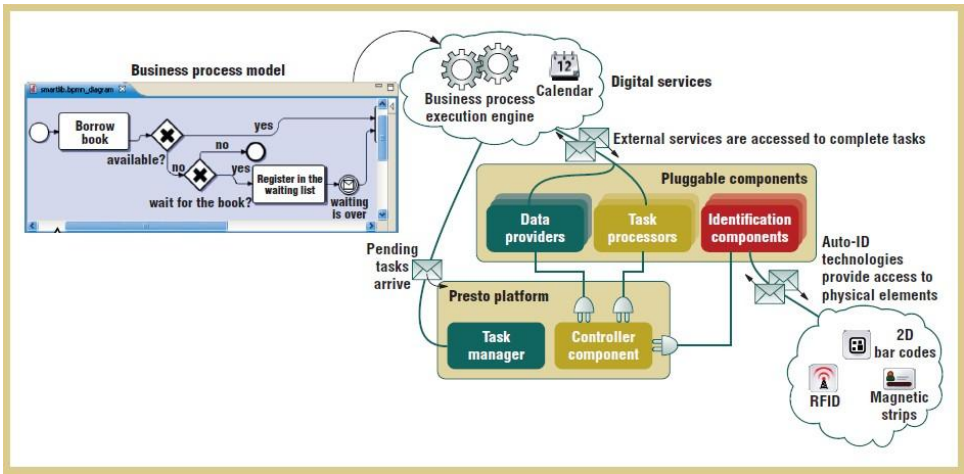
3.2.2 Presto

O trabalho desenvolvido por Giner *et al.* (2010) tem por objetivo fornecer suporte a fim de sistematizar o desenvolvimento de *workflows* por parte dos desenvolvedores. Esse trabalho provê aos desenvolvedores *building blocks* que são utilizados para descrever a conexão física-virtual dos objetos. Para prover suporte à execução dos sistemas definidos segundo esses conceitos, esse trabalho propõe o Presto, uma arquitetura de software plugável, demonstrada na Figura 13 .

O Presto foi definido seguindo o processo arquitetural proposto por Völter (2005). Esse processo é definido de modo a obter arquiteturas de software afetadas minimamente pelos ciclos tecnológicos, bem como permitir que os requisitos evoluam à medida que o desenvolvimento se torna automatizado.

A arquitetura do Presto, como demonstrado na Figura 13 é constituída de cinco

Figura 13 – Visão geral da arquitetura do Presto, descrita por Giner *et al.* (2010)



Fonte: Retirado de Giner *et al.* (2010)

elementos: (i) *task manager*, (ii) *task processors*, (iii) *identification components*, (iv) *data providers*, e (v) *controller*.

O *Task Manager* (i) comporta-se como um *buffer* onde peças pendentes aguardam conclusão. Ele recebe mensagens de serviços externos e espera que os componentes processem as mensagens. O *Task Processor* (ii) tem como finalidade suportar a extensibilidade da funcionalidade das regras de negócio do sistema. Tem ainda como finalidade fornecer aos usuários as informações, os serviços e a interação necessárias para concluir um tipo específico de tarefa. O *Identification Components* (iii) tem como responsabilidade suportar a extensibilidade à nível tecnológico. Sendo assim, é responsável por prover mecanismos para acessar o ambiente físico e por transferir os identificadores dos dispositivos para o sistema. Também fornecem recursos para capturar ou gerar identificadores com uma tecnologia específica.

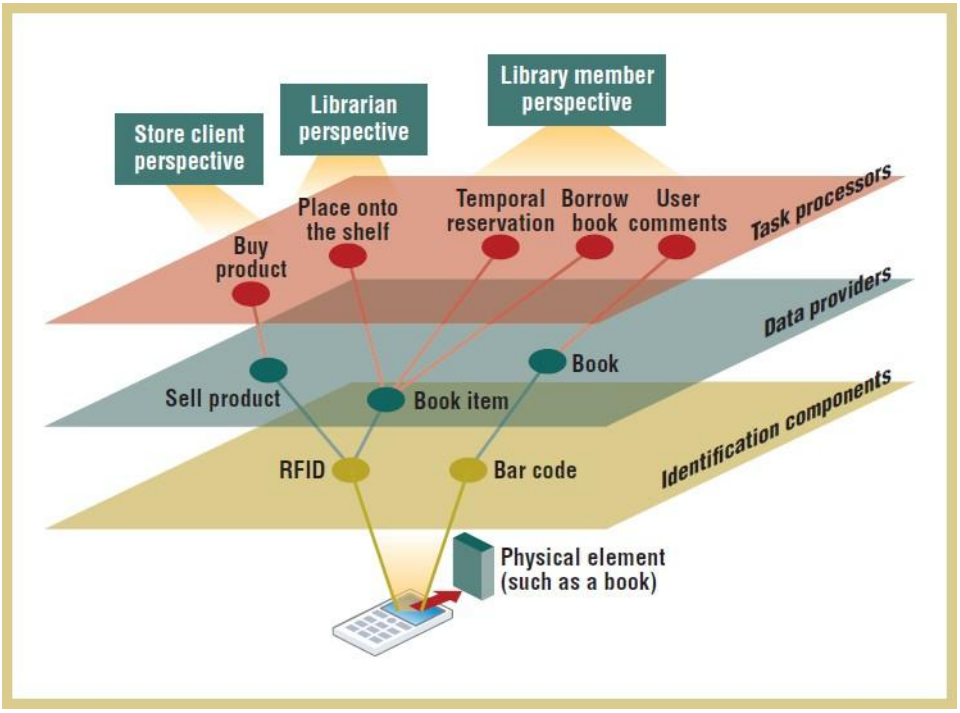
Os *Data Providers* (iv) são responsáveis por transformar os identificadores que a camada anterior (iii) fornece em informações relevantes para o usuário. Estabelecem dinamicamente a conexão entre elementos físicos e suas contrapartes virtuais, sendo portanto, cada *data provider* uma possível projeção de um elemento físico. O *Controller* (v) tem como responsabilidade coordenar todos os outros componentes e determinar quais perspectivas são válidas para um determinado contexto. Determinar uma perspectiva é definir o conjunto de *processors* (ii) associados à contraparte digital de um elemento físico.

Para exemplificar melhor a arquitetura proposta, Giner *et al.* (2010) utilizam como exemplo um sistema de biblioteca. No exemplo, dois componentes de identificação fornecem diferentes identificadores por meio de tecnologias RFID e código de barras. Os *data providers* mapeiam esses identificadores em diferentes contrapartes digitais: um *data provider* considera o

elemento físico como uma cópia específica de um livro que pode ser emprestado; Já outro *data provider* o considera como uma representação de uma obra literária.

O Presto é projetado para orientar os usuários para a realização das tarefas do *workflow*. Deste modo, os usuários podem esperar até que o sistema detecte um determinado objeto no mundo real e as possíveis tarefas a serem executadas ou decidir sua próxima tarefa à ser executada, baseado nas informações que o sistema fornece.

Figura 14 – Exemplificação dos elementos envolvidos na arquitetura do Presto



Fonte: Retirado de Giner *et al.* (2010)

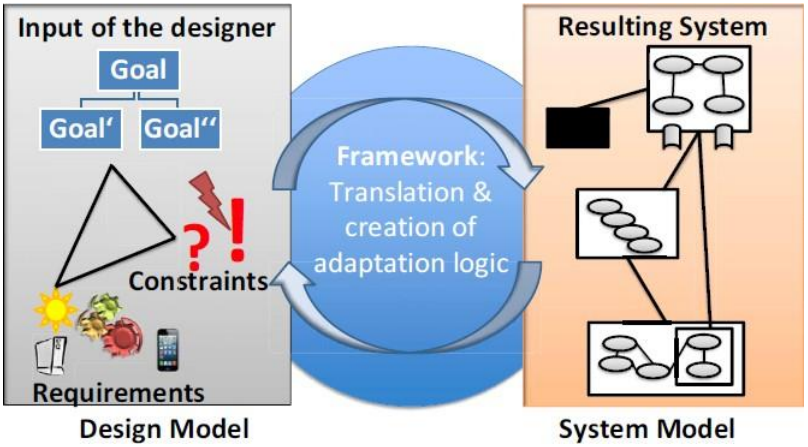
Como método de avaliação, é usado o *Mobile Business Service Questionnaire* (MOBIS-Q), um *survey* para medir a experiência em serviços *mobile*. As dimensões avaliadas foram: (i) a usabilidade percebida, (ii) a adequação ao contexto e (iii) o impacto em termos de produtividade. Portanto, não é apresentado nenhuma questão com relação à qualidade da adaptação fornecida ou ao desempenho do *framework*. Além disso, não é possível perceber se sua arquitetura dá suporte à atuação no ambiente, parecendo apenas utilizar o sensoriamento como ponto de partida para uma possível adaptação.

3.2.3 FESAS

O *framework* proposto por Krupitzer *et al.* (2013) tem como objetivo simplificar o desenvolvimento de SAS através de uma estrutura que se baseia em processos e componentes

reutilizáveis. O foco deste trabalho é na adaptação composicional, que tem a troca de componentes do sistema como base para o processo de adaptação. Para isso, o especialista, chamado nesse trabalho de "*designer*", deve fornecer, em tempo de desenvolvimento, uma abstração de alto nível do sistema. Essa abstração deve englobar os objetivos, as restrições e os requisitos da aplicação e essa abstração, chamada de *Design Model*, é traduzida em requisitos para a lógica da adaptação e essa tradução é chamada de *System Model*. A Figura 15 apresenta esse processo.

Figura 15 – Elementos envolvidos no tempo de desenvolvimento



Fonte: Retirado de Krupitzer *et al.* (2013)

No trabalho de Krupitzer *et al.* (2013), uma versão orientada a serviços estava em fase de desenvolvimento e não foram encontrados trabalhos posteriores do FESAS. Nesse primeiro protótipo, a plataforma de *middleware* utilizada era o BASE (BECKER *et al.*, 2003). O BASE era utilizado como suporte ao uso de serviços em dispositivos remotos e com sua utilização, o sistema era modelado como uma conexão de serviços. Deste modo, a adaptação consistia como uma troca de serviços.

Por propor um uso genérico, a ideia de avaliação desse *framework* consistia de diferentes cenários, como hospital e tráfego inteligente. Nesse trabalho, ainda era necessária a definição de uma métrica relacionada à qualidade da adaptação. Como descrito, esse trabalho apresenta uma versão inicial daquilo que seria proposto. Além disso, apresenta algumas lacunas a definir, como o componente de transformação do *design model* que ainda deveria ser implementado.

3.2.4 SimSota

O SimSOTA, apresentado por Abeywickrama *et al.* (2014), é um plugin para a IDE Eclipse e tem como finalidade auxiliar no processo de arquitetura, engenharia e implementação de um SAS. De modo prático, o SimSOTA adota o desenvolvimento dirigido a modelos, a fim de permitir a modelagem do sistema e simular padrões arquiteturais auto-adaptativos, bem como automatizar a geração de código Java.

O SimSOTA utiliza o modelo SOTA (do inglês, *State of Affairs*) como uma estrutura de análise de requisitos de autoconsciência e de auto-adaptação. O modelo SOTA é caracterizado por um espaço de estados virtuais n-dimensionais no qual a execução do sistema se situa. Nesse modelo, um objetivo do sistema é considerado como a eventual realização de um determinado estado, enquanto que um *utility* - estrutura definida nesse modelo - descreve alguma restrição no caminho da execução e que deve ser respeitado ao tentar atingir um determinado objetivo. Deste modo, um sistema é considerado autoconsciente se puder reconhecer autonomamente sua posição atual e direção de movimento nesse espaço n-dimensional, enquanto que a auto-adaptação diz respeito à capacidade do sistema de direcionar dinamicamente sua trajetória.

A validação desse trabalho ocorre através de um estudo de caso, o e-veículo, cujo o objetivo é chegar ao destino com a energia planejada e no tempo planejado, considerando as restrições da carga da bateria, temperatura, aceleração, velocidade e localização atual do veículo.

Entretanto, esse trabalho explora apenas questões relacionadas às características de modelagem do sistema, através de uma linguagem estendida da UML e a tradução desses modelos para *self-adaptive patterns* em Java. Além disso, é explicitado que o SimSOTA foi desenvolvido para ajudar na implementação de SAS que se baseiam na abordagem de *loop* que o trabalho descreve, sendo esse *loop* uma extensão do MAPE-K Loop tradicional. Porém, o artigo não deixa claro a capacidade de suporte para o *loop* tradicional ou outros *loops*. Também, outra questão que não é esclarecida é o seu comportamento em tempo de execução do SAS, parecendo tratar e suportar apenas quesitos da descrição do SAS.

3.2.5 PROtEUS

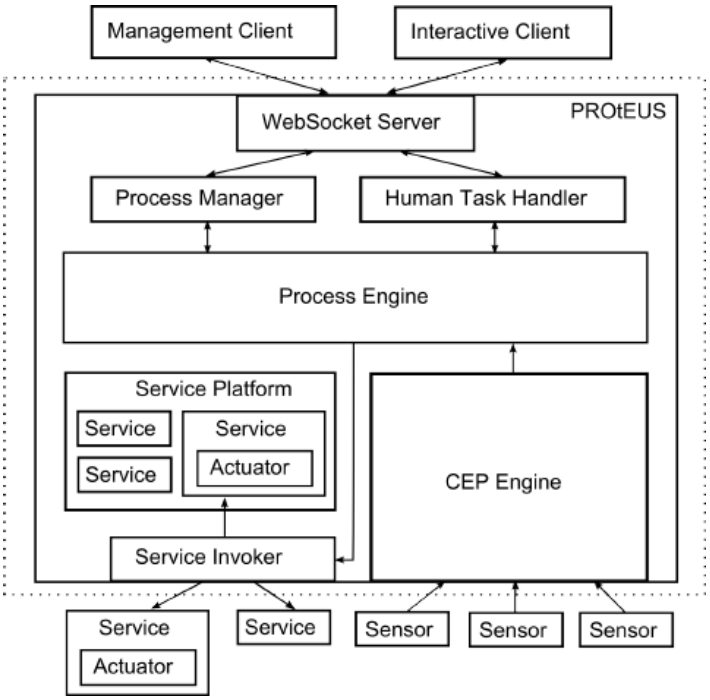
O trabalho desenvolvido por Seiger *et al.* (2015) apresenta o PROtEUS (do inglês, *integrated process execution system for cyber-physical systems*), um ambiente de execução composto por um mecanismo principal para executar processos baseados em modelos, um

mecanismo de processamento de eventos complexos (CEP) para a integração e processamento de dados de sensores e um invocador de serviço para acionar serviços locais ou externos.

O núcleo do PROtEUS consiste de um metamodelo que descreve a estrutura dos modelos de processo e um mecanismo de processo responsável por instanciar esses modelos, executando instâncias. O acesso à funcionalidade de controle de processo do mecanismo é obtido pelo gerente de processos. O mecanismo de processamento de eventos complexos (CEP) é responsável por processar os dados de sensores externos e acionar eventos de alto nível dentro de instâncias de processos.

Uma visão geral da arquiteura do PROtEUS é apresentada na Figura 16. O núcleo do ambiente de execução é o *Process Engine* que tem seu funcionamento baseado em redes de Petri. A *Engine* instancia um modelo de processo e percorre a descrição do processo com base no metamodelo do processo. O *Process Manager* é responsável pelo gerenciamento dos modelos de processo e suas instâncias. Fornece também acesso à informações de monitoramento sobre modelos de processos e estados de instâncias de processos.

Figura 16 – Visão geral da arquitetura do PROtEUS



Fonte: Retirado de Seiger *et al.* (2015)

O *Complex Event Processing Engine* (*CEP Engine*) é responsável por perceber um determinado conjunto de eventos. O mecanismo CEP analisa o fluxo de entrada de eventos de baixo nível procurando padrões registrados. Ao reconhecer um padrão, os ouvintes corresponden-

tes são informados e os eventos de alto nível são ativados na instância do processo, continuando com a execução.

O PROtEUS utiliza serviços com a finalidade de controle de atuadores, bem como para fazer atribuições de funcionalidades remotas. Desse modo, o *Service Invoker* é responsável por fornecer acesso a serviços Web externos por meio de adaptadores específicos de protocolo. Ao necessitar, o modelo do processo é enviado para o *service invoker*, passando os atributos necessários (e.g. URI e parâmetros) para o invocador de serviço. Para Seiger *et al.* (2015), o *service invoker* pode ser considerado como uma plataforma de *middleware*, ao criar solicitações de acordo com o tipo de serviço.

O *Human Task Handler* é responsável por enviar solicitações de interação para os dispositivos de interação conectados ao sistema de execução. O mecanismo aguarda a resposta do usuário para que continue a execução posteriormente. A comunicação é feita através de *publish/subscribe* usando o servidor *WebSocket*.

Como prova de conceito do PROtEUS, foi desenvolvido um estudo de caso, empregando um protótipo do PROtEUS em uma Smart House. Por ser foco do trabalho apenas apresentar uma arquitetura de referência para execução de processos voltada para *cyber-physical systems*, avaliações de performance e relacionadas à execução ficaram como trabalhos futuros. Além disso, o trabalho que dá continuidade ao PROtEUS (HUBER *et al.*, 2016) sugere a necessidade de o instalar, bem como de instalar os dispositivos do ambiente através de uma estrutura de *Dashbord* , o *OpenHub*, o que torna a execução acoplada ao ambiente ou dispositivos.

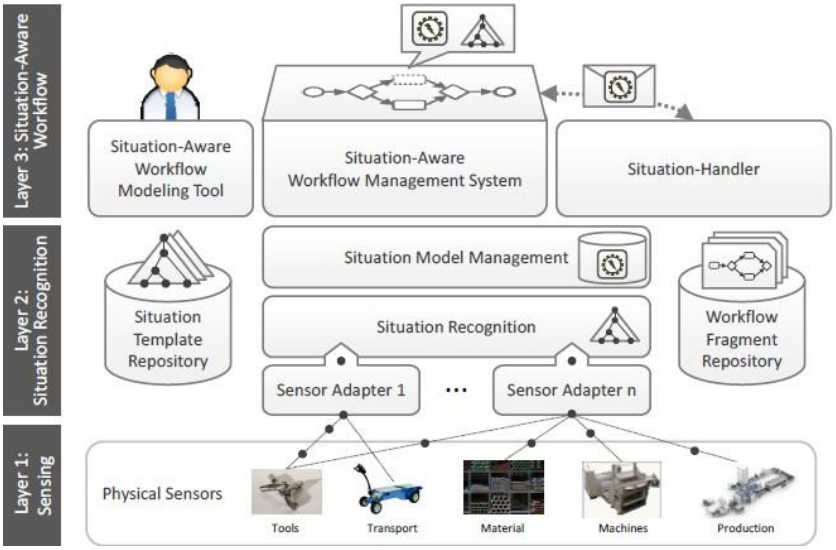
3.2.6 SitOPT - A General Purpose Situation-Aware Workflow Management System

A abordagem apresentada por Wieland *et al.* (2015) - o SitOPT - é baseada em informações de contexto agregadas, caracterizando uma situação. Uma situação pode ser usada por diferentes *workflows* a fim de prover adaptação à situação atual em seu ambiente de execução. O SitOPT permite a detecção de situações usando sistemas de reconhecimento de situação, troca de informações sobre situações detectadas e adaptação em tempo de execução com base nas situações reconhecidas.

O conceito principal do SitOPT é o desacoplamento da modelagem do *workflow*, adaptação e o de reconhecimento de situações. O *workflow* é modelado por um especialista do domínio, especificando o fluxo padrão e todas as possíveis exceções em diferentes situações. Para cada exceção, um sub-fluxo alternativo do *workflow* afetado pode ser definido, sendo definido

como um fragmento do *workflow*. As tarefas afetadas no *workflow* original são definidas como controle de fluxo situacional. Esses controles de fluxo são marcados com certas situações que devem ser consideradas ao executar essas tarefas. Caso uma determinada situação ocorra, o sistema de gerenciamento do *workflow* terá que trocar o fluxo de controle situacional do *workflow* padrão por um fragmento apropriado à situação. Uma visão geral da arquitetura do SitOPT é apresentada na Figura 17.

Figura 17 – Visão geral da arquitetura do SitOPT



Fonte: Retirado de Wieland *et al.* (2015)

A arquitetura do SitOPT é definida em três camadas, sendo: (i) *Sensing*, (ii) *Situation Recognition* e (iii) *Situation-Aware Workflow*. A camada de *Sensing* é composto pelo ambiente e seus sensores físicos. Os sensores fornecem todas as informações contextuais e eventos para a camada superior. Além disso, os modelos de contexto também podem ser usados como sensores. Na camada de *Situation Recognition*, os dados vindos dos sensores são filtrados e agregados usando fluxos de dados ou processamento de eventos complexos. Com isso, o reconhecimento da situação deriva do estado situacional atual do ambiente. A última camada, chamada de *Situation-Aware Workflow*, é responsável por executar e adaptar o *workflow* com base nas situações detectadas.

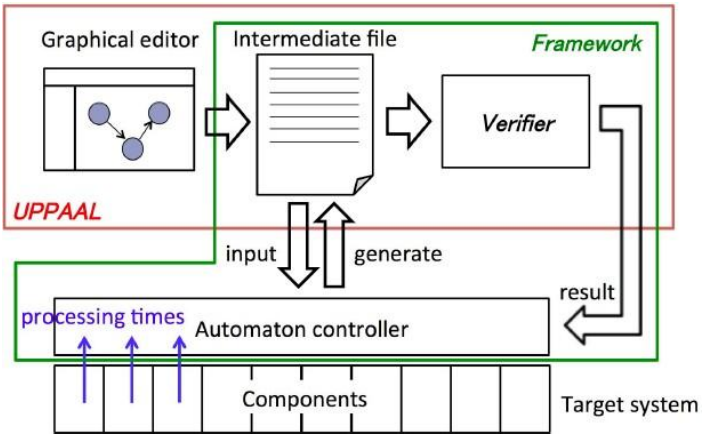
Entretanto, esse trabalho se concentra em apresentar os conceitos que o norteiam e a arquitetura do *framework*. Ainda nesse trabalho, não é apresentado validação referente ao seu desempenho e capacidade de adaptação. Além disso, nesse primeiro protótipo, é utilizado o Node-Red como base para os workflows modelados. Por isso, apesar de reconhecer uma determinada situação e instanciar um *workflow* para essa situação, a execução desse *workflow*

segue uma sequência de execução rígida, sem considerar o estado do sistema após uma única adaptação do fluxo.

3.2.7 A Dynamic Verification Mechanism for Real-Time Self-Adaptive Systems

No trabalho desenvolvido por Tsuda *et al.* (2016) é proposto um mecanismo de verificação dinâmica de restrições em tempo de execução. Para isso, é utilizado o UPPAAL, uma ferramenta de verificação de modelos e que pode ser usado para projetar SAS. Com este mecanismo, um protótipo de *framework* foi desenvolvido. Esse *framework* monitora periodicamente os tempos de processamento dos componentes do sistema. Quando há um aumento ou diminuição brusca no tempo de processamento do sistema, o *framework* faz a análise do comportamento atual do sistema com relação às restrições de tempo definidas. Esse *framework* considera que o sistema é implementado em uma arquitetura baseada em componentes. A Figura 19 mostra uma visão geral do funcionamento desse *framework* proposto.

Figura 18 – Elementos envolvidos no tempo de desenvolvimento



Fonte: Retirado de (TSUDA *et al.*, 2016)

Nesse *framework*, o UPPAAL é utilizado tanto para modelar o sistema quanto para sua verificação em tempo real. Através do editor gráfico da ferramenta, é possível contruir uma rede de autômatos cronômetrados que é usada como entrada inicial. Aliado a isso, um verificador é utilizado em tempo de execução e, caso uma fórmula não esteja satisfeita, procura uma nova configuração que satisfaça todas as fórmulas. Deste modo, geram-se novos autômatos cronometrados, substituindo os autômatos em execução. Por fim, ao satisfazer todas as fórmulas, o framework reconfigura o sistema baseado no novo autômato.

O seu funcionamento segue o processo a seguir. Com o *input* inicial de um autômato,

é criado um autômato raiz do sistema. O autômato raiz é construído de modo que consiga representar todas as funções do sistema. Cada função é decomposta em estruturas chamadas de *task*. Essas *tasks* são representadas pelos estados do autômato raiz. Deste modo, as *tasks* continuam sendo decompostas em *tasks* mais específicas e a cada decomposição, cada task criada compõe o autômato. A decomposição continua até que um autômato possa ser processado por um componente do sistema, que a partir desse momento, é chamado de *module*. Caso necessite ser processado por mais de um componente do sistema, é chamado de *strategy*. Com isso, é possível calcular o tempo de processamento de uma função somando os *modules* relevantes.

Nesse trabalho, foi conduzido um experimento simulado com sistema de controle de drone. Foram construídas dez funções para o drone (e.g. decolar e aterrisar). Cada função foi decomposta em *tasks* e foi criado um automaton raiz que conectava as dez funções. Cada tarefa foi decomposta, resultando em quatro *strategies* e 17 *modules* e foram definidas restrições de tempo para cada função. Dois cenários foram simulados, um com o mau funcionamento dos componentes e outro com a degradação do motor. Com essa simulação, foi possível perceber que ao injetar falhas, o *framework* detectava as respectivas violações e disparava o processo de reconfiguração. Entretanto o foco o trabalho é baseado na restrição de tempo e não há detalhes sobre a possibilidade de implementação de outras restrições como contexto ou estado do sistema.

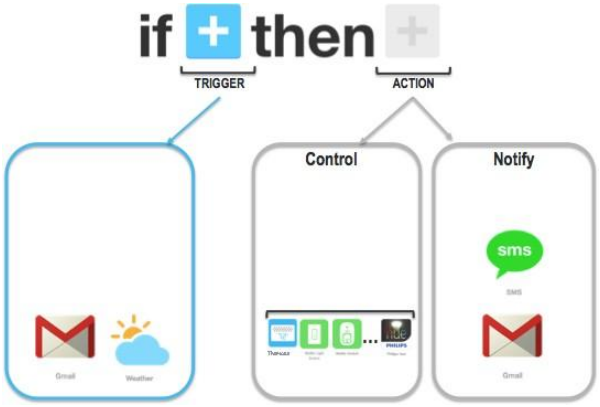
3.2.8 IFTTT

O IFTTT [8] tem seu nome derivado da declaração condicional utilizada em programação "If this, then that"e é uma plataforma com capacidade de integrar aplicações, dispositivos e serviços. Para atingir esse objetivo, a plataforma cria uma corrente de declarações condicionais, chamadas de *applets*. Um *applet* é disparado por meio de alguma mudança que ocorra nos *web-services* das aplicações ou serviços.

A sua estrutura de funcionamento é composta de cinco elementos, sendo: *Channels*, *Triggers*, *Actions*, *Ingredients* e *Recipes*. O *Channel* é o elemento central da execução do IFTTT. Cada *channel* possui seus próprios *triggers* e *actions*. Os *Triggers* são a parte análoga ao *"this"* e são possibilidades de ocorrência, como : "Fui marcado em uma foto no Facebook". As *Actions* são a parte análoga ao *"that"* e são atividades a serem executadas, como: "me envie uma mensagem de texto". Os *Ingredients* são pedaços isolados de dados do *Trigger*. Como exemplo, um *ingredient* do *trigger* acima citado seria o "usário que marcou"ou "data da marcação". Por

[8] https://ifttt.com

Figura 19 – Exemplo de execução do IFTTT



Fonte: Adaptado de http://www.bemoss.org/bemoss-application-and-data-management-layer/

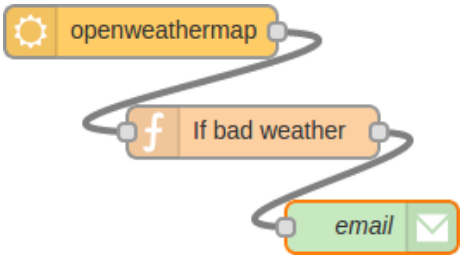
fim, os *Recipes* são combinações de *trigger*s e *action*s de um *channel* ativo.

Apesar de ser uma estrutura que trabalha com o conceito de contexto e informação, o seu foco de utilização é a automação de pequenas ações. Isso não garante que a mesma consiga desempenhar o processo de adaptação para tarefas mais complexas do que as dispostas em sua ferramenta. Além disso, não foi encontrado nenhum estudo acerca de sua capacidade de prover esses serviços ou acerca de seu desempenho.

3.2.9 Node-Red

O Node-Red[9] é uma ferramenta de desenvolvimento baseada em fluxos, originalmente desenvolvida pela IBM a fim de integrar dispositivos físicos, APIs e serviços. O Node-Red fornece um editor via browser para o desenvolvimento dos fluxos. Através desse editor, é possível criar funções em JavaScript, bem como salvar elementos da aplicação para reuso posterior[10]. Para prover a execução, a aplicação é montada com base no Node.js. Um exemplo de execução no Node-Red é mostrado na Figura 20.

Figura 20 – Exemplo de execução do Node-Red

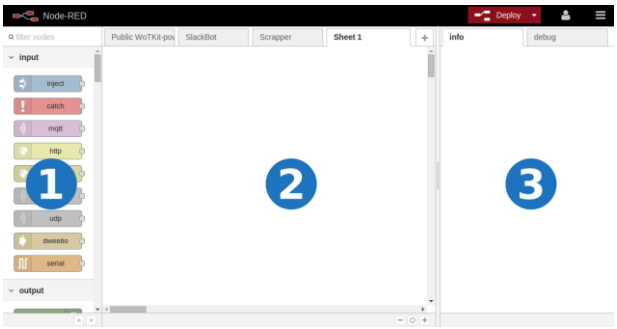


Fonte: http://developers.sensetecnic.com/article/introduction-to-node-red/

[9] https://nodered.org
[10] http://developers.sensetecnic.com/article/introduction-to-node-red/

A gerência das instâncias do Node-Red é feita através do *Front-End* do Node-Red (do inglês, FRED). Essa plataforma fornecida pelo Node-Red se responsabiliza por gerenciar e otimizar as instâncias.

Figura 21 – Tela do FRED



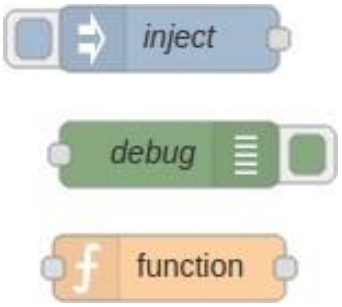
Fonte: Retirado de http://developers.sensetecnic.com/article/introduction-to-node-red/

O Node-Red é composto de três elementos básicos, descritos na Figura 21, sendo: *Node Panel*, *Sheets Panel* e *Info e Debug Panel*.

O *Node Panel* (1) contêm uma lista de *nodes*, organizados por categorias, que podem ser utilizados na aplicação. Dentre as categorias estão opções de *input*, *output*, função e armazenamento. O *Sheet Panel* (2) é o local onde os fluxos de execução dos *nodes* são descritos. Para isso, basta arrastar os *nodes*, organizá-los em estruturas de folhas e conectá-los. O *Info* e *Debug Panel* (3) fornece as abas de informações e *debug*. A aba de informações fornece informes acerca de um determinado node escolhido. Essas informações tem por objetivo facilitar o entendimento das funcionalidades de determinado *node*, bem como características como o tipo, propriedades, modos e requisitos de uso. A aba de *debug* fornece informações relacionadas ao processo de execução e debug, como mensagens de erros ou de *log*.

Em relação aos *nodes*, existem três tipos, como exemplificado na Figura 22, sendo: *Input*, *Output* e de Processamento.

Figura 22 – Exemplos de tipos de Nodes



Fonte: http://developers.sensetecnic.com/article/introduction-to-node-red/

Os *Input Nodes* permitem a inserção de dados no fluxo. Esses dados podem ser provenientes de outros serviços, como mídias sociais ou *sockets*, bem como dados inseridos de modo manual, como injeção de dependências. Como característica visual, eles possuem um quadrado branco no lado direito. Os *Output Nodes* permitem a saída de dados do fluxo, sendo utilizado para enviar dados para outros serviços como os supracitados ou para enviar *logs* para o console de *debug*. Como característica visual, eles possuem um quadrado branco no lado esquerdo. Por fim, os Nodes de Processamento tem por objetivo processar dados, transformando dados recedibos (e.g. json) e os usando para um determinado processamento. Como característica visual possuem quadrados brancos nos lados esquerdo e direito.

Contudo, tal qual o IFTTT, o seu foco de utilização é a automação de pequenas ações, o que não garante que consiga desempenhar o processo de adaptação. Além disso, o Node-Red mostra uma estrutura rígida de execução, restringindo a execução a um fluxo pré-determinado.

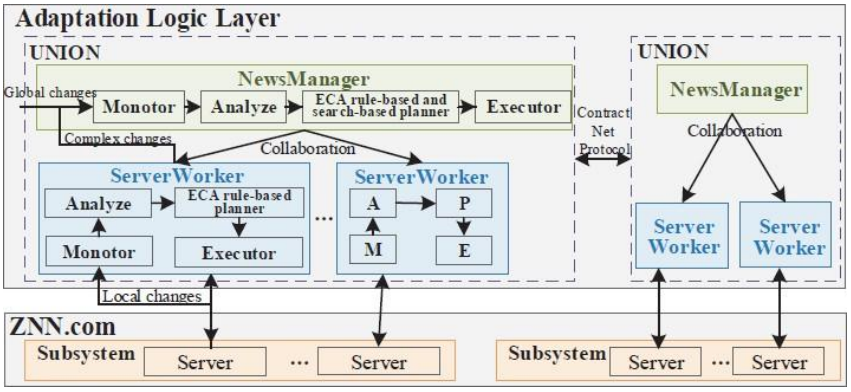
3.2.10 ADAPT

O ADAPT (*Agent-based Development toolkit and operAtion PlaTform*) é um *toolkit* desenvolvido por Li *et al.* (2017) para dar suporte ao desenvolvimento de SAS. A proposta desse *toolkit* é lidar com mudanças complexas, baseando-se na teoria de sistemas multi-agentes (do inglês, Multi-Agent Systems (MAS)) e engenharia de software baseada em busca(do inglês, Search-Based Software Engineering (SBSE)). Por sua estrutura de *toolkit*, em tempo de *design*, ele permite modelar a organização da estrutura do sistema, além de gerar uma estrutura de código e partes dos Agentes e suas relações.

Com o intuito de permitir que os SAS possam lidar efetivamente com múltiplas e interrelacionadas mudanças, os autores definem uma arquitetura de referência (Figura 23)

composta de duas partes: *Target System Layer* e o *Adaptation Logic Layer*. Essa arquitetura é baseada nas teorias de MAS e SBSE.

Figura 23 – Arquitetura de Referência de SAS descrita por Li *et al.* (2017)



Fonte: Retirado de Li *et al.* (2017)

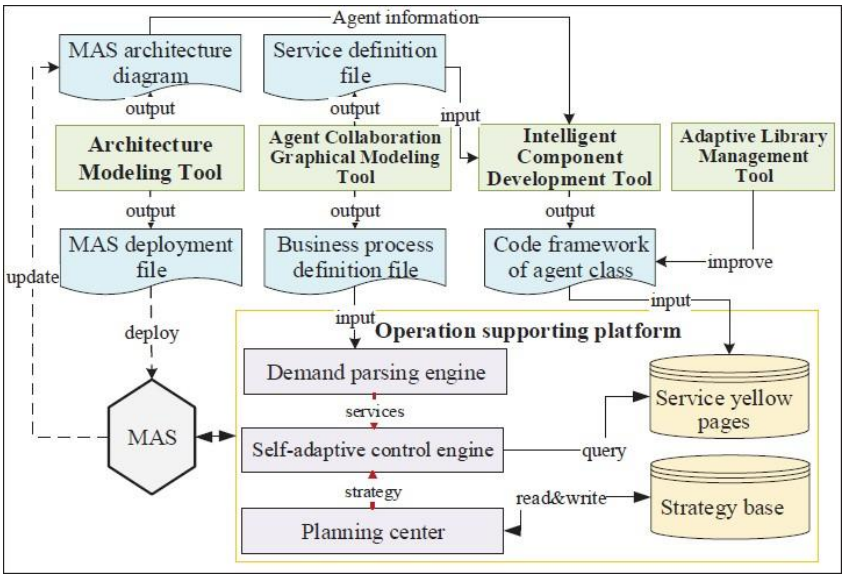
A *Target System Layer* é composta pelo *software* que não possui a capacidade adaptativa. A *Adaptation Logic Layer* controla o software a fim de permitir a adaptação, implementando duas camadas do MAPE-K Loop. Na Figura 23, o *Target System Layer* é exemplificado pelo *software* ZNN.com.

Nessa arquitetura, existem dois tipos de agentes: *Manager* e *Worker*. O *Manager* é responsável por perceber as mudanças globais do sistema (e.g. mudança de requisitos) bem como pelas mudanças que os *workers* não conseguem lidar. Para lidar com as mudanças pode-se utilizar planejamento baseado em busca ou planejamento baseado em regras, tendo a mudança como determinante para essa escolha. Além disso, é responsável pela organização do *workers*. Os *Workers* são responsável por perceber mudanças locais (e.g. problemas com components) e utilizam planejamento baseado em regras. Na Figura 23, o *Manager* é localizado na caixa de cor verde e os *Workers* estão localizados nas caixas azuis.

A arquitetura do ADAPT é demonstrada na Figura 24. Com a finalidade de atingir o desenvolvimento, em tempo de *design* do sistema, as quatro *tools*, representadas pela cor verde, interagem entre si, ao passo que, em tempo de execução, os arquivos gerados por essas *tools* são utilizados para permitir a adaptação do sistema.

A *Architecture Modeling Tool* (AMT) é usada para modelar a arquitetura do sistema. O diagrama da arquitetura gerado por essa tool, é passada para a *Intelligent Component Development Tool* (ICDT) para que os diferentes agentes do sistema sejam gerados. A *Agent Collaboration Graphical Modeling Tool* (ACGMT) permite a modelagem do processo de colaboração entre os agentes envolvidos no sistema. Por ser baseado na teoria dos agentes, essa

Figura 24 – Arquitetura do Adapt definido por Li *et al.* (2017)



Fonte: Retirado de Li *et al.* (2017)

fase recebe mais importância, haja vista que a colaboração entre esses agentes é utilizada para permitir a adaptação. Como saída dessa *tool* é gerado um arquivo de colaboração e de definição de serviço.

A *Adaptive Library Management Tool* (ALMT) é utilizada como complemento da ICDT, podendo gerar segmentos de códigos necessários pela classe do agente. O objetivo dessa complementação é diminuir o trabalho dos desenvolvedores através da geração automática de código.

Nesse trabalho, entretanto, é explicitado a necessidade de completar o desenvolvimento das funcionalidades da AMT. Desse modo, a funcionalidade de descrição da arquitetura do sistema é prejudicada. Por fim, não são descritos detalhes acerca do seu processo de adaptação, não sendo possível definir a flexibilidade de sua estrutura.

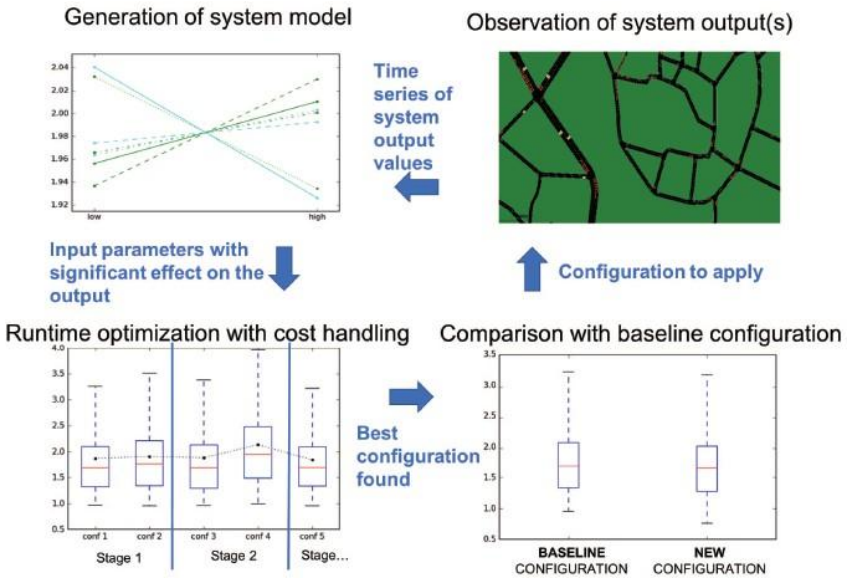
3.2.11 Adapting a System with Noisy Outputs with Statistical Guarantees

O trabalho de Gerostathopoulos *et al.* (2018) propõe um *framework* que combina otimização de tempo de execução com dados obtidos de testes estatísticos com um método para lidar com custos de más decisões de adaptação. Esse trabalho parte da incapacidade de auto-adaptação por parte das abordagens atuais perante sistemas com: (i) comportamento interno complexo, o que dificulta a modelagem, (ii) resultados ruidosos, e (iii) alto custo de más decisões de adaptação. Ainda segundo Gerostathopoulos *et al.* (2018), sistemas com essas características são difíceis e arriscados de adaptarem em tempo de execução.

Esse trabalho tem como foco a otimização do desempenho a nível de sistema. Aliado a isso, investigam as seguintes questões: (i) como construir modelos de sistema a partir de observações de saídas ruidosas do sistema; (ii) como reutilizar esses modelos para otimizar o sistema em tempo de execução, mesmo em situações recentemente encontradas; e (iii) como incorporar a noção de custo, referindo-se ao impacto negativo das decisões de adaptação, nos processos supracitados. É necessário ressaltar que saídas ruidosas referem-se a observações de métricas do sistema que podem ser usadas para a auto-adaptação, mas possuem alta variância, já o custo refere-se ao impacto negativo de más decisões de adaptação do sistema.

Para tal, a estrutura de auto-otimização implementada nesse trabalho, baseia-se em procedimentos estatísticos (e. g. análises de variância e testes binomiais). Para a otimização de tempo de execução, é empregada otimização bayesiana aliado a processos gaussianos. Essa técnica é utilizada por sua capacidade de oferecer suporte à otimização multivariada e lidar eficientemente com as variações das saídas do sistema.

Figura 25 – Visão geral da abordagem proposta por Gerostathopoulos *et al.* (2018)



Fonte: Retirado de Gerostathopoulos *et al.* (2018)

Essa abordagem é composta por três fases: (i) *Geration of system model*, (ii) *Runtime Optimization*, (iii) *Comparision with baseline configuration*. A fase (i) consiste da construção e manutenção do conhecimento necessário para a auto-otimização. Aqui é usado análise fatorial de variância a fim de processar dados brutos e criar um modelo estatisticamente relevante. Esse modelo descreve os impactos das mudanças de parâmetros de entrada na saída. Essa fase é executada durante o processo anterior à implantação do sistema, através de um simulador, até a

fase de *deploy* usando um monitoramento em tempo de execução a fim de gerar conhecimento acerca do real funcionamento do sistema. Como saída, essa fase fornece uma lista de parâmetros ou combinações de parâmetros de entrada, ordenados por níveis de impactos decrescentes e níveis de significância correspondentes.

A fase (ii) seleciona valores para os parâmetros de entrada do sistema a fim de encontrar uma configuração na qual o sistema execute melhor. As configurações testadas nessa fase são elaboradas por um otimizador que é executado durante vários estágios da abordagem. Em caso de indícios - através de testes binomiais - de que uma determinada configuração resulta em alto custo, a aplicação é interrompida e o otimizador se responsabiliza por avaliar uma próxima configuração.

A fase (iii) garante que uma nova configuração determinada na fase anterior seja implementada somente quando houver uma melhoria estatisticamente significante com relação à configuração existente. Para tal substituição, é necessário verificar se a nova configuração: (i) traz benefício ao sistema em um determinado nível de significância estatística e (ii) se o benefício é suficiente para justificar a interrupção.

Entretanto, o foco do trabalho é no processo de auto-otimização do sistema através da utilização de procedimentos estatísticos, como análise de variância. Por exemplo, para gerar um modelo de sistema é necessário observar (i) a situação em que o sistema reside, (ii) sua configuração e valores dos parâmetros de entrada, e (iii) a saída do sistema. Para cada situação, é construído um modelo que classifica os parâmetros de entrada de acordo com seu efeito na saída, do maior para o menor.

3.3 Comparação e Discussão

Como descrito no Capítulo 1 desta dissertação, o cenário de IoT é composto de sensores e atuadores embutidos no ambiente físico através dos *smart-objects*. Dada a previsão, até 2020 mais de 20 bilhões de dispositivos como esses estarão implantados ao redor do mundo. Em concordância com o que foi apresentado por Hughes (2018), os SAS são pontos importantes para que o cenário de IoT seja, de fato, alcançado.

Isto posto, é necessário avaliar quais os requisitos necessário para o desenvolvimento de um arcabouço que propicie o desenvolvimento de um SAS. Nesse sentido, o trabalho desenvolvido por Serugendo *et al.* (2007) descreve 12 requisitos que são descritos a seguir.

- **R1**: Os componentes devem ser desacoplados de modo que seja possível detectar e

responder à possíveis falhas ou indisponibilidade sem prejudicar globalmente o sistema

- **R2**: Os componentes devem ser dissociados das descrições de suas capacidades, restrições, fluxos de execução, modo de interações ou políticas.
- **R3**: Os componentes devem ser dissociados de qualquer infraestrutura de coordenação subjacente que suporte as interações.
- **R4**: Capacidade de, em tempo de execução, detectar ou monitorar eventos.
- **R5**: Capacidade de, em tempo de execução, atuar ou adaptar em diferentes níveis do sistema.
- **R6**: Disponibilidade em tempo de execução e uso de políticas de detecção ou monitoramento e atuação ou adaptação em diferentes níveis do sistema.
- **R7**: Disponibilidade em tempo de execução e uso de metas individuais e globais sob a forma de políticas.
- **R8**: Disponibilidade em tempo de execução e uso de políticas de restrições baseada no ambiente.
- **R9**: Descrição, em tempo de design, das propriedades esperadas do sistema ou dos componentes
- **R10**: Descrição, em tempo de design, dos padrões de comportamento
- **R11**: Descrição, em tempo de design, das diferentes políticas(R6, R7, R8).
- **R12**: Execução de políticas (R6, R7, R8).

Com o objetivo de simplificar a avaliação dos trabalhos presentes no estado da arte, é possível agrupar os requisitos descritos por Serugendo (SERUGENDO *et al.*, 2007). Para este trabalho de dissertação, esses requisitos foram agrupados em quatro critérios, como demonstrado na a Tabela 2 que apresenta o comparativo entre os trabalhos relacionados a este trabalho de dissertação.

O primeiro critério é o **desacoplamento** que compreende os requisitos descritos em R1, R2 e R3. Esse critério entra em concordância com a característica da heterogeneidade, presente no ambiente de IoT. O fato de um componente ser desacoplado de outras estruturas, arquiteturas ou dispositivos contribui para o desenvolvimento de sistemas para esse tipo de ambiente.

O segundo critério é a adaptação interna do sistema, que neste trabalho de dissertação é chamado de **endo-adaptação**. Isso significa a capacidade de adaptar o sistema a nível de aplicação. Aliado a este critério, está o de adaptação externa, que neste trabalho de dissertação é

Tabela 2 – Comparativo entre os Trabalhos Relacionados

Trabalhos Relacionados	Desaco-plado	Endo-Adaptação	Exo-Adaptação	Adaptação da Execução	Descrição
(MONTAGUT and MOLVA, 2005)	✓		✓		✓
(GINER et al., 2010)	✓	✓			✓
(KRUPITZER et al., 2013)	✓		✓	✓	
(ABEYWICKRAMA et al., 2014)	✓		✓		✓
(SEIGER et al., 2015)		✓	✓		✓
(WIELAND et al., 2015)		✓	✓		✓
(TSUDA et al., 2016)	✓	✓		✓	
IFTTT		✓		✓	✓
Node-Red		✓	✓		✓
(LI et al., 2017)	✓	✓	✓		
(GEROSTATHOPOULOS et al., 2018)	✓	✓		✓	

Fonte: Do autor

chamado de **exo-adaptação**, que significa, a capacidade de enviar pedidos de atuação no sistema, a nível de ambiente. A união desses critérios compreende o requisito R5.

O quarto critério é a **adaptação da execução**. Como descrito no Capítulo 2, na Seção 2.2, os SAS utilizam um loop de controle, o MAPE-K. Esse loop parte do princípio da necessidade de executar quatro tipos de passos a fim de manter a execução auto-adaptativa: Monitorar, Analisar, Planejar e Executar. Ao fim da execução desse loop, é necessária a verificação do estado do sistema. Sendo assim, não é interessante que ocorra uma sequência encadeada de execução, pois cada atuação ou adaptação altera o estado do sistema. Esse critério entra em concordância ainda com os requisitos R6, R7 e R8 e R12.

O quinto critério é a **descrição**. Esse critério diz respeito à possibilidade de descrever, em tempo de *design*, as características do sistema, como modo de execução, regras e padrões. Esse critério compreende os requisitos R9, R10 e R11.

3.4 Conclusão

Neste capítulo foram apresentados os trabalhos relacionados a este trabalho de dissertação, bem como suas características, em especial aquelas focadas no processo de adaptação

propiciado.

Seguindo o processo de revisão da literatura, foi executada uma busca por trabalhos que tratassem sobre especificação de adaptações em ambientes que possuíssem sensoriamento e/ou atuação, adaptações baseadas em planos de execução ou *workflows* e mecanismos ou *frameworks* que permitissem a especificação e execução de adaptações. Com essa busca, foram encontrados 11 trabalhos relacionados com o que é proposto nesta dissertação. Esses trabalhos foram elencados seguindo os seguintes critérios de comparação: Desacoplamento, Endo-Adaptação, Exo-Adaptação, Adaptação da Execução e Descrição.

Após o processo de comparação entre os critérios elencados e as soluções encontradas, foi possível perceber que nenhum dos trabalhos consegue contemplar todos os critérios apresentados ou os implementam apenas em partes.

4 SUCCEED

Neste capítulo é descrita a proposta desta dissertação, um *framework* para auxiliar o desenvolvimento de sistemas auto-adaptativos em ambientes IoT, o *decoupled SUpport meChanism for CrEating and Executing workflows* (SUCCEEd). Este capítulo está dividido da seguinte forma: a Seção 4.1 apresenta a visão geral do SUCCEEd; a Seção 4.2 detalha a arquitetura do SUCCEEd, discutindo os componentes envolvidos no processo de auto-adaptação; a Seção 4.3 apresenta e descreve os elementos presentes no SUCCEEd, sua classificação e seu uso; a Seção 4.4 descreve o processo de orquestração das adaptações que é proposto pelo SUCCEEd. Por fim, a Seção 4.5 apresenta a conclusão este capítulo.

4.1 Visão Geral

O SUCCEEd [1] é um *framework* baseado em Java e foi desenvolvido com o objetivo de permitir a descrição e a execução de adaptações através de uma estrutura de *workflows*. Embora seja um *framework* para dar suporte ao desenvolvimento, a sua atuação está localizada no processo de execução, de acordo com o loop de controle MAPE-K apresentado na Seção 2.2.

A instanciação dos *workflows*, em tempo de execução, pode ocorrer de duas maneiras: simples ou baseada em regras. A instanciação simples ocorre quando o *workflow* é executado a partir de uma solicitação feita por um determinado elemento do sistema. Este tipo de execução é a mais rígida em termos de auto-adaptação. A rigidez provêm do fato de que, com essa execução, as adaptações ocorrem em uma sequência e não levam em consideração que cada adaptação altera o estado do sistema.

A instanciação baseada em regras é aquela que leva em consideração os estados do sistema. Neste tipo de execução, o desenvolvedor especifica determinados estados, através de regras, e possíveis adaptações que podem ser executadas através do cumprimento destas regras. Com este tipo de execução, um *workflow* ou uma adaptação pode ser facilmente trocada em tempo de execução, considerando uma mudança de estado. Esta troca de *workflows* é um passo do MAPE-K Loop, descrito na Seção 2.2. Uma camada de inteligência ou um outro sistema deve constantemente analisar o estado do sistema e decidir se o *workflow* atual vai levar o sistema para um próximo estado válido. Esta camada não faz parte do *framework* proposto neste trabalho. Deste modo, o tipo de execução baseado em regras é a mais flexível em termos

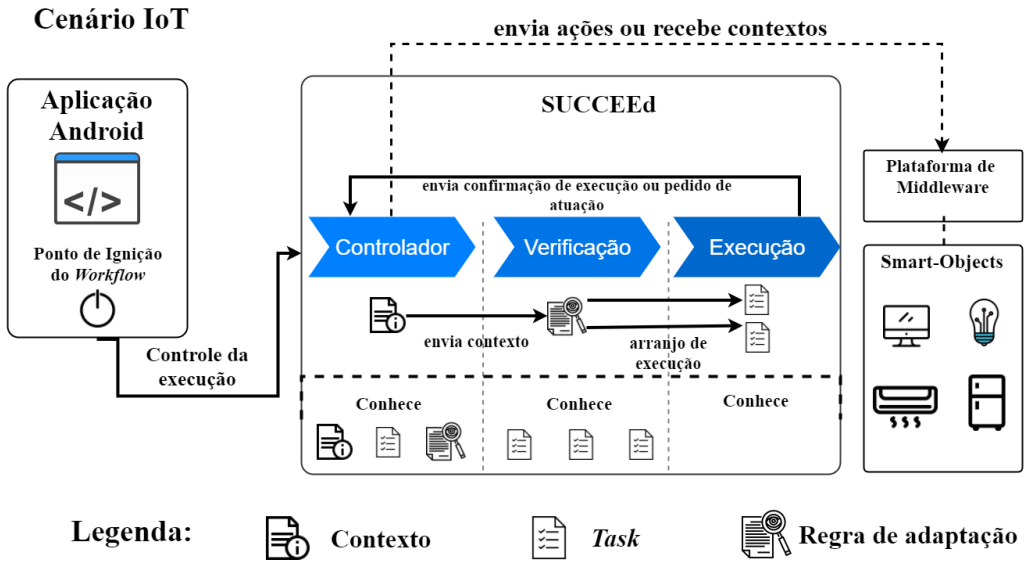
[1] https://github.com/Belmondo/SUCCEEd

de auto-adaptação da aplicação, pois considera os estados do sistema em tempo de execução. Essa forma de execução é melhor abordada na Seção 4.3.

Sendo assim, o cerne do SUCCEEd é propiciar para os desenvolvedores de sistemas auto-adaptativos para IoT, um mecanismo de execução de adaptações que seja capaz de selecionar, em tempo de execução, as adaptações mais adequadas para um determinado estado do sistema ou contexto desejado. Isso reduz a responsabilidade de instanciar, por parte dos desenvolvedores, as adaptações em tempo de desenvolvimento do sistema.

Para demonstrar a visão geral do funcionamento do SUCCEEd, uma aplicação Android executando em um cenário IoT é utilizada como exemplo (Figura 26). A partir de um momento determinado pelos desenvolvedores, que aqui será chamado de **Ponto de Ignição**, o SUCCEEd passa a controlar o fluxo de execução da aplicação, fundamentado nas estruturas definidas pelo desenvolvedor, que podem ser: contexto, adaptação, regras de adaptação e *gateways*. A definição do contexto depende da plataforma de *middleware* que o desenvolvedor escolher, sendo de interesse do SUCCEEd apenas o conhecimento e comunicação com a plataforma de *middleware* escolhido. A adaptação pode ser qualquer ação, interna do sistema, no ambiente ou externa ao ambiente, descrita pelo desenvolvedor. A regra de adaptação é uma estrutura definida que auxilia no processo de adaptação, verificando se um determinado estado do sistema necessita de uma adaptação. Os *gateways* são estruturas que podem ser descritas para indicar o modo de execução, usando operadores OU e E da lógica proposicional.

Figura 26 – Visão geral do SUCCEEd



Fonte: Do Autor

A começar do instante em que o SUCCEEd passa a controlar o fluxo da execução da aplicação, o seu funcionamento consiste em três etapas, conforme ilustrado na Figura 26: Controlador, Verificação e Execução.

O Controlador é responsável por gerenciar a execução do SUCCEEd, trabalhando como uma *engine*. É de sua responsabilidade conhecer todas as estruturas envolvidas na adaptação. O seu funcionamento ocorre da seguinte maneira: inicialmente observa o estado do sistema, tendo como base os contextos descritos em tempo de desenvolvimento. Ao tomar conhecimento do estado do sistema, informa para a etapa de verificação. Além disso, sempre que a execução de uma adaptação é findada, ele deve voltar ao controle da execução. Por ser a etapa responsável por conexão com a plataforma de *middleware*, então é de sua responsabilidade também enviar pedidos de atuação para os dispositivos necessários em uma adaptação.

A etapa de Verificação tem como responsabilidade entender, através daquilo que foi definido pelos desenvolvedores, se o estado que foi recebido demanda alguma adaptação, seja esta interna ou externa ao sistema. Para isso, deve conhecer as estruturas de adaptação e o modo como essas adaptações devem ser executadas.

A etapa de execução é a mais simples e tem como responsabilidade executar a adaptação que foi descrita pelos desenvolvedores.

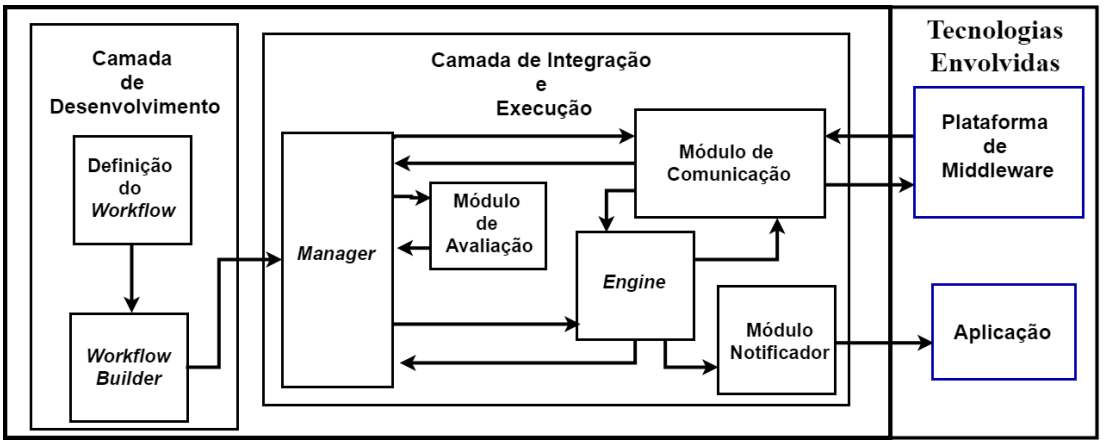
Cada uma dessas etapas conhece uma estrutura que foi modelada pelo desenvolvedor. O Controlador, por atuar gerenciando toda a execução do processo, deve conhecer o contexto, as regras de adaptação e as *tasks*. A etapa de verificação não necessita conhecer os contextos e as regras de adaptação, pois essa etapa só é acionada no momento em que uma ou mais regras de adaptação estão satisfeitas com um determinado estado do sistema e por esse motivo, a etapa de verificação necessita conhecer apenas as *tasks* que estão associadas com esta(s) regra(s) satisfeita(s). A etapa de execução é a adaptação propriamente dita, e portanto necessita conhecer somente a si.

4.2 Arquitetura

O SUCCEEd foi projetado em uma arquitetura extensível, baseada em componentes, com o objetivo de melhorar o desacoplamento. A Figura 27 ilustra as camadas que fazem parte da arquitetura do SUCCEEd: a Camada de Desenvolvimento e a Camada de Integração e Execução. Cada camada possui subcamadas que serão chamadas de módulos e que são descritas a seguir.

A Camada de Desenvolvimento é a camada mais aparente, do ponto de vista dos

Figura 27 – Arquitetura do SUCCEEd



Fonte: Do Autor

desenvolvedores por ser aquela com a qual há mais interação. Esta camada oferece suporte aos desenvolvedores, com as estruturas necessárias para a descrição do processo de adaptação. O primeiro módulo desta camada é o **Módulo de Definição do *Workflow***. Este módulo é o responsável por fornecer as estruturas de definição da adaptação. Portanto, é de sua responsabilidade conhecer as estruturas envolvidas, como regras de adaptação, descrição de adaptação, modos de execução, e estados do sistema. O Módulo ***Workflow Builder*** é um módulo auxiliar, pois tem como responsabilidade organizar as estruturas definidas no Módulo de Definição. De modo prático, representa o compilado de elementos modelados, sendo uma forma primitiva de um orquestrador das adaptações, entretanto, apenas conhecendo esses elementos.

A **Camada de Integração e Execução** é responsável, na prática, pelo processo de auto-adaptação do SUCCEEd. Esta camada organiza a execução do *workflow*, verifica regras e inicia as adaptações. Esta camada é ainda responsável pela comunicação externa com a plataforma de *middleware*. É importante salientar que todas estas estruturas provêm primariamente da Camada de Desenvolvimento, mais especificamente do Módulo de Definição, portanto, é necessário descrever neste módulo, a comunicação com uma plataforma de *middleware*, caso ocorra.

O primeiro módulo da **Camada de Integração e Execução** é o **Módulo *Manager***. Este módulo é responsável por organizar a execução da estrutura provinda do ***Workflow Builder***. Por ser o módulo que organiza a execução das adaptações, este módulo tem ciência de todas as estruturas que podem executar, portanto, é também responsável por conhecer os demais módulos e distribuir as responsabilidades dentro do *framework*.

O **Módulo de Avaliação** é responsável por avaliar os estados do sistema, que lhe são passados pelo Módulo *Manager*. Deste modo, ele pode informar ao Manager se há a necessidade

de alguma adaptação ou adaptações para aquele estado.

O **Módulo *Engine*** recebe do *Manager* a ordem de execução de uma determinada adaptação. É através deste módulo que o *workflow* é, de fato, caracterizado e executado. Por ser o módulo responsável pela execução das adaptações, caso uma destas adaptações demande ajustes em um dispositivo externo ao sistema, é então de responsabilidade da *Engine* comunicar ao Módulo de Comunicação o ajuste desejado. Ainda, é responsável por coordenar as adaptações, quando estas adaptações são executadas em paralelo ou de maneira exclusiva.

O **Módulo de Comunicação** é responsável por dar suporte à comunicações externas à aplicação. Este módulo interage diretamente com três estruturas diferentes: (i) Com o Módulo *Manager*, a fim de informar mudanças de estados do sistema, (ii) Com o Módulo *Engine* com o objetivo de informar se o pedido de ajuste externo foi executado e (iii) Com a Plataforma de *Middleware*. Deste modo, ele conhece com qual plataforma está se comunicando e envia pedidos de ação para serem executadas nos dispositivos, bem como, é de sua responsabilidade observar, caso o estado do sistema se altere e informar ao *Manager*.

O **Módulo Notificador** é um módulo dependente da *Engine*, sendo responsável pela comunicação com a aplicação, quando a adaptação demandar um envio de informações na interface ou ajustes na aplicação. É de sua responsabilidade ainda, a interação do usuário, quando a adaptação assim exigir, retornando isto ao *Manager*, pois toda adaptação altera o estado do sistema.

4.3 Elementos do *Framework*

O SUCCEEd tem seus elementos divididos com o objetivo de garantir a correta distribuição de responsabilidades dentro do próprio código e, assim, garantir a coesão do framework. Os elementos do framework podem ser divididos em três categorias: (i) Os Elementos Modeláveis, (ii) Os Elementos Semi-Modeláveis e, (iii) Os Elementos Não-Modeláveis. Cada uma dessas categorias e seus elementos são descritos a seguir.

4.3.1 Elementos Modeláveis

Os elementos presentes nesta categoria são aqueles os quais os desenvolvedores interagem através de extensão e modelagem destas extensões. Portanto, são dependentes, em sua maioria, da modelagem dada pelos desenvolvedores. Estes elementos são assim apresentados

pois, são aqueles responsáveis por representar as particulares dos sistemas desenvolvidos, i.e., as regras de adaptação, as adaptações e a sequência do fluxo destas adaptações e, cada uma destes deve ser flexível para tal uso. Por esta razão, os elementos presentes nesta categoria possuem um esboço mínimo, mas sua principal implementação é de responsabilidade dos desenvolvedores.

Para o SUCCEEd, existem três elementos que refletem as particularidades do sistema desenvolvido, que são: (i) Regras, que representam as regras de adaptação, (ii) Tasks, que representam as adaptações, e (iii) Gateways, que representam a sequência do fluxo destas adaptações. Cada um destes elementos é melhor descrito nas subseções a seguir.

4.3.1.1 Regras

Uma vez que o SUCCEEd é responsável pelas adaptações com base em estados do sistema, há a necessidade de prover um meio de raciocínio que permita inferir resultados com base nesses estados. Como descrito por Perera et al (PERERA *et al.*, 2013) existem algumas técnicas que podem suprir estas necessidades, sendo divididas em seis categorias: aprendizagem supervisionada, aprendizagem não supervisionada, regras, lógica fuzzy, raciocínio com base em ontologias e raciocínio probabilístico.

Cada técnica supracitada possui suas características e para este trabalho a técnica escolha foi a baseada em regras. O motivo desta escolha se deu por ser o método mais simples e direto de prover raciocínio, assemelhando-se ao formato if-then-else (ALEGRE *et al.*, 2016). Além disso, como sustentado por Perera et. al (PERERA *et al.*, 2013), as regras podem desempenhar um papel significativo em IoT, exatamente por ser a maneira mais fácil de modelar o raciocínio humano.

No SUCCEEd as regras são de fundamental importância, uma vez que são elas as responsáveis por avaliarem se determinado estado do sistema demanda adaptação e, consequentemente, por indicar ao orquestrador do SUCCEEd qual a adaptação demandada. Assim sendo, as regras são estruturas muito particulares de cada sistema e não é possível generalizar todo o conteúdo das regras. Por exemplo, para determinado sistema, a regra que avalia a temperatura é satisfeita apenas quando o valor deste contexto for maior que 20^{o} C, ao passo que para outro sistema, essa regra apenas é satisfeita com valores inferiores a 17^{o} C. Com essas características supracitadas - importância e flexibilidade - foi necessário desenvolver uma estrutura que possa ser implementada e flexibilizada pelos desenvolvedores e ao mesmo tempo mantivesse uma cerne estático para SUCCEEd.

Para atingir este objetivo, as estrutura de regras foi desenvolvida como uma classe abstrata, a fim de que os desenvolvedores possam estendê-la e, assim, adaptá-la às suas necessidades. A rigidez se encontra no retorno do resultado do método principal desta classe. O pseudocódigo 1 apresenta uma visão destas particularidades.

Algoritmo 1: Pseudocódigo de uma regra para temperatura menor do que 17º C

```
Entrada: Temperatura Atual
Saida: Verdadeiro ou Falso
início
    Atribua os valores dos parâmetros;
    temperaturaAtual ← TemperaturaAtual;
    temperaturaIdeal ← 17;
    se (temperaturaAtual < temperaturaIdeal) então
    Retorne Verdadeiro;
    senão
    Retorne Falso;
    fim se
fim
```

Como é possível observar no pseudocódigo 1, ao finalizar a análise da regra, a própria classe deve retornar ao orquestrador se, a avaliação foi positiva ou negativa. Caso negativo, esta regra não foi satisfeita, e nenhuma adaptação é demandada. Caso positiva, uma estrutura task ou gateway deve ser iniciada. Essas estruturas serão explicadas a seguir.

4.3.1.2 Task

A Task representa uma determinada adaptação desejada. Por ser tratar de um elemento de natureza própria do sistema modelado, o SUCCEEd fornece apenas uma base para a descrição desta adaptação. Assim sendo, cada adaptação desejada deve ser refletida no código através de uma classe que estenda a classe abstrata Task.

É possível perceber que para cada adaptação desejada será criada uma classe que deve representar aquela adaptação. Nitidamente ocorrerá um aumento no número de classes do projeto, entretanto este aumento é positivo, isto pois, esta modularização de adaptações evidencia a responsabilidade única (citar SRP) de uma classe. Em adição, caso as políticas da empresa sejam alteradas, tal modularização tornaria mais simples o processo de manutenção do sistema, uma vez que apenas uma classe ou poucas classes seriam afetadas.O pseudocódigo 2 demonstra

o processo de implementação de uma task

Algoritmo 2: Pseudocódigo da Criação de uma Task - Adaptação

Entrada: Dependente da Adaptação

início

    Estensão da classe abstrata Task;
    Inserir código da adaptação desejada dentro do método run();

fim

4.3.1.3 Gateways

Os Gateways são recursos usados para controlar como a sequência de execução do fluxo ocorre dentro do processo definido. É importante ressaltar que, por este motivo, se um fluxo não necessita ser controlado então um gateway se torna desnecessário. A principal diferença de um Gateway para as demais estruturas é que ela não representa uma ação sendo executada e tão pouco uma forma de representação de conhecimento. No SUCCEEd, os gateways implementados são: Exclusivo, Inclusivo e Paralelo.

- O Gateway Exclusivo permite a criação de caminhos alternativos dentro do fluxo de execução das adaptações, habilitando que apenas um dos caminhos possa ser seguida, tendo por base alguma decisão em um ponto específico da execução do processo. Por teoria, essa decisão possui um conjunto definido de respostas possíveis no qual cada resposta está associada a uma expressão de condição associada ao fluxo de execução.
- O Gateway Inclusivo permite criar fluxos alternativos de execução, bem como paralelos. A diferença deste gateway para o exclusivo reside no fato de que todas as decisões definidas são avaliadas e mesmo que uma decisão seja tomada, não ocorre a exclusão das demais decisões. Deste modo, todos os caminhos definidos são independentes e pode haver combinações destes caminhos em tempo de execução, podendo ocorrer desde a execução de um caminho até todos os caminhos definidos.
- O Gateway Paralelo permite sincronizar ou combinar fluxos de execução paralelos ou Tasks. Para isso, um caminho é criado sem a necessidade de decisões. Esse gateway pode ser usado também para garantir que a execução da próxima Task só ocorra quando caminhos paralelos estejam finalizados, sendo um ponto de convergência de fluxos. Diferente do gateway anterior, este executa somente duas tasks em paralelo. Apesar disso, é um gateway extensivo, permitindo a extrapolação deste valor.

4.3.2 Elementos Semi-Modeláveis

Os elementos presentes nesta categoria são aqueles que são pouco modeláveis pelos desenvolvedores, sendo necessário apenas definir algumas características para a melhor execução do framework. Isto acontece porque tais elementos apresentam características gerais quando comparadas aos demais elementos.

4.3.2.1 Conexão Externa

O elemento de conexão externa permite que a execução possa fazer acesso externo ao próprio dispositivo onde o sistema está sendo executado. Este elemento é capaz de fazer conexões a servidores através dos tipos de conexão *GET* e *POST*, de enviar comandos, bem como de trazer resultados destes comandos. Para este elemento, os desenvolvedores precisam apenas definir o tipo de conexão, o endereço do servidor e o comando desejado. Este elemento é útil principalmente para envios de mensagens e comando a *smart-objects* que estejam conectados por meio de algum servidor.

4.3.2.2 Workflow

O elemento *workflow* é o elemento principal do *framework* SUCCEEd. Este elemento atua como orquestrador de todo o processo de adaptação do sistema, seja esta execução baseada em regras ou apenas sequenciais. Portanto, em tempo de execução da aplicação, este elemento atua como a *engine* do processo de verificação e execução. Para isso, os desenvolvedores precisam informar as regras e adaptações criadas. Por fim, é necessário definir qual o tipo de execução, sequencial ou baseada em regras, e instanciar o objeto workflow no local do sistema a qual o framework será responsável.

O Pseudocódigo 3 exemplifica o uso para uma execução sequencial. Como é possível verificar, é necessário apenas informar a lista de adaptações ao objeto instanciado e, por fim, dar

início ao fluxo sequencial.

Algoritmo 3: Pseudocódigo da instanciação de um workflow sequencial no sistema

```
Entrada: Lista de Adaptações
início
    Declaração e Instanciação do objeto Workflow;
    FluxoSequencial ← listaDeAdaptacoes
    Iniciar FluxoSequencial
fim
```

O Pseudocódigo 4 exemplifica o uso para uma execução baseada em regras. Esta execução se difere da anterior especialmente no que diz respeito à entrada de dados para o funcionamento do SUCCEEd. Para o funcionamento baseado em regras, é necessário que o desenvolvedor explicite a lista dos Elementos do processo, i.e., Regras de Adaptação, Tasks e Gateways. Após isso, é necessário apenas iniciar o fluxo baseado em regras.

Algoritmo 4: Pseudocódigo da instanciação de um workflow baseado em regras no sistema

```
Entrada: Lista de Elementos (Regras, Tasks e Gateways)
Saida: Verdadeiro ou Falso
início
    Declaração e Instanciação do objeto Workflow;
    FluxoBaseadoemRegras ← listaDeElementos
    Iniciar FluxoBaseadoemRegras;
fim
```

4.3.3 Elemento Não-Modelável

O único elemento presente nesta categoria é o elemento Token. Este elemento é de uso exclusivo do SUCCEEd, e não interage com os desenvolvedores. Este elemento é utilizado apenas nas execuções sequenciais e sua função é permitir que a próxima execução tenha início apenas quando a anterior esteja finalizada. Isso garante que a execução possa seguir o fluxo de maneira mais precisa. Além disso, ele tem a função de informar, caso uma execução não seja finalizada ou sequer possa ser iniciada. Esta última funcionalidade se mostra importante em ambientes IoT, já que os dispositivos podem não estar presentes ou indisponíveis no momento da execução.

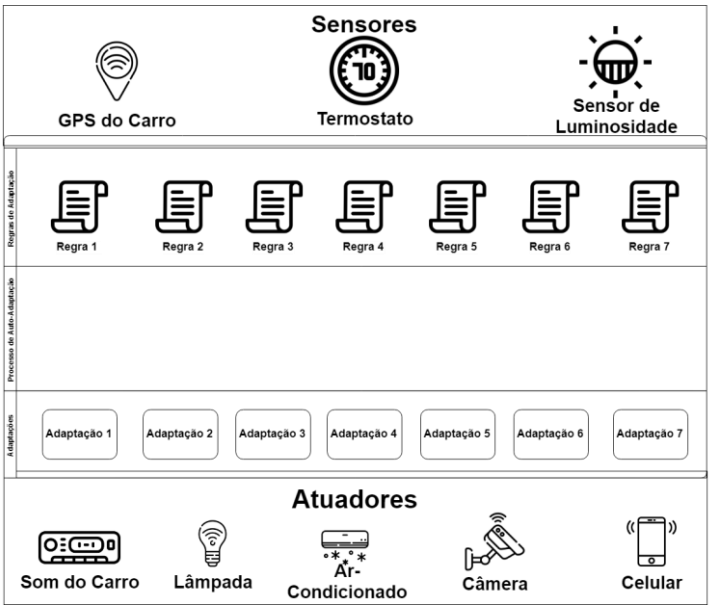
4.4 Processo de Adaptação

O processo de adaptação utilizado pelo SUCCEEd tem como objetivo garantir que, em tempo de execução, as adaptações sejam selecionadas e executadas. Deste modo, não é necessário sobrecarregar o código do sistema de estruturas de controle de fluxo, como *if* e *switch*. Para melhor demonstrar o processo de adaptação proposto nesta dissertação, um cenário IoT é descrito a seguir.

O usuário está saindo do prédio onde trabalha para a sua residência. Ele entra no carro e começa a dirigir. Neste momento, o sistema coloca no som do carro a música "*Don't stop Believing*" da Banda Journey, a favorita do usuário. Ainda, ao perceber a movimentação no veículo, as notificações do *smartphone* do usuário, sejam de redes sociais ou ligações, são silenciadas. A única exceção são para ligações de familiares mais próximos, como pais, esposa e filho. Durante o trajeto, o sistema toma ciência dos contextos presentes na residência, como o da luminosidade e a temperatura local, por exemplo, e adapta-os a fim de manter uma luminosidade e temperatura condizentes. Com a proximidade do usuário de sua residência, as luzes externas são acessas e a câmera externa vira para o sentido de onde o veículo se aproxima e continua seguindo o carro, até que ele esteja completamente dentro da residência.

Para identificar melhor cada passo e adaptação, o cenário é detalhado em momentos e são demonstrados nas Figuras 28, 29, 30, 31, 32 e 33. No primero momento do cenário, nenhuma adaptação é requerida. Deste modo, o sistema ainda não conhece quais ações deve tomar e o *workflow* que descreve a sequencia de adaptações ainda não existe, como demonstrado na Figura 28.

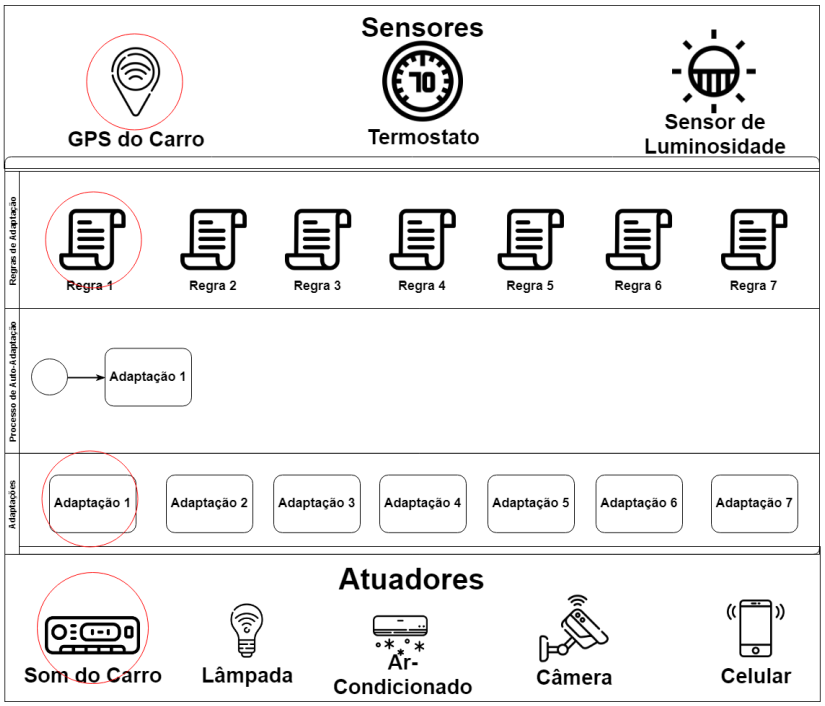
Figura 28 – Primeiro Momento - Nenhuma adaptação é requerida



Fonte: Do Autor

O segundo momento corresponde ao instante em que o usuário **entra no carro e começa a dirigir**. À princípio, a **Regra 1** é satisfeita e toma como adaptação a **Adaptação 1**, que diz respeito à ação de ligar o som do carro na música favorita do usuário, como mostrado na Figura 29.

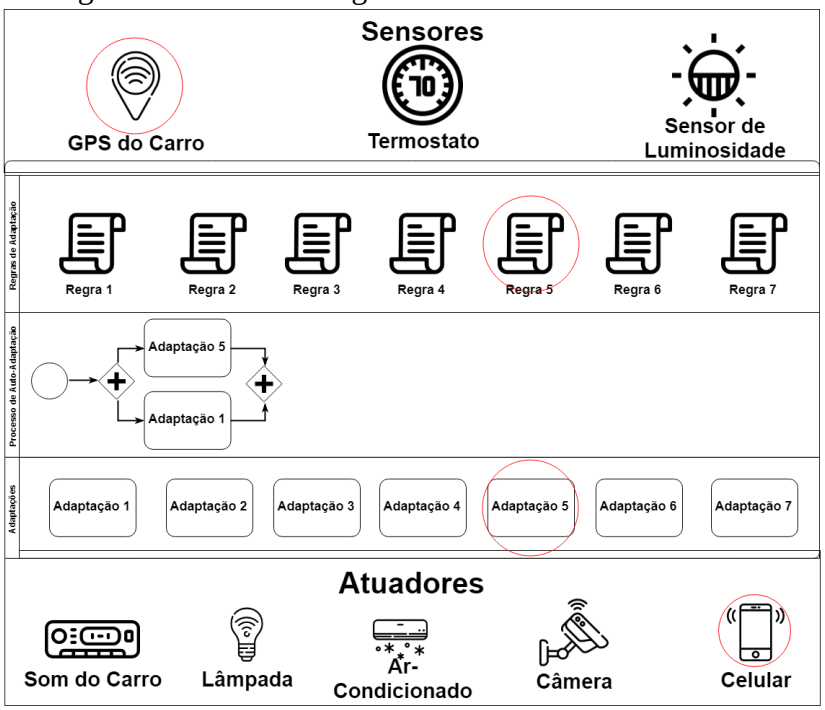
Figura 29 – Segundo Momento - Regra 1 satisfeita



Fonte: Do Autor

Entretanto, duas regras são satisfeitas como a mesma condição: “ **entrar no carro e se locomover** ”. Considerando que duas regras são satisfeitas com uma mesma condição, duas adaptações são selecionadas e ambas executam ao mesmo tempo, como mostrado na Figura 30. Esta segunda regra satisfeita é a **Regra 5** e a adaptação referente é a **Adaptação 5** e diz respeito ao silenciamento de notificações, exceto dos familiares do usuário. Com esse fato, o próprio fluxo é alterado em tempo de execução do sistema.

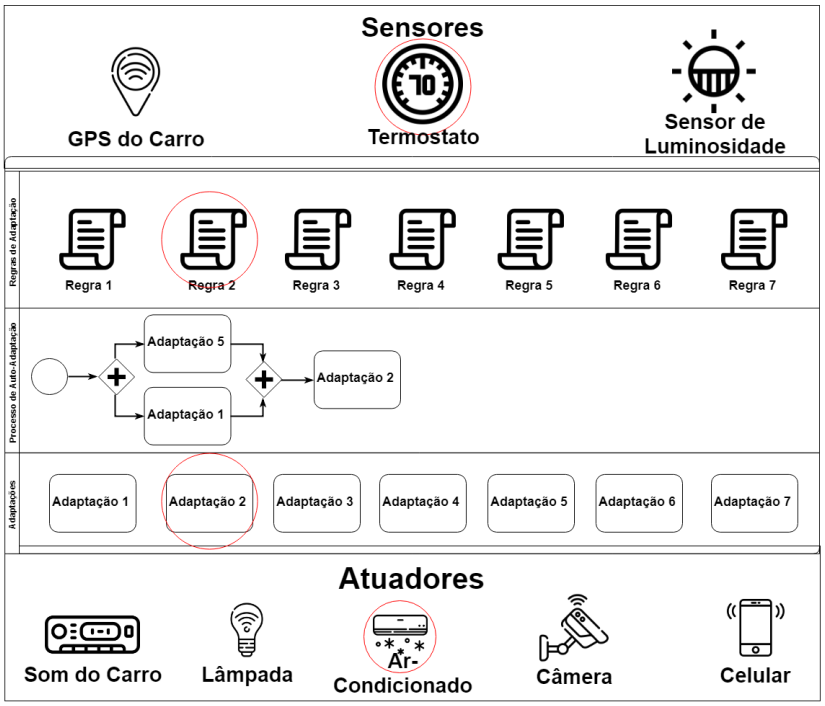
Figura 30 – Segundo Momento - Regra 5 satisfeita



Fonte: Do Autor

O terceiro momento diz respeito à verificação da temperatura da residência. Essa verificação satisfaz a **Regra 2** e instancia como adaptação a **Adaptação 2**, acionando o ar-condicionado e regulando sua temperatura, como demonstrado na Figura 31.

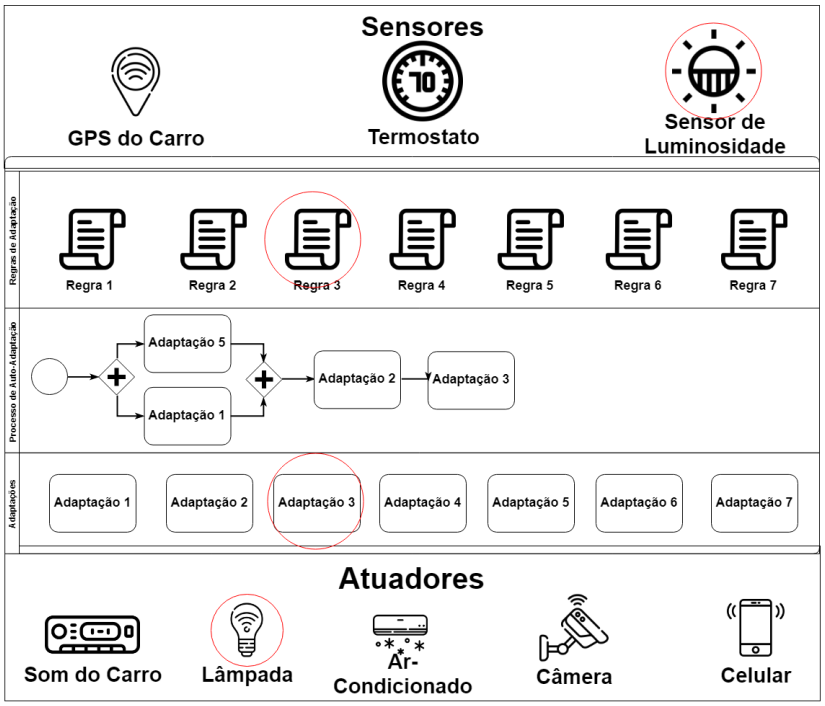
Figura 31 – Terceiro Momento - Regra 2 satisfeita



Fonte: Do Autor

O quarto momento diz respeito aquele em que o sistema toma ciência da luminosidade da casa. Com a ciência disto, e tendo uma regra (**Regra 3**) para modificação da luminosidade, uma adaptação é instanciada e executada (**Adaptação 3**), como demonstrado na Figura 32.

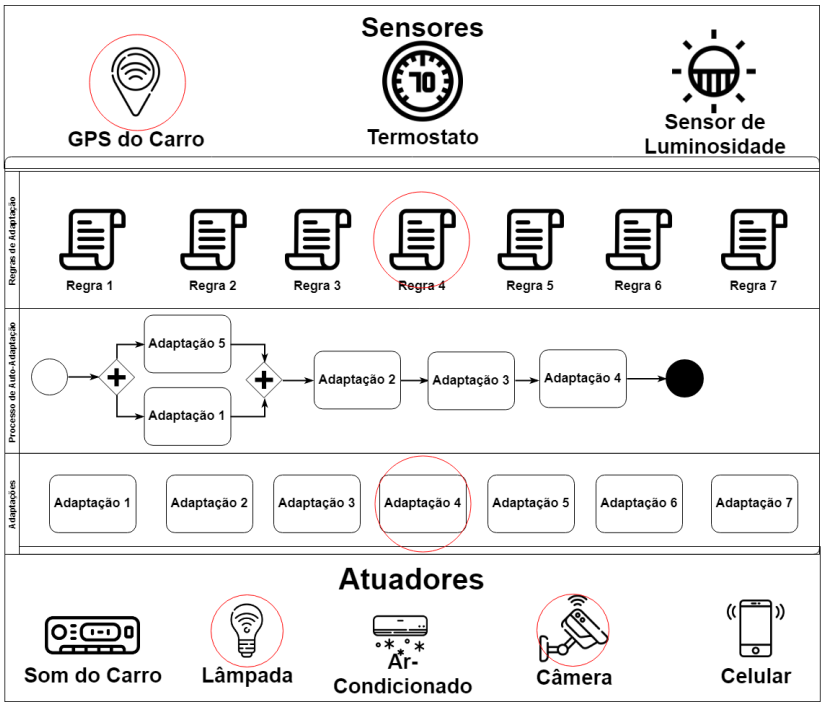
Figura 32 – Quarto Momento - Regra 3 satisfeita



Fonte: Do Autor

É importante salientar, que apesar das Adaptações 3 e 4 executarem no mesmo ambiente, elas não devem ser executadas em paralelo. Isto porque as regras que as instancia são dependentes de sensores diferentes, portanto, são dois estados diferentes do sistema. Neste caso, a execução poderia acontecer na ordem como descrito ou em outra ordem (Adaptação 4 e depois a Adaptação 3), o que vai determinar é a maneira como a regra foi implementada ou questões relacionadas aos sensores, como por exemplo: se estão presentes no ambiente ou se retornam o resultado em menor tempo.

Figura 33 – Quinto Momento - Regra 4 satisfeita



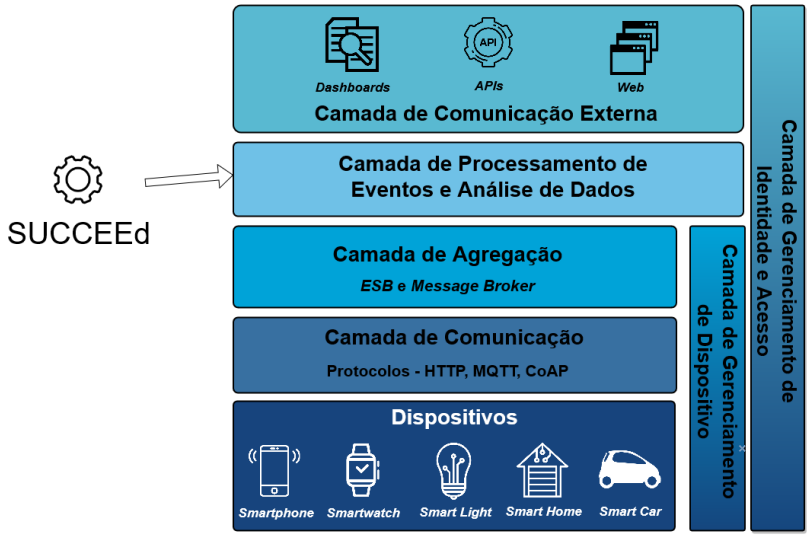
Fonte: Do Autor

Por fim, o último momento é quando há a proximidade do usuário de sua residência. Neste momento, a Regra 4 é satisfeita com os dados que o GPS envia. A Adaptação 4 é satisfeita e é responsável por acender as lâmpadas externas da casa e as câmeras de segurança. Neste caso, apesar de apenas a Regra 4 ter sido satisfeita, a adaptação 4 foi definida como uma ação em dois dispositivos. Por isso, tanto a lâmpada quanto a câmera são controladas em uma mesma adaptação, como mostrado na Figura 33.

4.5 SUCCEEd, as Arquiteturas de Referência e o MAPE-K Loop

Após demonstrar o processo de adaptação, é possível localizar o SUCCEEd nas arquiteturas apresentadas na Seção 2 desta dissertação. Em relação à Arquitetura de Referência de IoT, o SUCCEEd se encontra na Camada de Processamento de Eventos e Análise de Dados, pois esta camada prevê o processamento e a reação de possíveis eventos provenientes da camada anterior. A Figura 34 demonstra a localização do SUCCEEd na Arquitetura de Referência da WSO2.

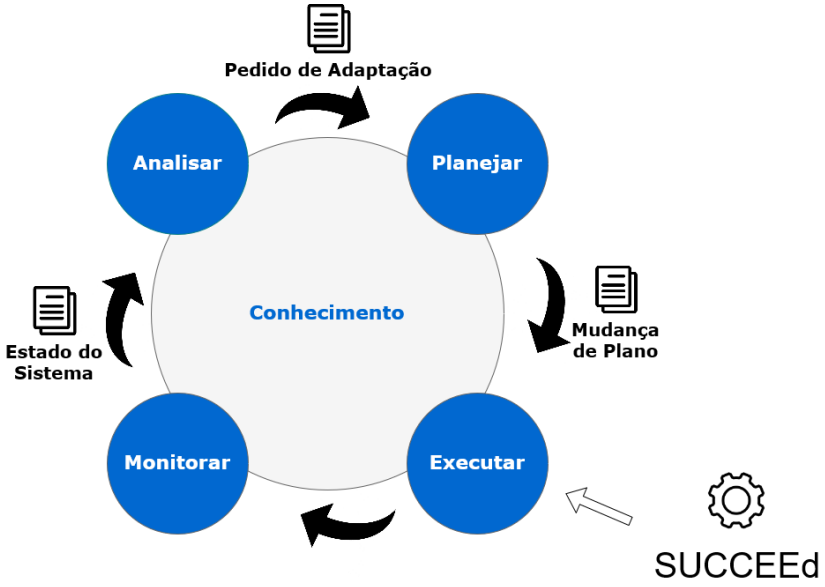
Figura 34 – Localização do SUCCEEd na Arquitetura de Referência da WSO2



Fonte: Adaptado de Fremantle (2015)

Em relação ao MAPE-K Loop, o SUCCEEd se encontra na camada de execução, pois esta camada é responsável por executar as ações planejadas na fase anterior. Apesar de o foco estar na camada de execução, os elementos desenvolvidos para o *framework* foram desenvolvidos de modo a permitir uma camada de comunicação com a fase de Planejamento, desde que esta fase seja baseada em regras. A Figura 35 demonstra a localização do SUCCEEd no MAPE-K Loop.

Figura 35 – Localização do SUCCEEd no MAPE-K Loop



Fonte: Adaptador de IBM (2005)

4.6 Conclusão

Este capítulo apresentou o SUCCEEd, um *framework* de suporte ao desenvolvimento e execução de adaptações para ambientes IoT. O foco do SUCCEEd é a orquestração das adaptações descritas se baseando em regras definidas e no estado atual do sistema em execução.

O SUCCEEd têm seus elementos divididos com o objetivo de garantir a coesão do *framework*. Os elementos presentes são divididos em três categorias: (i) Os Elementos Modeláveis, (ii) Os Elementos Semi-Modeláveis e, (iii) Os Elementos Não-Modeláveis. Os Elementos Modeláveis são aqueles responsáveis por representar as particulares dos sistemas desenvolvidos, i.e., as regras de adaptação, as adaptações e a sequência do fluxo destas adaptações. Portanto são os mais dependentes da modelagem dada pelos desenvolvedores. Nesta categoria estão as regras de adaptação, as adaptações e os *gateways*. Os elementos semi-modeláveis são aqueles que necessitam apenas receber algumas definições para melhor executar o processo. Nesta categoria estão os elementos conexão externa e *workflow*. Por fim, os Elementos não-modeláveis são aqueles usados exclusivamente pelo próprio *framework*. Nesta categoria está o elemento *token*.

O processo de adaptação do SUCCEEd ocorre através de uma estrutura de *workflow* dinâmico. Isso significa que o *workflow* é conhecido apenas ao fim da execução, pois, por sua capacidade de executar as adaptações com base no estado do sistema, o fluxo de adaptações é alterado em tempo de execução.

Em relação à Arquitetura de Referência de IoT, o SUCCEEd se encontra na Camada de Processamento de Eventos e Análise de Dados, pois esta camada prevê o processamento e a reação de possíveis eventos provenientes da camada anterior. Ao passo que, em relação ao MAPE-K Loop, o SUCCEEd se encontra na camada de execução, pois esta camada é responsável por executar as ações planejadas na fase anterior.

O próximo capítulo apresenta a avaliação do *framework* proposto neste capítulo através da implementação de uma prova de conceito, avaliação de acoplamento e avaliação da qualidade da auto-adaptação.

5 AVALIAÇÃO

Neste capítulo é apresentada a avaliação conduzida para verificar se os resultados alcançados neste trabalho atingem os objetivos esperados. Esta avaliação foi particionada em três etapas: uma Prova de Conceito, um estudo do Acoplamento e um estudo da Qualidade da Adaptação.

A Seção 5.1 apresenta a prova de conceito desenvolvida para demonstrar o funcionamento do SUCCEEd, incluindo as definições de adaptação e regras, e detalhes do funcionamento da adaptação. A Seção 5.2 apresenta o estudo de acoplamento, a métrica utilizada e os resultados. A Seção 5.3 detalha a avaliação da qualidade da auto-adaptação promovida pelo SUCCEEd. A Seção 5.4 apresenta a ameaça à validade dessa avaliação e a Seção 5.5 traz a conclusão deste capítulo.

5.1 Prova de Conceito

O objetivo da Prova de Conceito (do inglês, *Proof of Concept* (PoC))[1] é ilustrar o processo de auto-adaptação em tempo de execução, considerando as capacidade de sensoriamento e atuação. Como descrito no capítulo 4, o SUCCEEd é um *framework* baseado na linguagem Java e, portanto, pode ser estendido para aplicações que também sejam baseadas nesta linguagem. Para o SUCCEEd, o nicho principal é IoT, logo, para essa PoC foi implementado um assistente virtual (AV) em Android. Para essa PoC, além desta aplicação, foi utilizada a plataforma Arduíno com o objetivo de permitir a comunicação com um ar-condicionado e, por fim, três lâmpadas inteligentes.

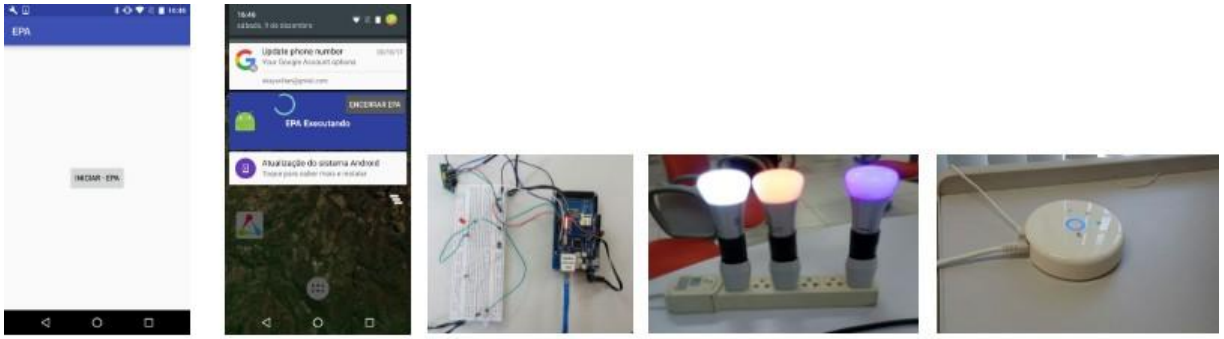
No cenário proposto, ao chegar ao local de trabalho, o usuário recebe um e-mail de confirmação de ponto e o ar condicionado de sua sala é ligado. Quando o usuário entra no prédio e se aproxima da sala, as luzes recebem uma ordem para acendimento. O *framework* adapta a aplicação, mostrada na Figura 36 (a) e (b). Para esta PoC foram definidas duas regras de adaptação: uma baseada na localização do usuário e uma baseada na proximidade da sala. A regra baseada em localização avalia a proximidade geográfica do local de trabalho do usuário, considerando os dados do GPS. A regra baseada na proximidade da sala avalia a distância do usuário e de sua sala, com base em um sinal de *beacon*.

Para a implementação dos objetos inteligentes, foi utilizado um Arduíno Mega,

[1] https://www.entrepreneur.com/article/307454

com interfaces de comunicação *Bluetooth* e WiFi, respectivamente. A Figura 36 apresenta o hardwares utilizados para a PoC, a saber: Figura 36(c) os componentes para controlar o ar condicionado. Na Figura 36(d) são exibidas as lâmpadas e na Figura 36(e) o *hub* utilizado para comunicação com a lâmpada.

Figura 36 – Aplicação e Hardware utilizados na prova de conceito

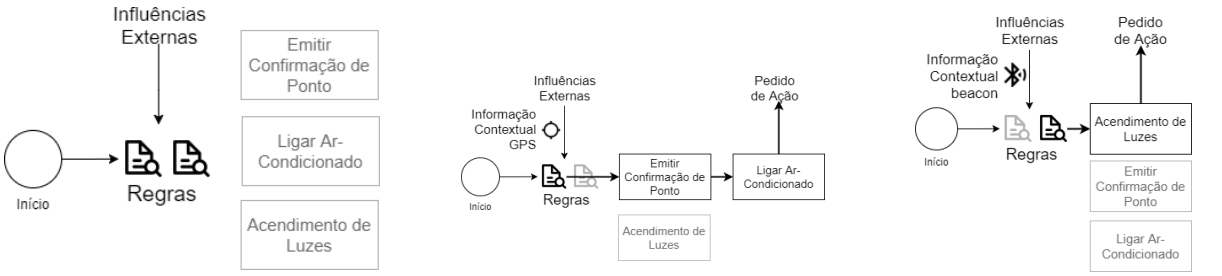


Fonte: Do Autor

Nesta PoC foram definidas três ações: a ação responsável por acender as luzes, a ação que confirma a presença do usuário e a ação responsável por ligar o ar-condicionado. Essas ações fazem a transição para o estado de execução apenas quando as regras percebem que é necessário iniciar uma adaptação. Desse modo, as ações permanecem inativas enquanto a regra responsável pela adaptação não seja satisfeita. Os estímulos das regras são as informações contextuais, sejam estas provindas do ambiente (físico e lógico) ou do próprio sistema. Como resposta das regras, alguma adaptação pode ocorrer. Na implementação dessa PoC, o *middleware* LoCCAM (MAIA *et al.*, 2013), apresentado na seção 2.4.2, foi utilizado como plataforma de suporte à interação com os *smart-objects* dispostos no ambiente físico.

O fluxo de execução do AV é modificado em tempo de execução (Figura 37). No início, o fluxo permanece inativo, pois as informações contextuais recebidas não atendem às regras definidas (a). Quando uma das regras é satisfeita (b), o fluxo de adaptação relacionado é acionado. Após a sua conclusão, o fluxo retorna ao estado inativo (a). Quando a regra de proximidade é satisfeita (c), o fluxo de adaptações referente torna-se ativo.

Figura 37 – Representação do *workflow* da PoC



(a) Sem fluxo definido

(b) execução de fluxo baseado na Regra de Localização

(c) execução de fluxo baseado na Regra de proximidade

Fonte: Do Autor

5.2 Acoplamento

Acoplamento é a interação entre classes, de modo que uma classe use métodos ou atributos definidos por outra classe (RODRIGUEZ; HARRISON, 2001). Isso, pois como descrito por Rodriguez e Harrison (2001), pode indicar a complexidade exigida no design e desenvolvimento de um sistema. Esta avaliação é considerada importante para este trabalho devido à demanda que a constante mudança tecnológica exige, no qual as modificações devem ser suportadas, com alterações simples ou sem alterações em módulos existentes.

Para verificar esta característica, foi utilizada a métrica conhecida como Fator de Acoplamento (do inglês, *Coupling Factor* (COF)). Esta métrica considera o conceito de relação cliente-servidor entre as classes constituintes de um software e indica o quanto um software é conectado (ABREU; CARAPUÇA, 1994; RODRIGUEZ; HARRISON, 2001).

O Fator de Acoplamento é dado pela razão entre o número total de conexões entre as classes e o maior número possível de conexões do software. A equação referente ao Fator de Acoplamento é demonstrada na Equação 5.1.

$$COF = \frac{\sum_{i=1}^{TC} \left[\sum_{j=1}^{TC} isClient(Ci, C\,j) \right]}{TC^2 - TC} \tag{5.1}$$

Onde:

$$isClient(C_c, C_s) = \begin{cases} 1 & \text{se } C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0 & \text{caso contrário} \end{cases} \tag{5.2}$$

A relação cliente-servidor, representada na Equação 5.2, significa que a classe cliente (C_c) contém pelo menos uma referência de não-herança para um recurso da classe servidora (C

$_s$). Deste modo, o Fator de Acoplamento representa o número real de acoplamentos não causados por herança de classes.

O SUCCEEd deve ser importado para o projeto da aplicação e algumas de suas classes devem ser herdadas ou implementadas pelos desenvolvedores. Como essa métrica é baseada no conceito de relacionamento cliente-servidor, essa avaliação foi dividida em duas situações. A primeira situação consiste em um projeto no qual a classe *Communication* do SUCCEEd se comunica com alguma outra classe de *ContextManager*. A segunda situação é aquela na qual a classe *Communication* do SUCCEEd é estendida e implementada como um *Listener* ou *ContextManager*.

Na primeira situação, o acoplamento varia dependendo do número de classes presentes no projeto, dado que afeta o número de possibilidades de comunicação. No entanto, como existem apenas cinco métodos de comunicação na classe de comunicação do SUCCEEd, a fórmula para calcular o COF pode ser expressa como:

$$COF = \frac{5}{TC^2 - TC} \tag{5.3}$$

Para esta situação foi utilizada a PoC implementada e o COF foi igual a 0,032. Ainda, há evidência para acreditar que este cálculo não se altere, mesmo com a ausência de uma classe de *ContextManager*, uma vez que o importante para o cálculo do COF são apenas os métodos de comunicação do SUCCEEd.

Na segunda situação, a classe de comunicação é estendida e implementada como um *Listener* ou *ContextManager*. Para calcular esta métrica nesta situação, foram consideradas as classes e possibilidades de comunicação dentro do próprio SUCCEEd. Além disso, nessa situação, a classe de comunicação consiste apenas em três métodos. Isso ocorre porque, nessa situação, a própria classe de comunicação é um servidor. O cálculo pode ser feito com a Equação 5.4.

$$COF = \frac{3}{110} \tag{5.4}$$

Nesta situação, o cálculo tem como base o código do próprio SUCCEEd, resultando em um COF igual a 0.027.

Um *software* totalmente conectado possui um COF igual a 1 (ABREU; CARAPUÇA, 1994; RODRIGUEZ; HARRISON, 2001). Em ambas as situações, o valor de COF é muito

menor que o valor de referência definido pela métrica. Isto dá indícios que o SUCCEEd possui uma estrutura flexível, um alto grau de independência e potencial de reuso.

5.3 Qualidade da Auto-Adaptação

Os sistemas auto-adaptativos exigem uma evolução contínua das capacidades de adaptação e reconfiguração em tempo de execução. Consequentemente, é importante monitorar e avaliar a qualidade das adaptações. Nesse contexto, Kaddoum *et al.* (2010) propõem um conjunto de métricas para avaliar sistemas auto-adaptativos. Essas métricas são agrupadas em quatro categorias: metodológicas, arquitetônicas, intrínsecas e tempo de execução.

Para a avaliação executada neste trabalho, a investigação partiu de duas questões: **"O SUCCEEd é capaz de auto-adaptar os sistemas de maneira eficiente?"** e **"O SUCCEEd foi implementado de modo a permitir que o sistema recupere seu comportamento após algum desequilíbrio ocorrido no próprio sistema?"** Para responder à essas questões, as métricas de Kaddoum *et al.* (2010) foram estudadas e duas destas métricas foram escolhidas: *Working Vs. Adaptivity Time* e *Time for Adaptation*.

A métrica *Working vs. Adaptivity Time* (WAT) indica o tempo gasto para prover a adaptação no ambiente ou no sistema em relação ao tempo gasto para executar funcionalidade associada. Deste modo, esta métrica visa expressar se o tempo para adaptar é maior do que o tempo necessário para executar determinada funcionalidade. O WAT é calculado pela Equação 5.5:

$$WAT = \frac{WorkingTime}{AdaptivityTime} \tag{5.5}$$

No WAT, o numerador *WorkingTime* representa o tempo decorrido de um determinado número de adaptações. Já o denominador *AdaptivityTime* representa o tempo decorrido apenas para prover o processo de adaptação.

A métrica *Time for Adaptation* (TA) indica o tempo requerido para retornar ao funcionamento habitual do sistema. O TA é dado pela Equação 5.6:

$$TA = Time\,for Adaptation \tag{5.6}$$

A fim de permitir uma maior similaridade com um ambiente heterogêneo real,

foi desenvolvida uma biblioteca Java[2] capaz de instanciar sensores (com base nas chaves do LoCCAM). Esta biblioteca tem um número de regras de adaptação implementadas que são selecionadas aleatoriamente e associadas quando as adaptações são geradas. Com isso, é possível, de maneira randômica, seleciona um determinado sensor, uma determinada regra e injetar adaptações para esta regra, permitindo medir a execução das adaptações. Aqui é importante informar que as adaptações modeladas eram simples, sem a necessidade de interação com a interface do usuário.

Essas métricas de qualidade foram analisadas em cenários com 1, 10, 100, 1.000, 5.000 e 10.000 adaptações. Cada cenário foi executado 10 vezes e a média de execução foi considerada como o resultado final, conforme a Tabela 3. Para demonstrar que as médias utilizadas são representativa para o experimento, o desvio padrão também foi calculado. Quando o desvio padrão está alto, é indício de que os dados são heterogêneos, ao passo que, quanto mais próximo de 0, mais homogêneos são os dados. Para WAT, o desvio padrão encontrado foi de 2,10, enquanto que para TA o desvio padrão encontrado foi de 179,97. Portanto, ambos os conjuntos de dados são heterogêneos e representativos para o experimento.

Tabela 3 – Resultado das Métricas

Métrica	***1 A***	***10 As***	***100 As***	***1.000 As***	***5.000 As***	***10.000 As***	***Desvio Padrão***
WAT	0	1,6	2,5	6	4	4	2.10
TA	0ms	0ms	7,6ms	65,33ms	248,33ms	438,33ms	179.97

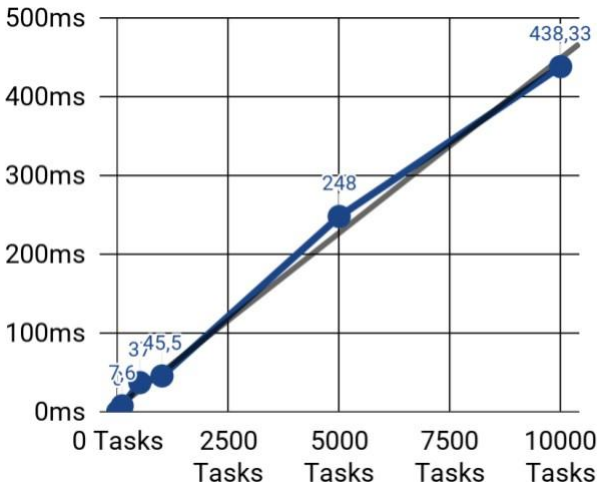
Se o WAT for menor que 1, muito tempo será gasto para adaptação. Os resultados mostram que somente no cenário com uma adaptação, o SUCCEEd passa mais tempo orquestrando a adaptação do que executando a funcionalidade associada. Estes resultados dão indícios de que o SUCCEEd é capaz de adaptar o sistema de modo eficiente.

Apesar de apresentar uma disparidade considerável na métrica TA, como mostrado na Tabela 3 e na Figura 38, os dados analisados mostram que o SUCCEEd tende a seguir uma complexidade linear quando executado em um cenário onde um grande número de ações ou adaptações são necessárias. A tendência linear sugere que a solução é eficiente se considerarmos que o SUCCEEd processa *n* adaptações dependentes do contexto em tempo de execução. Para esta avaliação, foi importante verificar apenas a capacidade de execução das adaptações e estes

[2] https://github.com/Belmondo/SUCCEEd/tree/master/GeraObjetos

resultados mostram que as propriedades do SUCCEEd são implementadas para permitir que o sistema recupere seu comportamento nominal após uma necessidade de alteração.

Figura 38 – Avaliação da métrica *Time for Adaptation* do SUCCEEd



Fonte: Do Autor

No que diz respeito às avaliações da qualidade da auto-adaptação, é importante indicar que foi executado um processo de *warming-up* da JVM do Java. A máquina virtual Java ou JMV é uma plataforma que carrega e executa as aplicações desenvolvidas na linguagem Java e quando é iniciado um novo processo na JVM, todas as classes necessárias são carregadas na memória por uma instância do *ClassLoader*, que trata-se de um objeto responsável pelo processo de carregamento de classes. Todas as classes usadas são colocadas no cache da JVM, tornando-as mais acessíveis e consequentemente diminuindo o tempo de execução.

Dito isto, é interessante armazenar em cache todas as classes, para que elas fiquem disponíveis mais rapidamente quando acessadas em tempo de execução. Este processo de é conhecido como *warming-up*. Neste ponto da avaliação do SUCCEEd, o processo de *warming-up* foi executando, uma vez que há razões para acreditar que tal ação permite focar a avaliação nos tempos do próprio framework e desconta o tempo proveniente do processo de *loading* da JVM. A saber, o processo de *warming-up* utilizado no SUCCEEd foi através de uma implementação. Consistiu de uma instanciação das classes do *framework* repetidas vezes assim que a aplicação para a avaliação era iniciada.

5.4 Ameaças à Validade

A metodologia proposta por Wohlin *et al.* (2012) é utilizada neste trabalho de dissertação para verificar a validade dos experimentos executados. Segundo essa metodologia, é possível categorizar ameaças relacionados a validade em quatro grupos: de conclusão. ameaças internas, ameaças externas, e de construto. As ameaças identificadas neste trabalho de dissertação e as estratégias empregadas para mitigá-las são apresentas a seguir.

As ameaças à validade de conclusão dizem respeito a questões que afetam a capacidade de tirar a conclusão correta sobre as relações entre a entidade e o resultado de um determinado experimento. Dentro desse grupo, uma ameaça que é possível verificar neste trabalho de dissertação é o Baixo Poder Estatístico. O poder de um teste estatístico diz respeito a capacidade do teste para revelar um padrão verdadeiro nos dados. Se o poder é baixo, há um alto risco de que uma conclusão errônea seja tirada.

Como explicitado na Seção 5.3, foi implementada uma biblioteca Java capaz de simular a existência de sensores e instanciar qualquer quantidade de adaptação. Esta biblioteca tem um número de regras de adaptação implementadas que são selecionadas aleatoriamente e associadas quando as adaptações são geradas.

A julgar pelo fato de que os experimentos foram guiados com seu uso, a maneira a qual essa biblioteca foi implementada pode trazer algum viés para os valores dos experimentos, através de uma seleção não randômica de *smart objects*, adaptações e estados. E isso iria de encontro à característica da heterogeneidade.

A fim de mitigar essa ameaça de baixo poder estatístico, a biblioteca Java utilizada foi submetida a testes estatísticos conhecidos como testes de ensaio ou testes de *runs*. Antes, porém, é necessário detalhar como funciona a atribuição aleatória de um sensor para a biblioteca Java desenvolvida. O código a seguir mostra mais detalhes dessa atribuição.

```
1
2 for( int i=0; i < numero Sensores ; i++) {
3
4     Sensor sensorAleatorio = new Sensor();
5     sensorAleatorio.chave = listaChaves.get(numeroAleatorio(0,
          listaChaves.size()));
6     listaSensores.add(sensorAleatorio);
7
```

```
8 }
```

Nesse código, a variável “numeroSensores” representa o número desejado de sensores. Na linha de código “ listaChaves.get(numeroAleatorio(0, listaChaves.size()))” ocorre a seleção de uma chave que representa um sensor. Esse processo consiste de atribuir uma chave, que está em uma posição x, sendo esta posição x definida por um número aleatório e, sendo esse número aleatório definido com base no tamanho máximo de chaves salvas. Por exemplo, caso o método “numeroAleatorio(0, listaChaves.size())” retorne o número 1, então é retornada para o “sensorAleatorio.chave”, a chave na posição 1 da lista.

Tendo como princípio esse trecho de código, foi atribuído para cada chave um número do conjunto *N**. Ao fazer a atribuição de um determinado sensor, o número relativo a cada atribuição aleatória era impresso através de *logs*. De posse desses números, esse valores foram inseridos em um software estatístico e de análise de dados, chamado *Minitab Statistical* [3]. Como supracitado, uma forma de testar se os dados são aleatórios é usar um teste de ensaios ou testes de *runs* para procurar por um padrão nos dados. A análise dos dados é mostrada na Figura 39.

Figura 39 – Análise de Aleatoriedade de Dados da Biblioteca Utilizada para a Simulação

Teste de Ensaios: C1

Estatísticas Descritivas

		Número de observações	
N	K	≤ K	> K
1000	5,983	450	550

K = média amostral

(a) Estatísticas Descritivas

Teste

Hipótese nula	H_0: a ordem dos dados é aleatória
Hipótese alternativa	H_1: a ordem dos dados não é aleatória

Número de ensaios

Observado	Esperado	Valor-p
501	496,00	0,749

(b) Resultado do Teste

Fonte: Do Autor

Para determinar se a ordem dos dados é aleatória, é necessário comparar o valor-p ao nível de significância, sendo este 0,05. Como demonstrado na Figura 39 (b) o valor-p é 0,749 para essa sequência de dados analisados. Considerando que o valor de p > nível de significância, então não é possível rejeitar a hipótese H_0, pois não há provas suficientes para concluir que a ordem dos dados não é aleatória.

[3] http://www.minitab.com/pt-br/

5.5 Conclusão

Este capítulo apresentou a avaliação desenvolvida para o SUCCEEd. Para isso, uma aplicação Android foi utilizada como prova de conceito para ilustrar o processo de auto-adaptação em tempo de execução. A PoC mostrou a utilização do SUCCEEd em um cenário real, com dispositivos implementados utilizando uma plataforma Arduíno. As informações relevantes para o usuário eram a localização geográfica e a localização *indoor*, utilizando GPS e sinal de *beacon*, respectivamente, como sensores. Como atuadores, foram utilizados o ar condicionado e luzes. A informação contextual e o envio de ações para o ambiente foi por intermédio da plataforma de *middleware* LoCCAM. Essa PoC apresentou indícios que a solução é vantajosa e pode ser usada em um cenário real da IoT.

Outra avaliação, foi o Fator de Acoplamento ou COF que visa entender quão acoplado um sistema é de outro utilizado o conceito de cliente-servidor. Nesta avaliação, os resultados apresentam indícios que o SUCCEEd é desacoplado, e portanto, uma boa possibilidade de reuso.

Além destas, outras duas avaliações de qualidade da auto-adaptação foram descritas. A primeira visava entender se o mecanismo demanda menos tempo para adaptar do que executando a funcionalidade associada. A segunda visava entender o tempo necessário pelo *framework* para retornar a aplicação para um comportamento nominal após uma perturbação. Nesta análise, os resultados indicam que o tempo utilizado pelo SUCCEEd para adaptar o sistema é menor que o tempo utilizado para executar as suas funcionalidades que eram objetivo da adaptação; e que o número de etapas necessárias para adaptação dos sistemas é mínimo.

Neste capítulo, foi apresentado também a ameaça à validade relacionada ao baixo poder estatístico. Para o desenvolvimento das avaliações, foi utilizada uma biblioteca a fim de simular sensores, permitindo representar o SUCCEEd perante um cenário heterogêneo. Visando validar a aleatoriedade empregada por essa biblioteca e assim mitigar a ameaça à validade, foi empregado testes de ensaio ou teste de *runs*. Os resultados da análise dos dados indica que não é possível rejeitar a hipótese da aleatoriedade, por falta de provas suficientes para concluir o contrário.

6 CONCLUSÃO

Este capítulo sumariza o que foi apresentado neste trabalho de dissertação e os resultados alcançados por este trabalho, bem como os trabalhos futuros. A Seção 6.1 apresenta os principais resultados atingidos por este trabalho. A Seção 6.2 descreve as limitações do SUCCEEd e a Seção 6.3 elicita as expectativas de trabalhos futuros.

6.1 Resultados Alcançados

Este trabalho apresentou o SUCCEEd, um *framework* de suporte ao desenvolvimento de aplicações auto-adaptativas para o ambiente de IoT. O SUCCEEd fornece um conjunto de estruturas para definição e especificação de adaptações, estratégias e regras de adaptação. Todos esses elementos foram desenvolvidos de modo que o *framework* pudesse ser genérico o suficiente para permitir sua extensão de maneira flexível. Ainda, ele integra esses componentes a fim de garantir a adaptação em tempo de execução. Além disso, por ser baseado em Java, o SUCCEEd mostra-se como uma alternativa para qualquer sistema implementado através desta linguagem, desde aplicações desktop, aplicações móveis e web.

Para avaliar o SUCCEEd foi desenvolvida uma aplicação com o propósito de servir como prova de conceito. Como cenário para essa prova de conceito, foi imaginado um ambiente onde a aplicação automatizaria algumas tarefas do usuário, como confirmar presença e ligar a lâmpada e o ar-condicionado. Para isso, a aplicação utilizou os estados do sistema, que eram modificados por informações contextuais, a fim de entender o ambiente e determinar quais adaptações deveriam ser instanciadas. Essa aplicação apresenta indícios de que o SUCCEEd pode ser utilizado em um cenário real, sendo portanto, viável a sua implementação no processo de desenvolvimento de um SAS.

Para mostrar a eficiência do SUCCEEd em prover a adaptação em tempo de execução, bem como demonstrar a sua capacidade de recuperar o sistema, foi realizada uma avaliação por meio de simulação. A simulação permitiu investigar o comportamento do SUCCEEd em um ambiente com alto número de dispositivos e adaptações. Os resultados desse processo indicam que o tempo utilizado pelo SUCCEEd é menor do que o tempo necessário para executar a funcionalidade da adaptação. Além disso, os resultados demonstraram que o número de passos necessários para adaptar o sistema é mínimo. Essas implicações demonstraram a capacidade de resposta do SUCCEEd em tempo de execução.

Quando comparado com os demais trabalhos relacionados a mecanismos com a capacidade de auxiliar no processo de auto-adaptação, o SUCCEEd consegue atender aos critérios definidos por Serugendo *et al.* (2007). A Tabela 4 mostra a comparação entre o SUCCEEd e os demais trabalhos.

Tabela 4 – Comparativo entre os Trabalhos Relacionados e o SUCCEEd

Trabalhos Relacionados	Desacoplado	Endo-Adaptação	Exo-Adaptação	Adaptação da Execução	Descrição
(MONTAGUT and MOLVA, 2005)	✓		✓		✓
(GINER et al., 2010)	✓	✓			✓
(KRUPITZER et al., 2013)	✓		✓	✓	
(ABEYWICKRAMA et al., 2014)	✓		✓		✓
(SEIGER et al., 2015)		✓	✓		✓
(WIELAND et al., 2015)		✓	✓		✓
(TSUDA et al., 2016)	✓	✓		✓	
IFTTT		✓		✓	✓
Node-Red		✓	✓		✓
(LI et al., 2017)	✓	✓	✓		
(GEROSTATHOPOULOS et al., 2018)	✓	✓		✓	
SUCCEEd	✓	✓	✓	✓	✓

Fonte: Do autor

Ainda como resultados do mestrado, durante este período, quatro artigos foram publicados em eventos nacionais e internacionais, incluindo publicações em *workshops* e conferências/simpósios. A Tabela 5 apresenta detalhes destes artigos.

O primeiro artigo (n. 1) apresenta a estrutura inicial do SUCCEEd, com avaliações relacionadas a sua performance e foi apresentado no Simpósio Brasileiro de Computação Ubíqua e Pervasiva. O segundo artigo (n. 2) descreve a estrutura completa do SUCCEEd, seu modo de funcionamento e avaliações direcionadas a qualidade da auto-adaptação, bem como de acoplamento do *framework*, tal qual apresentado nos capítulos 4 e 5, respectivamente.

Ainda na Tabela 5 são apresentados mais dois trabalhos produzidos em co-autoria durante o período do mestrado. O terceiro artigo (n. 3) tinha como objetivo compreender a forma a qual os os usuários utilizam aplicativos de mobilidade urbana e avaliar a qualidade da interação e da interface desses aplicativos. O quarto artigo (n. 4) apresenta o GreatRoom, uma aplicação Android que utiliza sinais de *beacons* distribuídos em um ambiente, a fim de permitir a detecção

Tabela 5 – Lista dos artigos científicos produzidos

Número	Referência	Tipo	Situação
1	**Aragão Junior, B. R.**; Nogueira, T. P. ; Maia, M. E. F; Andrade, R. M. C. *Framework* de Suporte ao Desenvolvimento e Evolução de Aplicações Auto-Adaptativas em IoT. In: X Simpósio Brasileiro de Computação Ubíqua e Pervasiva. XXXVIII Congresso da Sociedade Brasileira de Computação, Natal, RN, 2018.	Simpósio	Publicado
2	**Aragão Junior, B. R.**; Andrade, R. M. C.; Maia, M. E. F; Nogueira, T. P. *SUCCEEd: Support Mechanism for Creating and Executing Workflows for Decoupled SAS in IoT. In: XII IEEE International Workshop on Quality Oriented Reuse of Software . XXXXII IEEE International Conference on Computers, Software & Applications*, Tokyo, Japão, 2018	*Workshop*	Publicado
3	Almeida, R. L., Mesquita, L. B. M., Carvalho, R. M., **Junior, B. R.**, & Andrade, R. M. (2016). Quando a tecnologia apoia a mobilidade urbana: Uma avaliação sobre a experiência do usuário com aplicações móveis. In Proceedings of the XV Brazilian Symposium on Human Factors in Computer Systems (IHC 2016). Sociedade brasileira de Computação-SBC, Porto Alegre, Brazil (2016, in Portuguese) Google Scholar.	Conferência	Publicado
4	Darin, T., Barbosa, J., **Rodrigues, B.**, & Andrade, R. M. (2016). GreatRoom: Uma aplicação android baseada em proximidade para a criação de salas virtuais inteligentes. In XV Workshop de Ferramentas e Aplicações (WFA). XXII Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2016), Teresina, Piauí.	*Workshop*	Publicado

Fonte: Do autor

de usuários nas salas ou cômodos. Por ser uma aplicação voltada para o ambiente de sala de aula e palestras, permite ainda o compartilhamento de documentos para os presentes em determinada sala.

6.2 Limitações

Uma das principais limitações do SUCCEEd é o fato de sua implementação atual ter sido direcionada apenas para o sistema operacional Android. Esse fato pode trazer restrições relacionadas à comunicação entre os dispositivos. Outra limitação nesse sentido é que a linguagem utilizada para a implementação das adaptações no SUCCEEd é o Java, o que acarreta a necessidade de instanciação descritiva e manual por parte do programador.

Além disso, não há nenhum mecanismo no SUCCEEd que verifique a permissão do usuário, do sistema ou do dispositivo com relação à execução de determinada tarefa. Na versão atual do SUCCEEd, essa responsabilidade é demandada para possíveis camadas externas ao *framework* ou é deixada à cargo de definição explícita do programador.

No que diz respeito ao processo de execução, para esta versão do SUCCEEd não

foi implementado nenhum mecanismo que verifique a existência de estados circulares dentro da própria execução. Aliado a isto, não foi desenvolvido um meio de verificar adaptações conflitantes para uma mesma regra e cenários como esse necessitam ser levados em consideração em uma futura melhoria do *framework*.

6.3 Trabalhos Futuros

Este trabalho de mestrado deixa lacunas em aberto para a evolução do SUCCEEd que são descritos a seguir:

- O estado atual do *framework* considera um sistema apenas como centralizador das adaptações. Desse modo, é necessária uma melhor implementação a fim de garantir suporte distribuído.
- Evoluir a interface de descrição de regras, para permitir uma melhor alocação do SUCCEEd dentro do MAPE-K Loop. Isso permite uma melhor implementação de planos de execução por outros mecanismos ou *frameworks*.
- Evoluir a forma de interação com o programador, de modo a permitir a criação de adaptações e regras de maneira mais visual, através de modelagem.
- Em seu estado atual, o *SUCCEEd* não possui um mecanismo de verificação de estados circulares e de adaptações conflitantes. Por isso, necessidade de implementação de um mecanismo para esse fim que permita validar essas questões antes do início da adaptação ou mesmo em tempo de execução do sistema podem ser alvos de trabalhos futuros.
- Realizar mais avaliações relacionadas à qualidade da auto-adaptação e do consumo de recursos associados à sua execução (e.g. bateria). Além disso, realizar uma avaliação com desenvolvedores a fim de analisar o esforço demandando para a utilização do SUCCEEd.
- Analisar o comportamento do SUCCEEd com numerosas adaptações a fim de verificar o seu comportamento perante um estado de estresse.
- Durante o trabalho, não foi conduzida nenhum tipo de avaliação empírica com os demais trabalhos relacionados. É necessário que avaliações neste sentido sejam conduzidas, tanto para trazer mais validade ao *framework* desenvolvido, quanto para permitir encontrar possíveis problemas e oportunidades de melhoria.
- É importante utilizar o SUCCEEd com um sistema real e refazer o processo de avaliação do *framework*, a fim de verificar a sua real utilização em um ambiente real ou em um sistema com regras e adaptações mais complexas. Essas características podem causar

impactos negativos nos tempos alcançados nas avaliações, bem como permitir encontrar erros e pontos de melhoria do *framework*.

- Investigar as causas e levantar hipóteses que expliquem a disparidade de valores encontrados na avaliação da qualidade da auto-adaptação executada na Seção 5.3.
- Implementar as demais estruturas de *gateways*, a fim de permitir uma maior possibilidade de fluxos de execuções. A saber, as demais estruturas que podem ser implementadas são: *gateway* complexo, *gateway* de eventos, *gateway* inicial exclusivo e *gateway* inicial paralelo.

REFERÊNCIAS

ABEYWICKRAMA, D. B.; HOCH, N.; ZAMBONELLI, F. An integrated eclipse plug-in for engineering and implementing self-adaptive systems. In: **2014 IEEE 23rd International WETICE Conference**. [*S. l.*]: IEEE, 2014. p. 3–8.

ABREU, F. B.; CARAPUÇA, R. Object-oriented software engineering: Measuring and controlling the development process. [*S. l.*], v. 186, 1994.

AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys Tutorials**, v. 17, n. 4, p. 2347–2376, 6 2015.

ALEGRE, U.; AUGUSTO, J. C.; CLARK, T. Engineering context-aware systems and applications: A survey. **Journal of Systems and Software**, v. 117, p. 55–83, 7 2016.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787–2805, 10 2010.

BARESI, L.; GHEZZI, C. The disappearing boundary between development-time and run-time. In: **Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research**. New York, NY, USA: ACM, 2010. (FoSER '10), p. 17–22. ISBN 978-1-4503-0427-6.

BARNAGHI, P.; WANG, W.; HENSON, C.; TAYLOR, K. Semantics for the internet of things: early progress and back to the future. **International Journal on Semantic Web and Information Systems**, v. 8, n. 1, p. 1–21, 2012.

BARRETO, F. M. **COAP-CTX: EXTENSÃO SENSÍVEL AO CONTEXTO PARA DESCOBERTA DE OBJETOS INTELIGENTES EM INTERNET DAS COISAS**. 2017.

BASSI, A.; BAUER, M.; FIEDLER, M.; KRAMP, T.; KRANENBURG, R. van; LANGE, S.; MEISSNER, S. **Enabling Things to Talk: Designing IoT Solutions with the IoT Architectural Reference Model**. 1. ed. The address: Springer Publishing Company, Incorporated, 2016. ISBN 3662524945, 9783662524947.

BECKER, C.; SCHIELE, G.; GUBBELS, H.; ROTHERMEL, K. Base - a micro-broker-based middleware for pervasive computing. In: **Proceedings of the First IEEE International Conference on Pervasive Computing and Communications**. (PerCom 2003): IEEE, 2003. p. 443–451.

BRUN, Y.; SERUGENDO, G. M.; GACEK, C.; GIESE, H.; KIENLE, H.; LITOIU, M.; MüLLER, H.; PEZZè, M.; SHAW, M. Engineering self-adaptive systems through feedback loops. In: CHENG, B. H.; LEMOS, R.; GIESE, H.; INVERARDI, P.; MAGEE, J. (Ed.). **Software Engineering for Self-Adaptive Systems**. Berlin, Heidelberg: Springer-Verlag, 2009. v. 5525, p. 48–70.

CAVALCANTE, E.; ALVES, M. P.; BATISTA, T.; DELICATO, F. C.; PIRES, P. F. An analysis of reference architectures for the internet of things. In: **2015 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA)**. [*S. l.*]: IEEE, 2015. p. 1–4.

ENDLER, M.; BAPTISTA, G.; SILVA, L.; VASCONCELOS, R.; MALCHER, M.; PANTOJA, V.; PINHEIRO, V.; VITERBO, J. Contextnet: Context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In: **Proceedings of the Workshop on Posters and Demos Track**. [*S. l.*]: ACM, 2011. p. 2.

ENDLER, M.; SILVA, F. S. e. Past, present and future of the contextnet iomt middleware. **Open Journal of Internet Of Things**, v. 4, n. 1, p. 7–23, 2018.

EVANS. **How the next evolution of the internet is changing everything**. 2011. Disponível em: http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. Acesso em: 13 Jan. 2018.

FREMANTLE, P. A reference architecture for the internet of things. 10 2015.

GEORGAKOPOULOS, D.; HORNICK, M.; SHETH, A. An overview of workflow management: From process modeling to workflow automation infrastructure. **Distributed and Parallel Databases**, v. 3, n. 2, p. 119–153, 4 1995.

GEROSTATHOPOULOS, I.; PREHOFER, C.; BURES, T. Adapting a system with noisy outputs with statistical guarantees. In: **Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems**. New York, NY, USA: ACM, 2018. (SEAMS '18), p. 58–68. ISBN 978-1-4503-5715-9.

GIGLI, M.; KOO, S. G. Internet of things: Services and applications categorization. **Adv. Internet of Things**, v. 1, n. 2, p. 27–31, 7 2011.

GINER, P.; CETINA, C.; FONS, J.; PELECHANO, V. Developing mobile business processes for the internet of things. **IEEE Pervasive Computing**, v. 9, n. 2, p. 18–26, 4 2010.

GRIGORAS, D. Challenges to the design of mobile middleware systems. In: **Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on**. [*S. l.*]: IEEE, 2006. p. 14–19.

HOLLINGSWORTH, D.; HAMPSHIRE, U. Workflow management coalition: The workflow reference model. **Document Number TC00-1003**, Document Status, v. 19, 1995.

HUBER, S.; SEIGER, R.; KüHNERT, A.; SCHLEGEL, T. A context-adaptive workflow engine for humans, things and services. In: **Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct**. New York, NY, USA: ACM, 2016. (UbiComp '16), p. 285–288. ISBN 978-1-4503-4462-3.

HUGHES, D. **Self Adaptive Software Systems are Essential for the Internet of Things**. 2018. Keynote at the 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Gothenburg, Sweden.

IBM. **An architectural blueprint for autonomic computing**. 2005. Disponível em: https://www-03.ibm.com/autonomic/pdfs/AC\%20Blueprint\%20White\%20Paper\%20V7.pdf. Acesso em: 24 jan. 2018.

KADDOUM, E.; RAIBULET, C.; GEORGÉ, J.-P.; PICARD, G.; GLEIZES, M.-P. Criteria for the evaluation of self-* systems. In: **Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems**. [*S. l.*]: ACM, 2010. p. 29–38.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. **Computer**, v. 36, n. 1, p. 41–50, 1 2003.

KITCHENHAM, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, p. 2105–2125, 10 2004.

KRAMER, J.; MAGEE, J. Self-managed systems: an architectural challenge. In: **Future of Software Engineering, 2007. FOSE '07**. [*S. l.*]: IEEE, 2007. p. 259–268.

KRUPITZER, C.; VANSYCKEL, S.; BECKER, C. Fesas: Towards a framework for engineering self-adaptive systems. In: **2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems**. [*S. l.*]: IEEE, 2013.

LI, Y.; LI, Q.; WANG, L.; CHENG, W.; WU, T. Adapt: An agent-based development toolkit and operation platform for self-adaptive systems. In: **2017 IEEE Conference on Open Systems (ICOS)**. [*S. l.*]: IEEE, 2017. p. 53–58. ISSN 2473-3660.

LIMA, F. F. d. P. **SysSU - Um Sistema de Suporte para Computaçãoo Ubíqua**. 2011.

MAIA, M. E. F.; FONTELES, A.; NETO, B.; GADELHA, R.; VIANA, W.; ANDRADE, R. M. C. Loccam - loosely coupled context acquisition middleware. In: **Proceedings of the 28th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2013. (SAC '13), p. 534–541. ISBN 978-1-4503-1656-9.

MEHO, L. I.; YANG, K. Impact of data sources on citation counts and rankings of lis faculty: Web of science versus scopus and google scholar. **Journal of the American Society for Information Science and Technology**, v. 58, n. 13, p. 2105–2125, 10 2007.

MONTAGUT, F.; MOLVA, R. Enabling pervasive execution of workflows. In: **2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing**. [*S. l.*]: IEEE, 2005. p. 10 pp.–.

MORIN, B.; HARRAND, N.; FLEUREY, F. Model-based software engineering to tame the iot jungle. **IEEE Software**, v. 34, n. 1, p. 30–36, 1 2017.

NAKAGAWA, E. Y.; OQUENDO, F.; MALDONADO, J. C. Reference architectures. **Software Architecture 1**, p. 55–82, 4 2014.

NEELY, S.; DOBSON, S.; NIXON, P. Adaptive middleware for autonomic systems. In: **Annales des télécommunications**. [*S. l.*]: Springer, 2006. v. 61, n. 9-10, p. 1099–1118.

NETO, B. J. d. A. **SYSSU-DTS: UM SISTEMA DE SUPORTE À COMPUTAÇÃO UBÍQUA BASEADO EM ESPAÇO DE TUPLAS DISTRIBUÍDO**. 2013.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context aware computing for the internet of things: A survey. **IEEE communications surveys & tutorials**, v. 16, n. 1, p. 414–454, 5 2013.

PRYSS, R.; TIEDEKEN, J.; REICHERT, M. Managing processes on mobile devices: the marple approach. 2010.

RAZZAQUE, M. A.; MILOJEVIC-JEVRIC, M.; PALADE, A.; CLARKE, S. Middleware for internet of things: A survey. **IEEE Internet of Things Journal**, v. 3, n. 1, p. 70–95, 2 2016.

RODRIGUEZ, D.; HARRISON, R. **An Overview of Object-Oriented Design Metrics**. 2001.

SEIGER, R.; HUBER, S.; SCHLEGEL, T. Proteus: An integrated system for process execution in cyber-physical systems. In: **International Conference on Enterprise, Business-Process and Information Systems Modeling**. [*S. l.*]: Springer, 2015. p. 265–280.

SERUGENDO, G. D. M.; FITZGERALD, J.; ROMANOVSKY, A.; GUELFI, N. A generic framework for the engineering of self-adaptive and self-organising systems. **Organic Computing-Controlled Self-organization**, n. 08141, p. 165–189, 5 2007.

SOMMERVILLE, I. **Software Engineering**. 9. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0137035152, 9780137035151.

SUNDMAEKER, H.; GUILLEMIN, P.; FRIESS, P.; WOELFFLÉ, S. Vision and challenges for realising the internet of things. **Cluster of European Research Projects on the Internet of Things, European Commision**, v. 3, n. 3, p. 34–36, 3 2010.

TAIVALSAARI, A.; MIKKONEN, T. A roadmap to the programmable world: Software challenges in the iot era. **IEEE Software**, v. 34, n. 1, p. 72–80, 1 2017.

TSUDA, H.; NAKAGAWA, H.; TSUCHIYA, T. A dynamic verification mechanism for real-time self-adaptive systems. In: **2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)**. [*S. l.*]: IEEE, 2016. p. 265–266.

UDOH, I. S.; KOTONYA, G. Developing iot applications: challenges and frameworks. **IET Cyber-Physical Systems: Theory & Applications**, v. 3, n. 6, p. 65 – 72, 7 2018.

VÖLTER, M. **Software architecture patterns–a pattern language for building sustainable software architectures**. [*S. l.*]: Mar, 2005.

WEYNS, D.; SCHMERL, B.; GRASSI, V.; MALEK, S.; MIRANDOLA, R.; PREHOFER, C.; WUTTKE, J.; ANDERSSON, J.; GIESE, H.; GÖSCHKA, K. M. On patterns for decentralized control in self-adaptive systems. In: **Software Engineering for Self-Adaptive Systems II**. Berlin, Heidelberg: Springer, 2013. p. 76–107.

WIELAND, M.; SCHWARZ, H.; BREITENBüCHER, U.; LEYMANN, F. Towards situation-aware adaptive workflows: Sitopt — a general purpose situation-aware workflow management system. In: **2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)**. [*S. l.*]: IEEE, 2015. p. 32–37.

WOHLIN, C.; RUNESON, P.; HST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLN, A. **Experimentation in Software Engineering**. [*S. l.*]: Springer Publishing Company, Incorporated, 2012. ISBN 3642290434, 9783642290435.

XIAOJIANG, X.; JIANLI, W.; MINGDONG, L. Services and key technologies of the internet of things. **ZTE Commun., Shenzhen, China**, v. 2, p. 011, 2010.

ZHAO, Z.; GAO, C.; DUAN, F. A survey on autonomic computing research. In: **2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA)**. [*S. l.*]: IEEE, 2009. v. 2, p. 288–291.