UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RODOLFO FELIPE MEDEIROS ALVES

MORE4IOT - UM *MIDDLEWARE* MULTIPROTOCOLO PARA INTERNET DAS COISAS

Mossoró/RN

2021

RODOLFO FELIPE MEDEIROS ALVES

MORE4IOT - UM *MIDDLEWARE* MULTIPROTOCOLO PARA INTERNET DAS COISAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação - associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

Linha de Pesquisa: Projeto de Sistemas e Circuitos
Orientador: Dr. Paulo Gabriel Gadelha Queiroz
Coorientador: Dr. Sílvio Roberto Fernandes de Araújo

Mossoró/RN

2021

F474m Felipe Medeiros Alves, Rodolfo.
MORE4IOT - UM MIDDLEWARE MULTIPROTOCOLO PARA INTERNET DAS COISAS / Rodolfo Felipe Medeiros Alves. - 2021.
113 f. : il.

Orientador: Paulo Gabriel Gadelha Queiroz.
Coorientador: Sílvio Roberto Fernandes de Araújo.
Dissertação (Mestrado) - Universidade Federal Rural do Semi-árido, Programa de Pós-graduação em Ciência da Computação, 2021.

1. Middleware. 2. Internet das Coisas. 3. Microsserviços. 4. Multiprotocolo. 5. Agricultura de Precisão. I. Gabriel Gadelha Queiroz, Paulo, orient. II. Roberto Fernandes de Araújo, Sílvio, co-orient. III. Título.

RODOLFO FELIPE MEDEIROS ALVES

MORE4IOT - UM *MIDDLEWARE* MULTIPROTOCOLO PARA INTERNET DAS COISAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPgCC) para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: 18 / 08 / 2021.

BANCA EXAMINADORA

Dr. Paulo Gabriel Gadelha Queiroz
Presidente da Banca

Dr. Sílvio Roberto Fernandes de Araújo
Examinador Interno

Dr. Lenardo Chaves e Silva
Examinador Interno

Dr. Rosana Teresinha Vaccare Braga
Examinadora Externa - USP

Agradecimentos

A Deus, por todas as bênçãos na minha vida.

A minha família, especialmente Cinara Medeiros, minha mãe, minha guerreira, por todo amor, incentivo e suporte ao longo de toda minha vida. Agradeço a Kathy Duarte, por todo amor e carinho nesse período, pela ajuda em todos os momentos difíceis.

Aos meus orientadores: Dr. Gabriel Gadelha e Dr. Sílvio Fernandes, pela ajuda, orientações, conselhos e por toda disponibilidade. Com certeza foi muito enriquecedor para minha formação, não somente durante o mestrado, mas também durante a minha graduação.

Aos professores do mestrado, pela contribuição na minha formação.

Aos colegas do mestrado, pelos momentos nas disciplinas e discussões.

As demais pessoas que fizeram parte, contribuindo de forma direta ou indiretamente para esse momento.

"Faça o melhor, na condição disponível, enquanto não tem condições melhores para fazer melhor ainda!"

Mário Sérgio Cortella

Resumo

O termo Indústria 4.0 é utilizado para sintetizar as mudanças promovidas nos processos industriais por meio da tecnologia. No Brasil, há grandes desafios para a economia conseguir crescer no *ranking* mundial de inovação. Apesar disso, os dados apontam que a quarta revolução industrial é uma oportunidade para o país, principalmente na agropecuária, que no ano de 2021 deve atingir um recorde no Valor Bruto da Produção Agropecuária de 1,076 trilhão de reais. A *Internet of Things (IoT)* é um dos pilares da Indústria 4.0 e pode contribuir, por exemplo, na agricultura de precisão para melhorar o monitoramento e controle dos processos que envolvem a produção agrária. O objetivo da IoT é agrupar a infraestrutura de rede global com recursos escalonáveis baseados em protocolos de comunicação e objetos ou dispositivos físicos que possuem identidade e atributos, a fim de conseguir uma sinergia e oferecer serviços inovadores. No entanto, inserir a IoT em qualquer ambiente é muito custoso e necessita de muitas adaptações. Portanto, este trabalho tem como objetivo permitir a construção de aplicações para IoT, que usam e conectam diversos dispositivos com protocolos de comunicação diferentes, sem a necessidade de adaptações significativas. Para isso, foi definido e desenvolvido o MORE4IoT – um *middleware* multiprotocolo baseado em uma arquitetura de microsserviços para auxiliar na construção de aplicações para IoT, que possui dois componentes principais: uma infraestrutura baseada em microsserviços para auxiliar na comunicação, no gerenciamento dos recursos, dos dados e das ações de controle; e, um componente que deve ser incluído no dispositivo para auxiliar nas configurações de comunicação e nas regras de controle. Além disso, o *middleware* foi validado por meio da construção de um protótipo aplicado à agricultura de precisão utilizando os mecanismos de comunicação e gerenciamento providos pelo MORE4IoT. Como resultado, o *middleware* simplificou a construção da comunicação por protocolos diferentes, das regras de controle e auxiliou no gerenciamento dos dispositivos da aplicação.

Palavras-chave: *Middleware*, Internet das Coisas, Microsserviços, Multiprotocolo, Agricultura de Precisão.

Abstract

The term Industry 4.0 is used to synthesize the changes promoted in industrial processes through technology. In Brazil, there are great challenges for the economy to grow in the world ranking of innovation. Despite this, the data show that the fourth industrial revolution is an opportunity for the country, mainly in agriculture, which in 2021 should reach a record in the gross value of agricultural production of 1.076 trillion reais. The Internet of Things (IoT) is one of the pillars of Industry 4.0 and can contribute, for example, to precision agriculture to improve the monitoring and control of processes involving agricultural production. The objective of IoT is to bundle the global network infrastructure with scalable resources based on communication protocols and physical objects or devices that have an identity and attributes to achieve synergy and offer innovative services. However, inserting the IoT in any environment is very costly and requires a great amount of adaptation. Therefore, this work aims to allow the construction of Internet of Things applications, which use and connect different devices with different communication protocols without significant adaptations. For this, MORE4IoT which is a multiprotocol middleware based on a microservices architecture was defined and developed to assist in the construction of IoT applications, which has two main components: an infrastructure based on microservices to assist in communication, resource management, data and control actions; and, a component that must be included in the device to assist in communication and control rules. In addition, the middleware was validated through the construction of a prototype applied to precision agriculture using the communication and management mechanisms provided by MORE4IoT. As a result, middleware simplified the construction of communication by protocols, control rules and solved the device management of the application.

Keywords: Middleware, Internet of Things, Microservices, Multiprotocol, Precision Agriculture.

Lista de ilustrações

Lista de tabelas

Lista de Abreviaturas e Siglas

ABDI Agência Brasileira de Desenvolvimento Industrial. 15

ABINC Associação Brasileira de Internet das Coisas. 15

AMQP *Advanced Message Queuing Protocol*. 16, 47, 52, 53, 61, 64, 65, 67, 75

AP Agricultura de Precisão. 15, 16, 83, 102

API *Application Programming Interface*. 43, 45, 47, 57, 58, 61–64, 67, 69, 70, 73

CE Critério de Exclusão. 34

CI Critério de Inclusão. 33

CoAP *Constrained Application Protocol*. 16, 43–46, 52, 61, 67, 75

CoT *Cloud of Things*. 46, 54

CQ Critério de Qualidade. 35, 39

DDS *Distribution Data Service*. 44, 46, 47, 52, 53

GCM *Google Cloud Messaging*. 43, 52

HTTP *Hypertext Transfer Protocol*. 16, 43–47, 52, 61, 67, 70, 73

IA Inteligência Artificial. 47

IoT *Internet of Things*. 5, 9, 12, 16, 17, 20, 21, 24, 28–31, 40, 41, 43–56, 60, 62–64, 67, 69, 70, 72, 78, 80, 81, 83, 94, 100–102

IP *Internet Protocol*. 80

JSON *JavaScript Object Notation*. 7, 64, 70–73, 75–78, 80, 94

M2M *Machine-to-Machine*. 52

MQTT *Message Queuing Telemetry Transport*. 16, 43–46, 52, 61, 64, 65, 67, 75

NoSQL *Not Only SQL*. 58

P2P *Peer-to-Peer*. 47, 53

PaaS *Platform as a Service*. 48

QC Questão de Conhecimento. 19, 20, 101

QDP Questão de *Design*/Projeto. 19

QE Questão de Efeito. 20, 101

QP Questão Principal. 12, 31, 40

QPS Questão de Sensitividade. 20, 101

QS Questão Secundária. 13, 31, 48, 49, 52

QSR Questão de Satisfação de Requisitos. 20, 101

QT Questão de *Trade-Off*. 20, 102

REST *Representational State Transfer*. 16, 43–47, 52, 61, 67, 70, 73

RSL Revisão Sistemática da Literatura. 7, 9, 17, 25, 29, 30, 33, 36–39, 54, 56, 102

SaaS *Software as a Service*. 48

SDK *Software Development Kit*. 57, 64, 81

SGBD Sistema de Gerenciamento de Banco de Dados. 58, 63, 80, 81

SOA *Service-Oriented Architecture*. 13, 49, 58, 59

URI *Uniform Resource Identifier*. 72, 76

UUID *Universally Unique IDentifier*. 63, 72

VBP Valor Bruto da Produção Agropecuária. 5, 15

ZP Zootecnia de Precisão. 15, 16

Sumário

1 Introdução

1.1 Contextualização e Motivação

O termo Indústria 4.0 foi utilizado pela primeira vez em 2011, durante a feira de *Hannover Messe*, como forma de sintetizar as mudanças promovidas nos processos industriais através da informatização (GONZALEZ, 2016). Apesar do termo ser originalmente criado para atividade fabril, sua aplicação vai muito além das indústrias. Adotar novas tecnologias relacionadas à tendência da Indústria 4.0 contribui para a otimização e automatização de processos de negócios e para o aumento da produtividade das empresas (TADIM, 2020).

Segundo a Agência Brasileira de Desenvolvimento Industrial (ABDI), há grandes desafios para a economia brasileira, em especial para a indústria, devido ao fato de que o Brasil tem caído no *ranking* global de inovação, ocupando apenas a 69º posição. Apesar disso, os dados ainda apontam a quarta revolução industrial como uma oportunidade para o país. Dessa forma, a agência afirma que as principais tecnologias que permitem a fusão dos mundos físico, digital e biológico, são: a Manufatura Aditiva, a Inteligência Artificial, a Biologia Sintética, os Sistemas Ciber-Físicos e a Internet das Coisas (ABDI, 2020).

Entre as áreas econômicas que podem se beneficiar dessas tecnologias, destaca-se a agropecuária, que é uma atividade que possui grande relevância no mercado brasileiro. Além disso, ela apresenta desdobramentos significativos no âmbito do comércio internacional (IBGE, 2020). O Ministério da Agricultura (2021) estimou que o Valor Bruto da Produção Agropecuária (VBP) do Brasil em 2021 deverá atingir um recorde de 1,076 trilhão de reais, alta de 12,1% na comparação anual, obtendo máximas em vários segmentos do agronegócio.

Dentro do setor agropecuário, existe a Agricultura de Precisão (AP) e Zootecnia de Precisão (ZP), que segundo Martello (2017), tem ganhado espaço no setor agropecuário, sobretudo quando se entende que há uma necessidade cada vez maior para monitoramento e controle dos processos que envolvem a produção animal e agrária. No entanto, a massificação da aplicação de sistemas na Zootecnia depende dos avanços das pesquisas nessa área (MARTELLO, 2017). Segundo Associação Brasileira de Internet das Coisas (ABINC), a inserção da internet das coisas no campo tende a aumentar os benefícios da AP, tornando as fazendas verdadeiramente inteligentes. Com a ajuda de sensores é possível automatizar o sistema de irrigação e monitorar as condições do campo à distância (ABINC, 2020). Entre as tecnologias e cenários comentados anteriormente, destaca-se que a internet das coisas pode ser aplicada na construção de mecanismos que auxiliam na modernização da

AP e ZP, além de outras aplicações na Indústria 4.0 e em outros ambientes inteligentes.

O termo Internet das Coisas (do inglês *Internet of Things* - IoT) é creditado a Kevin Ashton, pois, em 1999, ele iniciou uma apresentação intitulada *"That 'Internet of Things' Thing"* (KRAMP; KRANENBURG; LANGE, 2013). No entanto, essa realidade foi prevista por Mark Weiser, quando propôs a ideia de um mundo adaptativo e inteligente em segundo plano completamente onipresente (WEISER, 1991).

Na expressão *Internet of Things*, o termo "Internet" vem da infraestrutura de rede global com recursos escalonáveis e configuráveis com base em protocolos de comunicação. Já o termo "*Things*" são objetos físicos ou dispositivos, objetos virtuais (dispositivos ou informações) que possuem uma identidade e atributos (BANDYOPADHYAY et al., 2011). Portanto, IoT refere-se a objetos e sistemas que podem constituir comunicação e compartilhar informações por meio da Internet.

Os sensores e atuadores são componentes fundamentais na *Internet of Things*. Os sensores são capazes de medir e responder a estímulos físicos, semelhantes a algumas estruturas encontradas no corpo humano, como a retina ou os padrões de voz. Além disso, são capazes de detectar mudanças em sua localização, posição, orientação, elevação, velocidade ou distância de movimento (HASSAN, 2018). Os atuadores são capazes de fazer mudanças na composição química de uma entidade e responder a níveis alterados de calor, luz ou som detectados pelos sensores (BUNZ; MEIKLE, 2017). Portanto, a função desses sensores e atuadores é detectar, registrar, atuar nas mudanças e compartilhar dados.

O objetivo da IoT é obter sinergia entre diferentes coisas por meio da Internet. Isso significa que elas devem interoperar e se comunicar automaticamente para fornecer serviços inovadores (HASSAN; KHAN; MADANI, 2018). Portanto, a padronização é necessária para garantir que as plataformas IoT permitam que diferentes sistemas interoperem com sucesso.

No entanto, diversos protocolos de aplicação foram disponibilizados, entre os quais destacam-se: o *Constrained Application Protocol* (CoAP), o *Message Queuing Telemetry Transport* (MQTT), o *Advanced Message Queuing Protocol* (AMQP) e *Hypertext Transfer Protocol/Representational State Transfer* (HTTP/REST) (SZTAJNBERG; STUTZEL; MACEDO, 2018), cada um com características próprias e objetivos distintos. Além disso, várias opções estão disponíveis na camada física para comunicação de rede sem fio, como Wi-Fi, LoRa, ZigBee, BLE (TEEL, 2018). Qualquer solução para IoT deve ser capaz de se comunicar com diferentes protocolos ou ser escalonável para incluir outras formas de comunicação. Assim, para se comunicar com diferentes dispositivos que utilizam diferentes protocolos e redes, é necessária uma camada intermediária chamada *middleware*.

Um *middleware* é uma interface entre a camada de *hardware* e a camada de aplicação, responsável por interagir com os dispositivos e gerenciar informações (CHAQFEH;

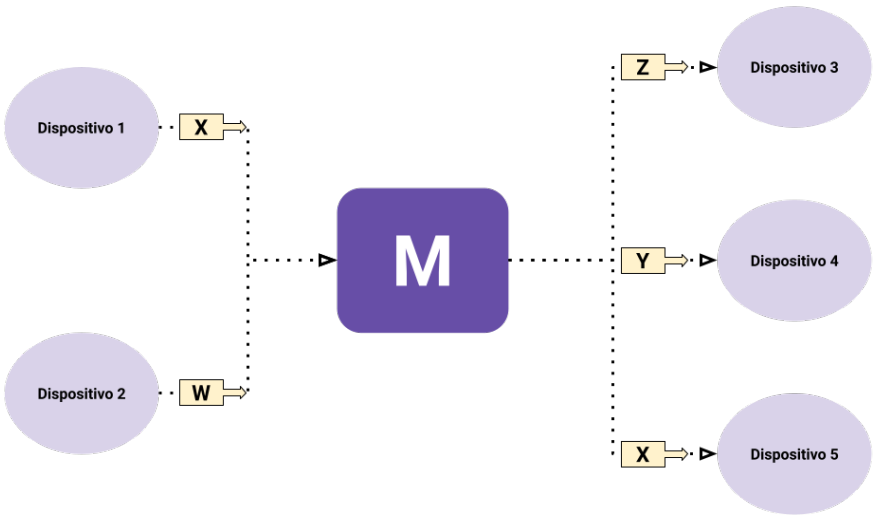
MOHAMED, 2012). O papel do *middleware* é apresentar um modelo de programação unificado para interagir com dispositivos. Além disso, o *middleware* para IoT é responsável por mascarar os problemas de heterogeneidade e distribuição que existem entre os dispositivos IoT (MULLIGAN, 2009). Além disso, oferecer a comunicação multiprotocolo não é uma tarefa trivial. Diante disso, um *middleware* multiprotocolo para IoT possui desafios relacionados a cada protocolo disponibilizado. Ademais, outros desafios podem ser destacados: a distribuição de dados entre dispositivos não relacionados; a ausência de controle pelas aplicações com os dispositivos e suas funcionalidades; e, uma perda histórica de dados para futuros processamentos e aplicações inteligentes (mais detalhes na Subseção 1.1.1).

Baseado nesses desafios, algumas questões de pesquisa e objetivos foram definidos para investigar, elaborar e oferecer uma arquitetura de *middleware* que possibilite a construção de um *middleware* multiprotocolo para auxiliar na construção de aplicações para internet das coisas, principalmente, para facilitar na construção de aplicações para agricultura de precisão. Para isso, uma Revisão Sistemática da Literatura (RSL) foi definida e executada para identificar e analisar os middlewares usados em aplicações para IoT. Como resultado, foram encontrados os middlewares mais recentes. No entanto, eles não disponibilizam todas as funcionalidades para enfrentar os desafios de um *middleware* multiprotocolo para IoT.

1.1.1 Desafios de um *Middleware* Multiprotocolo para IoT

Um *middleware* multiprotocolo, na sua essência, deve disponibilizar e intermediar a troca de dados entre protocolos de comunicação distintos. Na Figura 1, é possível visualizar um exemplo desse mecanismo de tradução entre protocolos (relacionamento N:N).

Figura 1 – Comunicação entre dispositivos gerenciados por um *middleware* multiprotocolo.



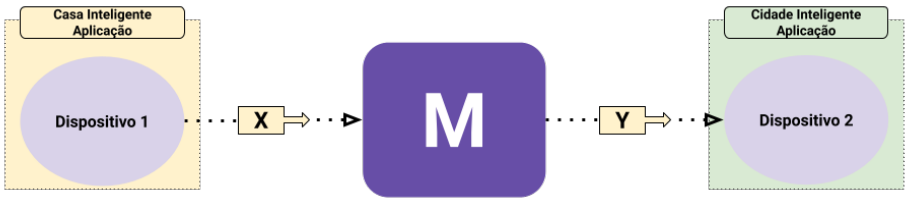
Fonte: autoria própria.

No exemplo apresentado, dois dispositivos (1 e 2) produzem dados que são enviados utilizando dois protocolos diferentes (X e W). A função de *middleware* multiprotocolo (destacado na Figura 1 como M) é traduzir essas informações e enviar para os dispositivos (3, 4 e 5) que estão conectados utilizando outros protocolos de comunicação (Z, Y e X).

No entanto, somente com essa abordagem, pelo menos três problemas ocorrem:

- **Distribuição de dados entre dispositivos não relacionados** (Figura 2): no exemplo, um dispositivo (dispositivo 1) envia um dado com um protocolo (protocolo X) para o *middleware*. O *middleware* traduz e envia esse dado por um protocolo (protocolo Y) para outro dispositivo (dispositivo 2).

Figura 2 – Comunicação entre dispositivos de aplicações diferentes.

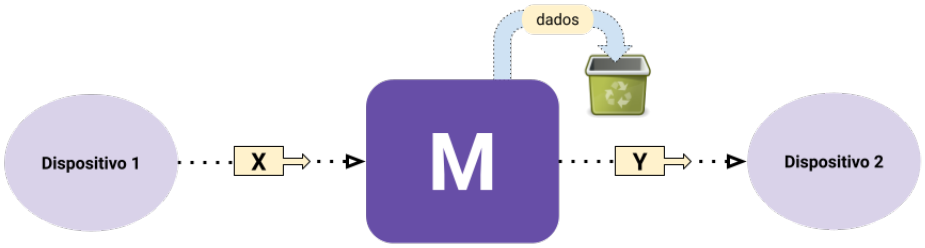


Fonte: autoria própria.

No entanto, os dispositivos estão inseridos em aplicações diferentes: o dispositivo 1 está em uma aplicação para casa inteligente; e, outro dispositivo 2 está em uma aplicação para cidade inteligente. Assim, o dispositivo 2 está recebendo um dado relacionado com outro domínio. Dessa forma, o *middleware* necessita de mecanismos para restringir a distribuição de dados entre aplicações distintas.

- **Uma perda histórica de dados para futuros processamentos e aplicações inteligentes** (Figura 3): no exemplo, um dado é gerado por um dispositivo (dispositivo 1) e enviado por um protocolo (protocolo X) para o *middleware*. O *middleware* recebe o dado e envia por um protocolo (protocolo Y) para um dispositivo (dispositivo 2). No entanto, o mesmo dado é descartado, pois não existem mecanismos de gerenciamento de dados. Portanto, um *middleware* multiprotocolo necessita de mecanismos de gerenciamento de dados produzidos pelos recursos.

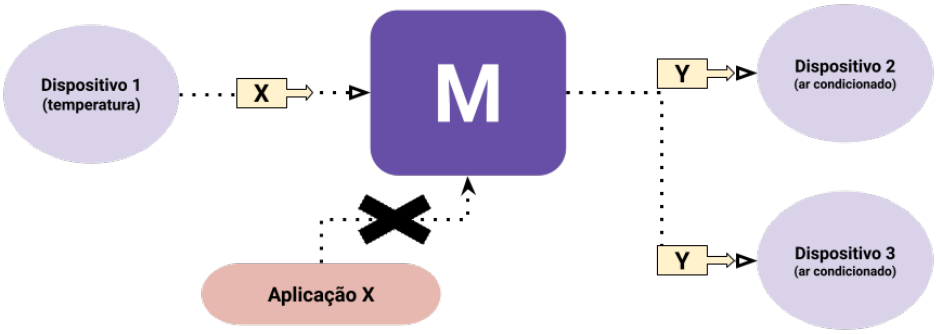
Figura 3 – Comunicação entre dispositivos com descarte de dados.



Fonte: autoria própria.

- **Ausência de controle pelas aplicações com os dispositivos e suas funcionalidades** (Figura 4): nesse cenário, o *middleware* não fornece formas de controle da distribuição desses dados para uma aplicação (aplicação X). Assim, ele recebe e envia por protocolos distintos, mas deixa as aplicações sem controle de suas funções com os dispositivos.

Figura 4 – Comunicação entre dispositivos sem mecanismos de controle



Fonte: autoria própria.

1.2 Questões de Pesquisa

Foram definidas questões de pesquisa baseadas na contextualização e motivação do trabalho. Observa-se que as questões de pesquisa definidas, partem da seguinte questão geral de pesquisa:

- ***Como facilitar a construção de aplicações para a internet das coisas que integram dispositivos heterogêneos?***

De acordo com Wieringa (2014) a questão geral pode ser decomposta em, pelo menos, dois tipos de questões: questões de conhecimento descritivas (QC) e questões de conhecimento sobre *design*/projeto (QDP), conforme apresenta-se a seguir.

1.2.1 Questões de Conhecimento

- QC 1: Quais *middlewares* foram construídos e utilizados em aplicações para internet das coisas?
  - QC 1.1: Quais requisitos são ou deveriam ser atendidos em *middlewares* para internet das coisas?
  - QC 1.2: Quais protocolos de comunicação são utilizados por *middlewares* para internet das coisas?
  - QC 1.3: Qual arquitetura de *middleware* pode ser utilizada para construir um *middleware* multiprotocolo para internet das coisas?

As questões de conhecimento sobre *design*/projeto podem ser decompostas em pelo menos quatro tipos: questões de efeito (QE), questões de *trade-off* (QT), questões de sensitividade (QPS) e questões de satisfação de requisitos (QSR). A seguir são descritas as questões de projeto.

1.2.2 Questões de *Design/Projeto*

Questões de Efeito

- QE 1: Como os dispositivos serão gerenciados?
- QE 2: Como os dados serão produzidos e armazenados pela interação e funcionamento dos dispositivos?
- QE 3: Como as aplicações poderão controlar seus dispositivos?

Questões de Sensitividade

- QPS 1: Como ocorrerá a comunicação entre dispositivos quando os protocolos forem os mesmos e quando forem diferentes?

Questões de Satisfação de Requisitos

- QSR 1: Os dispositivos que realizam a comunicação por protocolos diferentes conseguem enviar e receber informações uns dos outros?

Questões de *Trade-Off*

- QT1: Como middlewares, com propósitos similares, interferem na facilidade de construção de uma aplicação na IoT?

As respostas das questões de *design*/projeto respondem e justificam a facilidade de construção de novas aplicações para IoT sem adaptações significativas.

1.3 Objetivos

O projeto de pesquisa, guiado pelas questões apresentadas previamente, busca alcançar os objetivos subdivididos em objetivo geral e objetivos específicos, conforme detalhados a seguir.

1.3.1 Objetivo Geral

O objetivo geral da pesquisa é: ***permitir a construção de aplicações para internet das coisas, que usam e conectam diversos dispositivos com protocolos de comunicação diferentes, sem a necessidade de adaptações significativas***.

1.3.2 Objetivos Específicos

A partir das questões de pesquisa definidas e do objetivo geral, os objetivos específicos são:

- Definição de uma arquitetura de *middleware* multiprotocolo que atenda diferentes dispositivos e aplicações.
- Desenvolvimento e implantação de um *middleware* multiprotocolo que auxilie na construção de aplicações para internet das coisas.
- Construção de uma aplicação que utiliza o *middleware* desenvolvido para integrar e gerenciar os dispositivos que se comunicam por protocolos diferentes.

1.4 Organização do Documento

O restante deste trabalho está organizado da seguinte maneira:

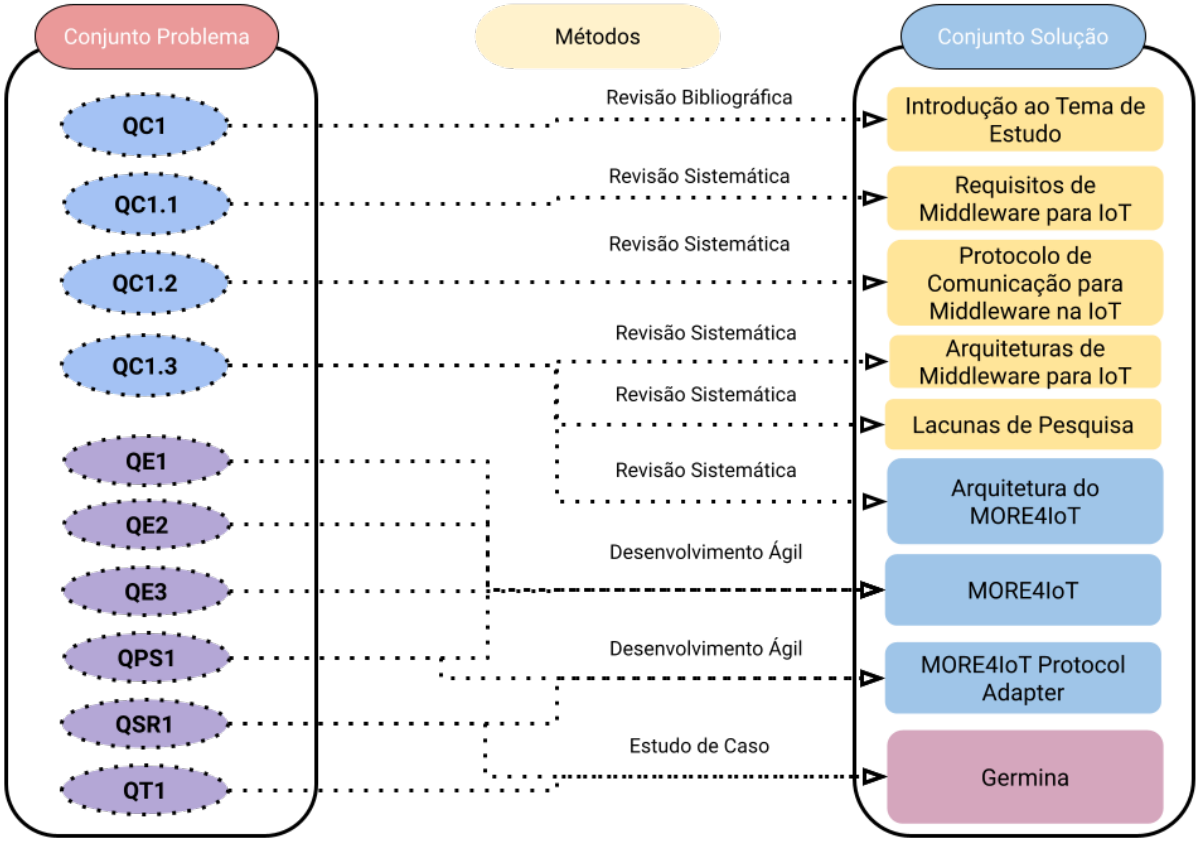
- **Capítulo 2:** detalhamento dos métodos e procedimentos utilizados neste trabalho. Nesse capítulo, são apresentados os métodos: revisão bibliográfica, revisão sistemática da literatura, o desenvolvimento ágil do *middleware* e o estudo de caso.
- **Capítulo 3:** apresentação da condução da revisão sistemática da literatura, a partir da execução do protocolo detalhado no Capítulo 2. Além disso, os resultados e os trabalhos relacionados são apresentados e discutidos.

- **Capítulo 4:** apresentação do *middleware* multiprotocolo orientado a microsserviços para internet das coisas. São definidos os requisitos, a arquitetura orientada a microsserviços e as funcionalidades.
- **Capítulo 5:** estudo de caso utilizando o *middleware* para construção de uma aplicação que utiliza os mecanismos de comunicação e gerenciamento disponibilizados.
- **Capítulo 6:** apresentação da conclusão, contribuições do trabalho e trabalhos futuros.

2 Métodos e Procedimentos

Este trabalho de mestrado é orientado pela perspectiva do paradigma *Design Science*. De acordo com Hevner et al. (2004), *Design Science* é uma técnica de investigação e resolução de problemas por meio de inovações que definam ideias, práticas e produtos, no qual a análise, projeto e implementação podem ser realizados. Além disso, nessa técnica, métodos específicos da pesquisa devem conduzir o conjunto problema ao conjunto solução. Na Figura 5, apresenta-se os métodos utilizados pelo trabalho, conectando as questões de pesquisa do conjunto problema aos objetos produzidos no conjunto solução.

Figura 5 – Métodos e procedimentos utilizados no trabalho



Fonte: autoria própria.

Um breve resumo dos métodos e procedimentos utilizados pela pesquisa:

- Revisão Bibliográfica: estudo de artigos, teses, livros e outras publicações relacionadas com *middleware* para internet das coisas, encontrados em repositórios como *IEEE Xplore Digital Library* e *ACM Digital Library* com o objetivo de capturar informações relevantes da temática pesquisada.

- Revisão Sistemática: pesquisa elaborada formalmente por um processo sistemático (BRERETON et al., 2007), com o objetivo de organizar o estado da arte sobre *middleware* para internet das coisas e arquiteturas para *middleware*, bem como requisitos e protocolos de comunicação utilizados. Mais detalhes são apresentados na Seção 2.1.

- Metodologia Ágil: o desenvolvimento ágil foi utilizado para a entrega do middleware. A metodologia ágil emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos (ALLIANCE, 2017). Neste trabalho, alguns princípios ágeis são utilizados para desenvolver e refinar as funcionalidades do *middleware*, validar os requisitos e construir a aplicação que utiliza o *middleware*. Mais detalhes são apresentados na Seção 2.2.

- Estudo de Caso: método de pesquisa observacional que investiga um fenômeno por meio de um caso específico (RUNESON, 2012). Neste trabalho, esse método possibilita validar as funcionalidades da infraestrutura de *middleware* construída em uma aplicação na IoT. Mais detalhes são apresentados no Capítulo 5.

Inicialmente, a revisão da bibliografia foi realizada para introduzir o tema de estudo e utilizada para auxiliar na definição do protocolo da revisão sistemática. Com o protocolo definido, a revisão foi executada e os resultados são expostos e discutidos no Capítulo 3. Como resultado da revisão sistemática, diversos middlewares para IoT foram encontrados e as lacunas de pesquisa foram detectadas. No entanto, não foi possível identificar soluções completas para os desafios de um *middleware* multiprotocolo. Dessa forma, este trabalho utilizou as principais características, pontos importantes e lacunas detectadas de um *middleware* IoT para definir os requisitos do *middleware* proposto.

Ademais, para o desenvolvimento do *middleware*, o desenvolvimento ágil foi utilizado, conforme comentando anteriormente. Na versão atual, o *middleware* é composto por sete componentes internos que possuem responsabilidades bem definidas para enfrentar os desafios de um *middleware* multiprotocolo na IoT. Além disso, os componentes externos auxiliam na construção das aplicações que usam e conectam dispositivos com protocolos diferentes, conforme detalhado no Capítulo 4.

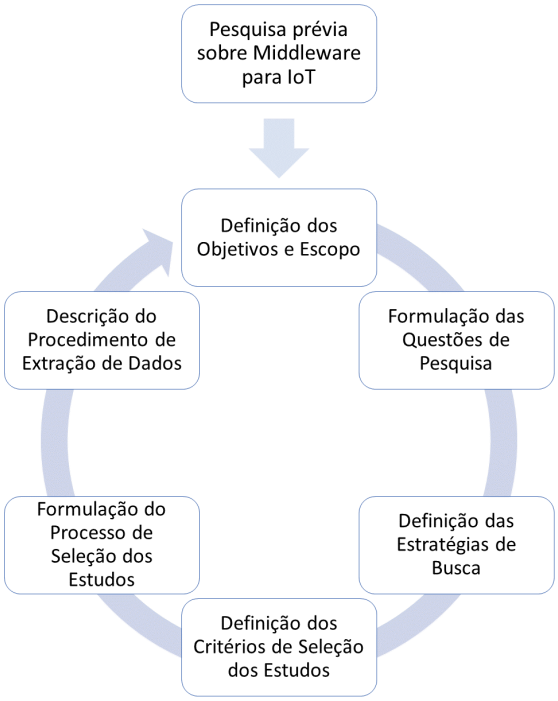
Ao final do desenvolvimento do *middleware*, deu-se início ao estudo de caso para definir e desenvolver uma aplicação para agricultura de precisão, conforme detalhado no Capítulo 5. Para isso, um estudo prévio do assunto foi realizado, os requisitos da aplicação foram identificados, a arquitetura foi definida e a prototipação das telas da aplicação foram desenvolvidas. Um protótipo da aplicação foi construído para validar as funcionalidades do *middleware* utilizando as ferramentas na construção da comunicação por diferentes protocolos. Por fim, no Capítulo 6, apresenta-se as conclusões e os trabalhos futuros.

2.1 Revisão Sistemática da Literatura

Uma revisão informal da literatura está sujeita a viés de investigação, como na formulação de questões de pesquisa, coleta, avaliação, interpretação e apresentação de dados, que podem ser influenciados pelos interesses pessoais dos pesquisadores envolvidos, levando a resultados ruins e não confiáveis (BIOLCHINI et al., 2005). As revisões informais da literatura, que são realizadas sem um planejamento a priori, são caracterizadas como incompletas, não repetíveis, não confiáveis e cuja qualidade depende da experiência dos pesquisadores (MAFRA; TRAVASSOS, 2006).

Em contraste, uma Revisão Sistemática da Literatura (RSL) é conduzida por meio de um processo composto por uma sequência bem definida de fases e apresenta uma avaliação justa sobre um tópico de pesquisa, usando um formulário de revisão rigoroso, confiável e auditável (BIOLCHINI et al., 2005). O protocolo de uma revisão sistemática tem como objetivo principal reduzir os erros que podem ocorrer durante a execução de uma RSL (FELIZARDO et al., 2017).

Figura 6 – Etapas do planejamento do protocolo da revisão sistemática da literatura



Fonte: autoria própria.

O protocolo definido nesta pesquisa segue o processo definido por Biolchini *et al.* (BIOLCHINI et al., 2007) e usa algumas das diretrizes definidas por Kitchenham *et al.* (KITCHENHAM, 2004). Assim, o protocolo engloba objetivos, questões de pesquisa, estratégias de busca, critérios de seleção de estudos, processo de seleção de estudos e procedimentos de extração de dados, conforme apresentado na Figura 6. Mais detalhes do

protocolo e da condução são apresentados no Capítulo 3.

2.2 Metodologia Ágil

Alguns objetivos foram definidos para essa etapa. A construção dos artefatos, descritos a seguir, é muito importante. Para atingir esses objetivos, em seguida, são descritos alguns princípios ágeis que foram utilizados para o desenvolvimento.

2.2.1 Objetivos

- Desenvolvimento e refinamento das funcionalidades do *middleware* multiprotocolo para internet das coisas
- Validação dos requisitos do *middleware*
- Construção da aplicação que utiliza o *middleware*

O ciclo de desenvolvimento ágil é uma abordagem que utiliza o interativo e incremental para refinar o produto e realizar entregas frequentes (ALLIANCE, 2017). Neste trabalho, os princípios ágeis, descritos a seguir, são utilizados para direcionar o desenvolvimento ágil do *middleware* e dos outros objetivos.

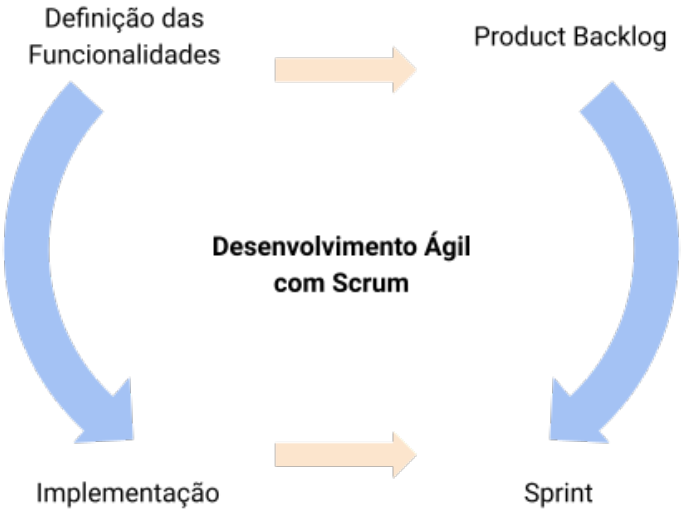
- Geração de valor contínuo;
- Adaptabilidade e versatilidade;
- Pequenas entregas em curtos espaços de tempo;
- Trabalho em conjunto;
- *Software* otimizado e que funciona bem;
- Processo de desenvolvimento sustentável e duradouro;
- Aprendizados e aperfeiçoamentos constantes;

Para facilitar a organização do desenvolvimento ágil do *middleware*, algumas etapas do Scrum, definido por Schwaber (2004), são utilizadas. O Scrum é um *framework* para desenvolver, entregar e manter produtos. Assim, pode-se tratar e resolver problemas complexos e adaptativos, enquanto entrega-se produtos com o mais alto valor possível. Ele é leve e simples de entender, mas difícil de dominar, pois consiste em papéis, eventos e artefatos. Neste trabalho, o Scrum é adaptado para otimizar os benefícios de sua utilização.

2.2.2 Desenvolvimento Ágil

O desenvolvimento ágil baseado em algumas etapas do Scrum pode agilizar o processo de desenvolvimento do *middleware* e da aplicação que utiliza o *middleware*. Para isso, algumas adaptações do *framework* ágil são descritas a seguir. Duas partes do Scrum são importantes para esse desenvolvimento (Figura 7): o *product backlog* e a *sprint*.

Figura 7 – Desenvolvimento ágil utilizando etapas do Scrum



Fonte: autoria própria.

- *Product Backlog*: é uma lista ordenada de tudo o que pode ser necessário no produto e a única fonte para quaisquer alterações a serem feitas. Ele evolui, conforme o produto e o ambiente no qual ele será usado evolui (SCHWABER, 2012). Ao iniciar o desenvolvimento de um *software*, define-se os requisitos e suas prioridades. Com isso, os requisitos são transformados em uma lista de funcionalidades e inseridas no *backlog*.

- *Sprint*: a *sprint* é uma etapa importante no desenvolvimento ágil. A cada *sprint* um subconjunto de funcionalidades são selecionadas e cria-se uma caixa de tempo com duração de um mês ou menos durante a qual um incremento do produto é criado, tornando-se utilizável e potencialmente liberável (SCHWABER, 2012). *Sprints* têm durações consistentes ao longo de um esforço de desenvolvimento. Uma *sprint* começa imediatamente após a conclusão de uma *sprint* anterior.

2.2.3 Prototipação

Um protótipo é uma versão inicial de um sistema de software, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis

soluções (SOMMERVILLE, 2016). O desenvolvimento rápido e interativo do protótipo é essencial para que os custos sejam controlados e os *stakeholders* do sistema possam experimentá-lo no início do processo de software (PRESSMAN, 2015). Um protótipo de software pode ser usado em um processo de desenvolvimento de software para ajudar a antecipar as mudanças que podem ser requisitadas. Para tanto, a prototipação é utilizada no estudo de caso para o desenvolvimento da aplicação na IoT utilizando o *middleware* proposto.

2.3 Considerações Finais

Neste capítulo, apresentou-se os métodos e procedimentos utilizados no trabalho. Para tanto, um revisão bibliográfica foi realizada para auxiliar na construção do protocolo da revisão sistemática sobre middlewares para internet das coisas. Com o protocolo da revisão definido, a condução da revisão foi executada. Ao término, as lacunas de pesquisa foram encontradas e auxiliaram na definição do MORE4IoT e seus requisitos. Com os requisitos definidos, foi utilizado os princípios ágeis para desenvolver e refinar as funcionalidades do *middleware*. Por fim, um estudo de caso foi definido e desenvolvido para validar as funcionalidades do MORE4IoT. Os detalhes do estudo de caso são descritos no Capítulo 5. No próximo capítulo, descreve-se o planejamento e condução da revisão sistemática.

3 Planejamento e Condução da Revisão Sistemática da Literatura

Neste capítulo, apresenta-se o planejamento e execução do protocolo da Revisão Sistemática da Literatura (RSL). Assim, os seguintes itens são detalhados: o protocolo da RSL, o processo de seleção dos artigos; os resultados da RSL; e, as lacunas de pesquisa.

3.1 Objetivos e Escopo

O objetivo principal desta revisão sistemática é identificar os middlewares projetados para auxiliar no desenvolvimento de aplicações para a IoT. Também é importante identificar os requisitos e protocolos de comunicação usados por esses middlewares. Outra parte fundamental é encontrar as lacunas de pesquisa sobre middlewares para a IoT. Portanto, os seguintes objetivos secundários foram definidos:

- **Objetivo 1:** identificação de middlewares utilizados em aplicações para Internet das Coisas.
- **Objetivo 2:** identificação das arquiteturas dos *middlewares*.
- **Objetivo 3:** identificação dos requisitos de *middleware* utilizados.
- **Objetivo 4:** identificação dos protocolos de comunicação utilizados pelos *middlewares*.
- **Objetivo 5:** comparação dos *middlewares* identificados em relação à arquitetura, requisitos e protocolos de comunicação.
- **Objetivo 6:** identificação de lacunas de pesquisa relacionadas com middlewares para IoT.

Ao final da pesquisa, o resultado esperado é obter uma definição do estado da arte para IoT em relação aos middlewares existentes e as lacunas de pesquisa. De acordo com os objetivos da RSL, os itens (Subseção 3.1.1) foram selecionados como específicos para o escopo da revisão.

3.1.1 Especificidades do Escopo

Essas especificidades de escopo, descritas a seguir, visam definir o alcance das questões de pesquisa. Entre elas estão: intervenção, controle, população, resultados, e

aplicação. Intervenção descreve o que será observado no contexto da revisão sistemática; Controle é o conjunto inicial de dados conhecidos antes do início do protocolo de revisão sistemática e utilizado para avaliar a *string* de pesquisa, verificando se a pesquisa realizada encontrou, pelo menos, os artigos já conhecidos; População refere-se ao grupo que será observado pela intervenção; Resultados listam os resultados esperados desta revisão sistemática da literatura; a Aplicação define as funções, profissionais ou áreas de aplicação que se beneficiarão com os resultados desta RSL.

Intervenção:

Trabalho, que descreve *middleware*, no contexto de interconexão de dispositivos (*Things*).

Controle:

- Georgi, N; Corvol, A.; Le Bouquin Jeannès, R. Middleware Architecture for Health Sensors Interoperability. IEEE Access, volume 6, 2018.
- Jeon, S., Jung, I. MinT: Middleware for Cooperative Interaction of Things. Sensors, volume 17, issue 6, 2017.
- Lalanda, P.; Morand, D.; Chollet, S.Autonomic Mediation Middleware for Smart Manufacturing. IEEE Internet Computing, volume 21, 2017.

População:

Trabalhos que descrevem middlewares utilizados no desenvolvimento de aplicações, revisões sobre middlewares, protocolos de comunicação e requisitos de *middleware* para IoT.

Resultados:

Middlewares para IoT, modelos e padrões arquiteturais, requisitos e protocolos de comunicação utilizados por middlewares para IoT.

Aplicação:

Pela indústria, pesquisadores, professores, engenheiros de software e profissionais autônomos interessados no conhecimento e desenvolvimento sobre *middleware* para IoT.

3.2 Questões de Pesquisa

A fim de satisfazer os objetivos da revisão sistemática da literatura, algumas questões de pesquisa foram formuladas. Dessa forma, uma questão primária e algumas questões secundárias foram definidas:

- **Questão principal (QP):** Quais *middlewares* foram construídos e utilizados em aplicações para IoT?
- **Questão secundária (QS1):** Quais arquiteturas foram utilizadas para construir esses *middlewares*?
- **Questão secundária (QS2):** Quais requisitos foram atendidos por esses *middlewares*?
- **Questão secundária (QS3):** Quais protocolos de comunicação foram utilizados por esses *middlewares*?

3.3 Estratégia de Busca

Para realizar uma busca de estudos primários, a estratégia consistiu em uma busca *online* nas cinco principais bibliotecas digitais que apresentam uma forte ênfase em Ciência da Computação. Além disso, foi necessário definir as palavras-chave e seus sinônimos para construir a *string* de busca a ser aplicada em cada plataforma selecionada. Por fim, foram definidos os parâmetros e as especificidades para a execução da pesquisa em cada repositório.

Bases de Trabalhos Selecionadas

- IEEE Xplore Digital Library[1]
- ACM Digital Library Library[2]
- ScienceDirect [3]
- Scopus [4]
- Web Of Science [5]

Antes de construir uma *string* de pesquisa padrão, foram definidas as palavras-chave relacionadas às questões de pesquisa. Para ampliar o escopo da pesquisa, foram definidos os sinônimos para cada palavra-chave conforme mostrado na Tabela 1.

Com as palavras-chave definidas, a *string* de pesquisa padrão foi criada usando os operadores lógicos. Sinônimos, abreviações e plurais de palavras-chave também foram incluídos:

[1]<https://ieeexplore.ieee.org/Xplore/home.jsp>
[2]<https://dl.acm.org/>
[3]<https://www.sciencedirect.com/>
[4]<https://www.scopus.com/>
[5]<https://www.webofknowledge.com>

Tabela 1 – Palavras-chave e sinônimos relacionados às questões de pesquisa

Palavra-chave	**Sinônimos**
Internet of Things	IoT, Future Internet, Web of Things, WoT and Industry 4.0
Middleware	Middle Software

- ***String* de Busca Padrão**: "("internet of thing" OR "internet of things" OR "iot" OR "web of thing" OR "web of things" OR "wot" OR "future internet" OR "industry 4.0") AND ("middleware" OR "middle software")"

A *string* de pesquisa foi validada e refinada por meio de uma pesquisa piloto realizada na Biblioteca Digital IEEE Xplore e foi adaptada para as plataformas selecionadas. As bibliotecas digitais selecionadas têm características diferentes, incluindo sintaxe. Por isso, a *string* padrão definida foi adaptada conforme apresenta-se na Tabela 2.

Tabela 2 – *String* de pesquisa padrão adaptada para cada biblioteca digital

Library	**Search String**
IEEE	("internet of thing" OR "internet of things" OR "iot" OR "web of thing" OR "web of things" OR "wot" OR "future internet" OR "industry 4.0") AND ("middleware" OR "middle software")
ACM	(recordAbstract:("internet of thing" "internet of things" "iot" "web of thing" "web of things" "wot" "future internet" "industry 4.0") AND recordAbstract:(middleware "middle software")) OR (acmdlTitle:("internet of thing" "internet of things" "iot" "web of thing" "web of things" "wot" "future internet" "industry 4.0") AND acmdlTitle:(middleware "middle software"))
Science Direct	("internet of thing" OR "internet of things" OR "iot" OR "web of thing" OR "web of things" OR "wot" OR "future internet" OR "industry 4.0") AND ("middleware" OR "middle software")
SCOPUS	TITLE-ABS-KEY(("internet of thing" OR "internet of things" OR "iot" OR "web of thing" OR "web of things" OR "wot" OR "future internet" OR "industry 4.0") AND ("middleware" OR "middle software")) AND (LIMIT-TO(SUBJAREA,"COMP"))
Web Of Science	(TS=(("internet of thing" OR "internet of things" OR "iot" OR "web of thing" OR "web of things" OR "wot" OR "future internet" OR "industry 4.0") AND ("middleware" OR "middle software"))) OR (TI=(("internet of thing" OR "internet of things" OR "iot" OR "web of thing" OR "web of things" OR "wot" OR "future internet" OR "industry 4.0") AND ("middleware" OR "middle software"))) AND SU=("Computer Science")

Devido a algumas características particulares de cada plataforma, as dificuldades e observações foram documentadas, conforme apresenta-se a seguir:

- **IEEE Xplore**: a pesquisa foi realizada usando a página de pesquisa avançada e

a opção de pesquisa por comando. A busca foi restrita a títulos e resumos das publicações, selecionando a opção "Metadata Only" na busca avançada.

- **ACM**: a busca foi restrita a títulos de publicações e resumos, adicionando as palavras "recordAbstract:" e "cmdlTitle:" antes dos termos de busca. Além disso, observa-se que a *string* foi inserida no campo especial "show query syntax" da página de pesquisa avançada da plataforma.
- **ScienceDirect**: a pesquisa foi restringida por títulos, resumos e palavras-chave adicionando a *string* no "Title, abstract or author-specified keywords" na página de pesquisa avançada.
- **SCOPUS**: adicionou-se o termo "TITLE-ABS-KEY" ao início da *string* de pesquisa para restringir a busca por títulos, resumos e palavras-chave. Além disso, foi adicionado um termo especificando a área de pesquisa para Ciência da Computação (LIMIT-TO (SUBJAREA, "COMP")).
- **Web Of Science**: a busca foi realizada usando a *tag* "TS" para tópicos e a *tag* "IT" para títulos. Além disso, foi adicionada a *tag* "SU" para restringir a busca pela área de Ciência da Computação.

3.4 Critérios de Seleção dos Estudos

Os artigos localizados foram avaliados por critérios de seleção definidos em duas categorias: inclusão e exclusão. Os critérios de inclusão objetivaram identificar os trabalhos com potencial para responder às questões de pesquisa. Os critérios de exclusão objetivaram excluir artigos não relacionados à esta RSL. Para a inclusão de um artigo em cada fase da análise, sendo necessário que ele atenda a pelo menos um dos critérios de inclusão. De maneira similar, os artigos que atendiam a, pelo menos, um dos critérios de exclusão foram descartados.

Critérios de Inclusão

- Critério de Inclusão (CI1): trabalhos que apresentem e apliquem um *middleware* para a internet das coisas.
- Critério de Inclusão (CI2): trabalhos que apresentem propostas de requisitos de *middleware* para a internet das coisas.
- Critério de Inclusão (CI3): trabalhos que apresentem protocolos de comunicação para *middlewares* aplicados em internet das coisas.
- Critério de Inclusão (CI4): trabalhos que apresentem uma revisão sistemática sobre *middlewares* para internet das coisas.

Critérios de Exclusão

Os estudos foram excluídos com base nos critérios de exclusão descritos a seguir. Cada critério definido pode ser justificado da seguinte forma: CE1, CE2, CE3 e CE4 se opõem aos critérios de inclusão, ou seja, são os critérios que excluem artigos que não sejam sobre *middleware*, arquitetura, requisitos, protocolos de comunicação ou trabalhos relacionados; CE5 e CE7 foram definidos porque a pesquisa de artigos curtos exige ainda mais recursos (tempo); e, as informações em artigos curtos podem não ser tão confiáveis ou completas, especialmente esses, que geralmente, apresentam resultados preliminares; CE8 foi definido porque este trabalho pretende reunir os middlewares mais recentes e encontrar lacunas que ainda estão abertas sobre *middleware* para a internet das coisas; e, CE10 e CE11 foram definidos porque esta pesquisa buscava os trabalhos mais importantes sobre *middleware* para IoT. Os outros critérios foram introduzidos para refinar a busca por artigos completos.

- Critério de Exclusão (CE1): trabalhos que apresentam um *middleware*, porém não são para o contexto da internet das coisas.
- Critério de Exclusão (CE2): artigos que apresentam requisitos, mas não são aplicáveis para *middleware* no contexto da internet das coisas.
- Critério de Exclusão (CE3): artigos que apresentam protocolos de comunicação, mas não são para *middleware* no contexto da internet das coisas.
- Critério de Exclusão (CE4): artigos que apresentam uma revisão sistemática, mas não cobrem *middleware* para a internet das coisas.
- Critério de Exclusão (CE5): estudos incompletos, indisponíveis ou duplicados.
- Critério de Exclusão (CE6): trabalhos escritos em idiomas diferentes do inglês ou do português.
- Critério de Exclusão (CE7): artigos com menos de cinco páginas.
- Critério de Exclusão (CE8): artigos publicados antes de 2016.
- Critério de Exclusão (CE9): versões antigas dos trabalhos selecionados.
- Critério de Exclusão (CE10): Fator de Impacto menor do que 0.9 para artigos de periódicos.
- Critério de Exclusão (CE11): Qualis Conferências inferior a A4 para artigos de conferências.

Critérios de Qualidade

Os estudos primários selecionados foram avaliados com base nos critérios de qualidade descritos a seguir. Os critérios de qualidade foram definidos para estimar a qualidade dos trabalhos encontrados sobre *middleware* para a internet das coisas. Existe um *trade-off* entre objetividade e subjetividade na avaliação da qualidade, mas procurou-se o equilíbrio nessa avaliação. Vale ressaltar que os critérios de qualidade não exclui artigos. O resultado é apresentado neste capítulo.

- Critério de Qualidade (CQ1): trabalhos com Fator de Impacto entre 0,9 e 1,9 ou Qualis A4 (1 ponto), entre 2,0 e 2,9 ou A3 (2 pontos), entre 3,0 e 5,0 ou A2 (3 pontos) e maior que 5,0 ou A1 (5 pontos).
- Critério de Qualidade (CQ2): *middleware* utilizado na construção de aplicações para resolução de problemas reais (5 pontos).
- Critério de Qualidade (CQ3): *middleware* aplicado na construção de aplicações para auxiliar a agricultura ou zootecnia de precisão (5 pontos).
- Critério de Qualidade (CQ4): trabalhos que fornecem o código desenvolvido (*open source*) (3 pontos).

3.5 Procedimento para Seleção dos Estudos

O processo de seleção dos estudos foi dividido em quatro fases: Coleta de Trabalhos, Seleção Inicial, Seleção Final e Validação da Seleção. As fases de seleção dos estudos são descritas em ordem cronológica a seguir.

- **Fase 1 - Coleta de Trabalhos:** realização das buscas nas bibliotecas digitais com a *string* definida, catalogando e removendo trabalhos duplicados.
- **Fase 2 - Seleção Inicial:** leitura de títulos e resumos de artigos com avaliação baseada nos critérios de seleção.
- **Fase 3 - Seleção Final:** leitura de artigos completos aprovados na fase anterior e reaplicação dos critérios de seleção.
- **Fase 4 - Validação da Seleção:** validação ou invalidação com base nos objetivos da pesquisa.

3.6 Extração de Dados e Apresentação dos Resultados

Para o procedimento de extração de dados, foi definida um formulário de extração de dados a fim de auxiliar na extração e incluir informações sobre os estudos lidos na íntegra e validadas de acordo com os critérios de seleção. O formulário contêm os campos necessários para responder às questões de pesquisa. Um formulário de extração de dados deve ser preenchido para cada artigo, incluindo o título, ano de publicação, fonte de publicação, fator de impacto, *middleware*, arquitetura, requisitos encontrados, protocolos de aplicação usados, vantagens, limitações e processo de validação. Na Tabela 3, apresenta-se o formulário completo com atributos escolhidos para caracterização dos estudos.

Tabela 3 – Atributos do formulário de extração de dados e suas descrições

Campos	**Descrição**
Título	Título do artigo selecionado
Ano	Ano de publicação do trabalho
Título de publicação	Nome da conferência, jornal ou livro em que o artigo foi publicado
Fator de Impacto	Identificação do fator de impacto do título de publicação
Número de Páginas	Número de páginas do trabalho
Linguagem de Escrita	Língua em que o artigo foi escrito
Middleware	Descrição do *middleware* apresentado
Arquitetura	Descrição da Arquitetura
Requisitos	Descrição dos requisitos de *middleware* (funcionais e não funcionais)
Protocolo de Comunicação	Descrição dos protocolos de comunicação aplicados
Vantagens	Descrição das vantagens do *middleware*
Validação	Descrição do processo de validação de *middleware*
Limitações	Descrição das limitações do *middleware*

Os dados extraídos são apresentados no Capítulo 3, por meio de uma síntese dos trabalhos aceitos, apresentação de tabelas comparativas, gráficos, bem como a apresentação das respostas para as questões de pesquisa definidas neste protocolo. A condução desta RSL foi apresentada e ilustrada por meio de gráficos e tabelas. Por fim, as palavras-chave dos estudos primários selecionados foram aplicadas ao software VOSViewer [6] (ECK; WALTMAN, 2010) a fim de construir uma rede entre a palavras-chave. Assim, foi possível compreender a escolha das palavras-chave definidas para esta revisão.

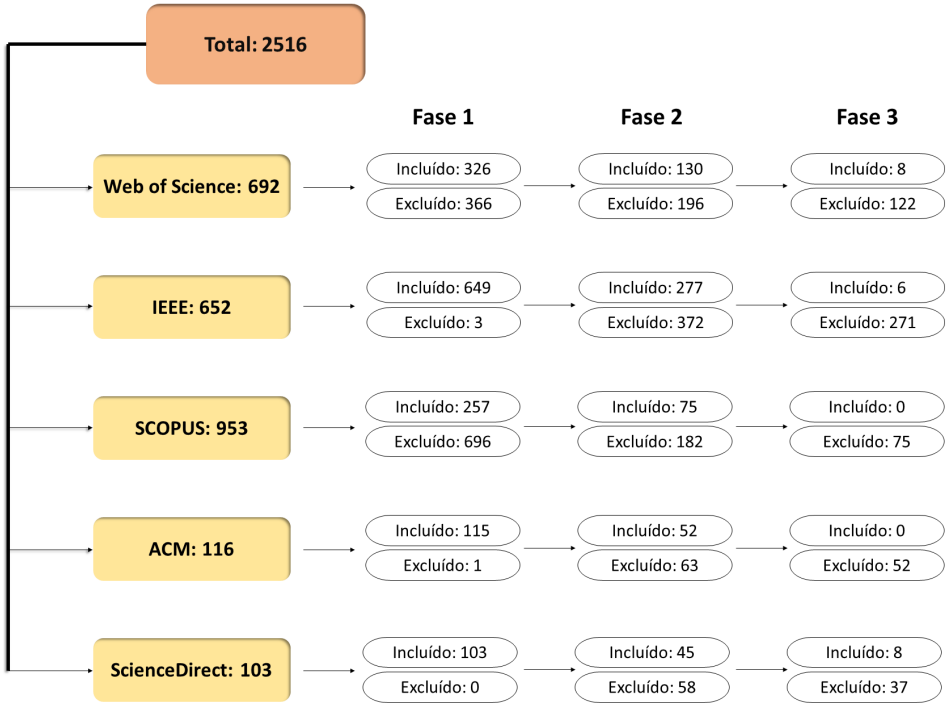
3.7 Coleta de Trabalhos

Inicialmente, foram realizadas buscas nas bases de dados selecionadas com as *strings* adaptadas para cada uma delas. Na Figura 8, apresenta-se o número de artigos coletados

[6] <https://www.vosviewer.com/>

por biblioteca digital, resultando em um total de 2516 artigos. Os trabalhos duplicados foram retirados, totalizando 1450 trabalhos individuais na primeira fase, entre um período de 2016 a abril de 2021.

Figura 8 – Os resultados da inclusão e exclusão dos estudos na RSL

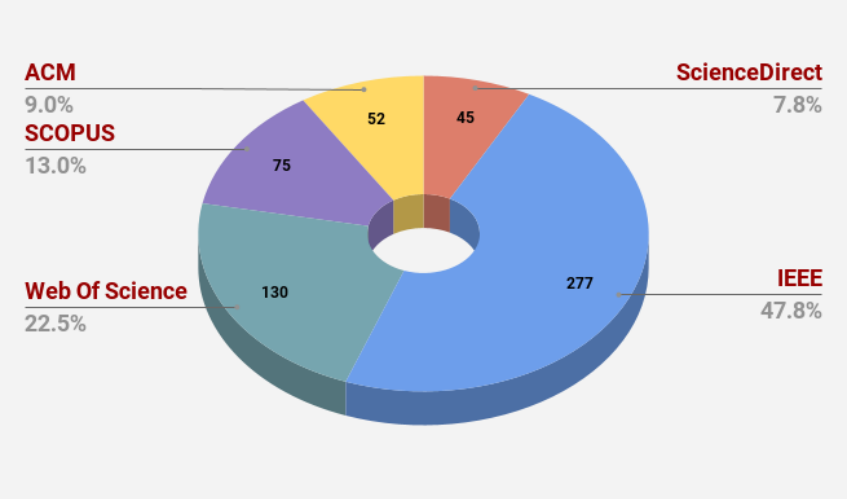


Fonte: autoria própria.

3.8 Seleção Inicial

Na seleção inicial (Fase 2), os artigos foram selecionados aplicando-se os critérios de inclusão e exclusão, a partir da leitura dos títulos e resumos dos artigos catalogados. Foram lidos 1450 resumos resultantes dos estudos remanescentes da Fase 1. Na Figura 8, apresenta-se o número de artigos incluídos e excluídos para cada plataforma. Além disso, apresenta-se a porcentagem dos trabalhos por biblioteca digital (Figura 9). Na Fase 2, a maior porcentagem de trabalhos incluídos foi da IEEE *Xplore* com 47.8%, seguida por Web of Science com 22.5%, SCOPUS (13%), ACM (9%) e ScienceDirect (7.8%). Ao final desta etapa, foram incluídos 579 trabalhos.

Figura 9 – Percentual de trabalhos selecionados resultante da segunda fase da execução da RSL

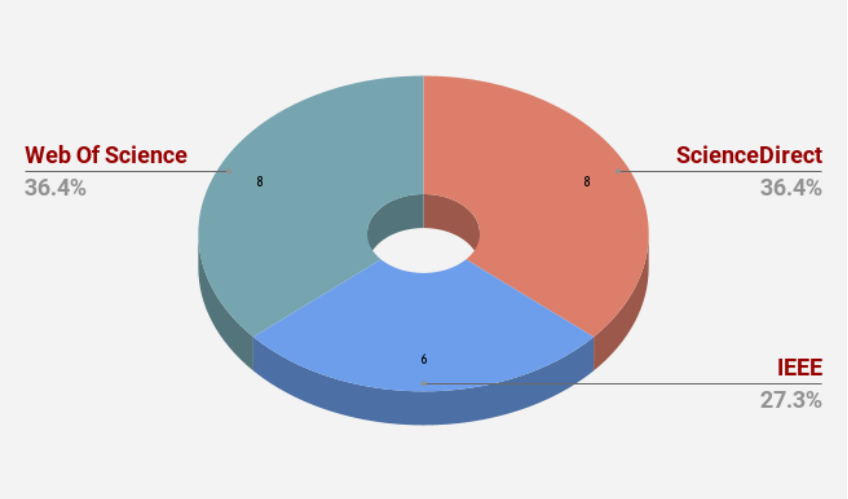


Fonte: autoria própria.

3.9 Seleção Final

Na última etapa da seleção (Fase 3), os artigos selecionados na fase anterior foram lidos e os critérios de inclusão e exclusão foram reaplicados. Em complemento, é possível visualizar a porcentagem obtida por biblioteca digital na Figura 10. Observa-se que as bases Web of Science, ScienceDirect e IEEE obtiveram porcentagens semelhantes com 36.4%, 36.4% e 27.3%, respectivamente.

Figura 10 – Percentual de trabalhos selecionados na terceira fase da execução da RSL



Fonte: autoria própria.

Por outro lado, nenhum artigo indexado pelas plataformas ACM e SCOPUS foi selecionado na Fase 3 da RSL. No entanto, diferentes plataformas de trabalhos podem indexar os mesmos periódicos e conferências. Por esse motivo, o mesmo número de trabalhos

pode ser obtido por outras plataformas que indexam os mesmos veículos de divulgação acadêmica.

Tabela 4 – Classificação dos trabalhos selecionados com base nos critérios de qualidade

ID	Autor	CQ1	CQ2	CQ3	CQ4	Total
1	Cruz et al. (2021)	5	5	0	3	13
2	Trilles, González-Pérez e Huerta (2020)	3	5	5	0	13
3	Zhao et al. (2018)	3	5	5	0	13
4	Antonić et al. (2016)	5	5	0	0	10
5	Xie et al. (2021)	5	5	0	0	10
6	Esposte et al. (2019)	5	0	0	3	8
7	Cruz et al. (2019)	5	0	0	3	8
8	Georgi, Corvol e Jeannes (2018)	3	5	0	0	8
9	Souza et al. (2019)	3	5	0	0	8
10	Gomes et al. (2017)	3	5	0	0	8
11	Alvisi et al. (2019)	3	5	0	0	8
12	Alvaro, Miguel e Molina (2019)	3	5	0	0	8
13	Lalanda, Morand e Chollet (2017)	2	5	0	0	7
14	Bouloukakis et al. (2019)	5	0	0	0	5
15	Benayache et al. (2019)	5	0	0	0	5
16	Mohamed et al. (2017)	3	0	0	0	3
17	Jeon e Jung (2017)	3	0	0	0	3
18	Lilis e Kayal (2018)	3	0	0	0	3
19	Dipsis e Stathis (2020)	3	0	0	0	3
20	Mesmoudi et al. (2018)	2	0	0	0	2
21	Merlino et al. (2019)	2	0	0	0	2
22	Shokrollahi e Shams (2017)	1	0	0	0	1

A validação dos artigos foi a última fase (Fase 4) da execução da RSL. Nessa fase, os artigos selecionados foram validados, durante as etapas de execução, por dois pesquisadores.

Na Tabela 4, os estudos selecionados foram qualificados da seguinte forma: o ID do trabalho, o autor, os pontos atribuídos pelos critérios de qualidade e o total de pontos. Observa-se que os artigos selecionados foram validados, e a lista de artigos não sofreu alterações. Os middlewares selecionados foram publicados em periódicos com impacto relevante. Dois middlewares foram aplicados na agricultura. Além disso, apenas três trabalhos disponibilizaram o código em repositório público. Nenhum trabalho aplicado na agricultura era *open source*.

3.9.1 Ameaças a Validação do Trabalho

O objetivo desta pesquisa foi reunir os middlewares mais recentes para a internet das coisas que atendessem aos critérios de seleção. Executou-se a pesquisa o mais amplamente

possível, sem perder o foco, para reduzir a lacuna entre o propósito geral de um *middleware* e um *middleware* para a internet das coisas. No entanto, algumas questões relevantes podem ameaçar os resultados deste trabalho, conforme listado a seguir:

- Foram coletados apenas artigos indexados nas bases de dados listadas. Portanto, se o artigo de um determinado periódico não foi indexado nessas bases de dados, esse artigo não foi incluído. Para mitigar essa ameaça, as bases de trabalho selecionadas indexam os periódicos e conferências mais importantes relacionados à Ciência da Computação e IoT.
- *Middleware* que não possui trabalho científico publicado não foi incluído. Embora existam alguns middlewares disponíveis no mercado, entende-se que não se pode ter acesso a todas as informações necessárias sobre esses middlewares. Além disso, seguiu-se um protocolo de pesquisa bem definido para abranger os trabalhos completos publicados para descrever o cenário atual de pesquisa sobre *middleware* para a internet das coisas.
- Artigos curtos não foram incluídos nos resultados. A procura de artigos curtos exige ainda mais recursos (tempo); e, as informações em artigos curtos podem não ser tão confiáveis ou completas, especialmente esses, que geralmente, apresentam resultados preliminares.
- Artigos publicados antes de 2016 não foram incluídos nos resultados. No entanto, os trabalhos relacionados (Seção 3.11) já tratam dos middlewares mais antigos. Este trabalho busca atualizar o estado atual sobre *middleware* para IoT e encontrar lacunas que ainda estão abertas sobre *middleware* para a internet das coisas.

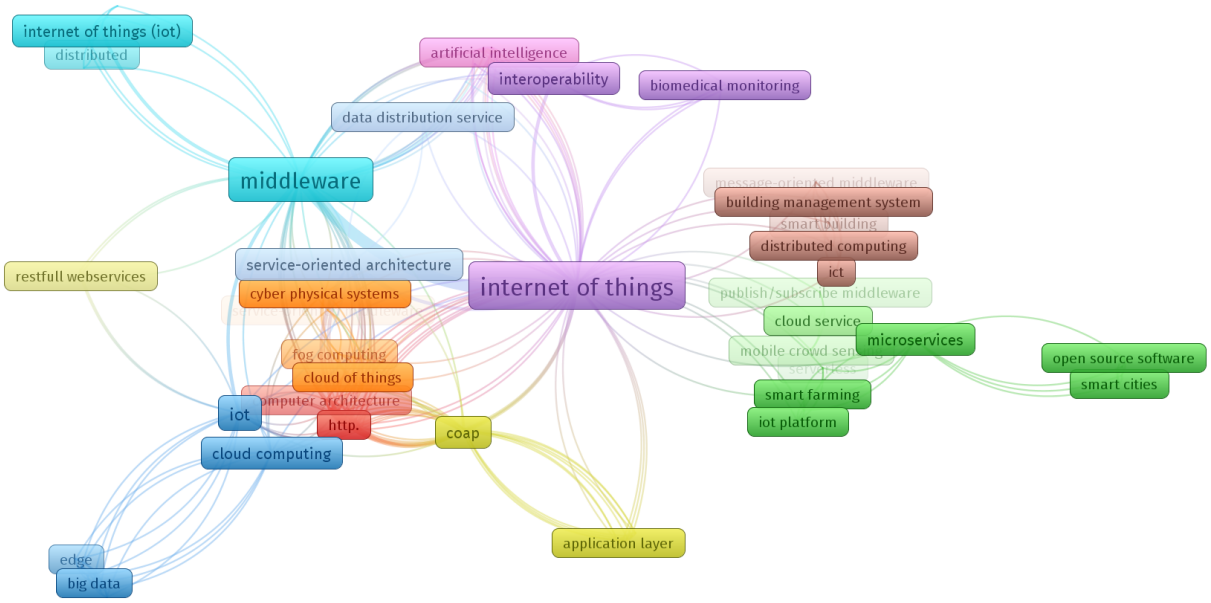
3.10 Resultados e Discussões

3.10.1 QP - Quais middlewares foram construídos e utilizados em aplicações para IoT?

Houve um total de 22 (vinte e dois) artigos selecionados que apresentaram *middleware* para a Internet das Coisas. Os artigos foram identificados pelos autores (Tabela 4). Além disso, os trabalhos foram avaliados com os critérios de qualidade, conforme definido na Seção 3.4. Na Tabela 5, mostra-se um resumo dos middlewares selecionados. A descrição está organizada da seguinte forma: o nome do *middleware*, o problema ou área que o *middleware* se propõe a resolver, a arquitetura do *middleware*, os requisitos, os protocolos de comunicação, a camada física utilizada, a disponibilidade e aplicação do *middleware*. Essa revisão teve como objetivo listar os middlewares mais atuais.

As palavras-chave de cada artigo foram inseridas no VOSViewer para gerar o gráfico de visualização da *network* (Figura 11). Foi possível identificar uma relação entre IoT e *middleware*. Acredita-se que essa relação seja forte devido à necessidade de uma infraestrutura que facilite a construção de aplicações IoT mais complexas. Os middlewares propostos representam essa infraestrutura e são usados na construção de aplicações IoT, principalmente para fins de validação. Observa-se que essas aplicações usadas para validação são responsáveis por outras correlações mais fracas, como IoT e *smart farming*, *smart farming* e *biomedical monitoring*.

Figura 11 – *Network* de palavras-chave dos trabalhos selecionados



Fonte: autoria própria.

Embora alguns autores mencionem que estão utilizando algumas tecnologias, não foi possível detectar esse respectivo uso em suas aplicações. Desse modo, as palavras indicadas com o símbolo "*" para as colunas "protocolos " e "camada física" foram marcadas (Tabela 5). Na coluna "disponibilidade", o *middleware* indica que eles eram de código aberto, mas o código do *middleware* não está totalmente disponível para acesso. Quando não foi possível detectar as informações em nenhuma coluna dos respectivos periódicos, foi atribuído um hífen.

Alguns trabalhos não foram aplicados em ambientes reais. Outra lacuna está relacionada ao fato do requisito "interoperabilidade" uma vez que muitos autores mencionaram que desenvolveram *middleware* com base nesse requisito. Além disso, pode-se observar que a camada física não foi especificada nem validada em aplicações reais.

Os trabalhos selecionados são discutidos a seguir:

Tabela 5 – Sumarização dos *middlewares* selecionados

ID	Nome	Problema	Arquitetura	Requisitos	Protocolos	Camada Física	Disponibilidade	Aplicação
1	In.iot	Smart Energy	Nuvem	Interoperabilidade, Escalabilidade , Confiabilidade	MQTT, CoAP, HTTP	-	Open Source	Monitoramento Inteligente de Energia
2	SEnviro Connect	Smart Farming	Borda	Interoperabilidade, Escalabilidade , Confiabilidade	HTTP, MQTT	2G/3G	-	Monitoramento no Cultivo de Uvas
3	-	Gerenciamento de Dispositivos	Nuvem	Interoperabilidade, Confiabilidade	HTTP/REST, MQTT, COAP	Wi-Fi, BLE*, ZigBee*, Z-Wave*, M-Bus*, CANBus*	-	Monitoramento do Cultivo de Flores
4	CUPUS	Sensoriamento de Multidão Móvel	Nuvem	Flexibilidade	HTTP/GCM	BLE	-	Monitoramento da Qualidade do Ar Urbano
5	-	Estações de Tratamento de Esgoto	Borda	Interoperabilidade	MQTT, HTTP	Modbus, RS485, RS422, RS232	-	Projeto de Monitoramento de Estação de Tratamento de Esgoto Rural
6	InterSCity	Smart City	Nuvem	Interoperabilidade, Flexibilidade, Escalabilidade, Confiabilidade	HTTP/REST, MQTT	-	Open Source	Simulação de Vagas de Estacionamento
7	MiddleBrigde	Protocolos de Aplicação Diferentes	Borda	Interoperabilidade	MQTT, CoAP, DDS, WebSocket, HTTP/REST	-	Open Source	Simulação Enviando os Dados Transformados
8	-	e-Health	Borda	Interoperabilidade, Confiabilidade	-	Wi-Fi, Bluetooth	-	Monitoramento de Dados Médicos
9	CoIoT	Smart Lab	Borda	Gerenciamento de Contexto, Interoperabilidade	HTTP/REST	Wi-Fi	-	Monitoramento de Testes de Sementes
10	M-Hub/CDDL	Qualidade de Contexto	Nuvem	Gerenciamento de Contexto	MQTT	Bluetooth	-	Monitoramento Remoto de Pacientes
11	SWaMM	Smart Watering	Nuvem	Interoperabilidade	HTTP/REST, MQTT	Wi-Fi, M-Bus, LoRa	-	Monitoramento Inteligente da Distribuição de Água
12	Thinger.io	Integração de Dados	Nuvem	Interoperabilidade, Conectividade	HTTP/REST, MQTT*, CoAP*	Wi-Fi, SigFox*, LoRa*	Pago, Open Source*	Monitoramento Meteorológico
13	Cilia	Smart Manufacturing	Nuvem	Flexibilidade	HTTP/GCM	ModBus	-	Monitoramento de Estações de Bombeamento
14	DeXMS	Protocolos de Aplicação Diferentes	Borda	Interoperabilidade	MQTT, CoAP, WebSocket, HTTP/REST	-	-	Simulação com Dispositivos Heterogêneos
15	MSM	Gerenciamento de Dispositivos	Nuvem	Interoperabilidade, Privacidade, Segurança	MQTT, HTTP, CoAP, WebSocket, DDS	-	-	Simulação
16	SmartCityAware	Smart City	Nuvem	Flexibilidade	HTTP/REST	Wi-Fi	-	Simulação
17	MinT	Gerenciamento de Dispositivos	Borda	Interoperabilidade, Confiabilidade	MQTT, CoAP, HTTP/REST	Wi-Fi, Bluetooth, BLE, ZigBee	Open Source*	Simulação
18	-	Smart Building	Borda	Interoperabilidade	Socket	Wi-Fi, Ethernet	-	Simulação
19	eVATAR+	Smart Home	Borda	Interoperabilidade	HTTP/REST	-	-	Simulação
20	MSOAH-IoT	Smart Home	Borda	Interoperabilidade	HTTP/REST	Wi-Fi, Bluetooth, 6LowPAN*, ZigBee*	-	Simulação
21	-	Smart Environment	Nuvem	Flexibilidade	HTTP/REST, AMQP	Wi-Fi	-	Simulação de Vigilância por Vídeo
22	RDS	Gerenciamento de Dispositivos	Borda	Interoperabilidade, Escalabilidade	DDS	-	-	Simulação

O In.Iot foi apresentado por Cruz et al. (2021) e é uma plataforma de *middleware* para IoT que apresenta uma abordagem para dispositivos IoT para conectar e compartilhar dados. In.Iot foi definido como uma arquitetura de microsserviços, mas uma versão monolítica foi disponibilizada. Sua primeira implementação foi escrita em Java. A solução usou restrições de tópico para incrementar recursos de segurança para um *broker* MQTT. Apesar de reduzir as chances da distribuição de dados, essa modificação não evita que os dispositivos possam receber dados não relacionados. Os autores afirmam que os protocolos MQTT, CoAP e HTTP são suportados. No entanto, os autores não disponibilizaram mecanismos bem definidos para comunicação com o *middleware* por meio desses protocolos. Além disso, os outros protocolos não são incluídos na abordagem de restrições. Por fim, como forma de validação, In.iot foi usado em um aplicação de monitoramento de energia inteligente.

O SEnviro Connect foi apresentado por Trilles, González-Pérez e Huerta (2020) e é uma plataforma de nuvem IoT para abranger o escopo físico e de nuvem para gerenciar dispositivos IoT e analisar os dados produzidos. A solução usou uma arquitetura de microsserviços dividida em quatro partes: persistência, análise, corretores e API. Nas camadas foram utilizados alguns serviços de código aberto e serviços privados. A plataforma usou o *broker* RabbitMQ para comunicação usando MQTT. Além disso, usou Micro Mu para análises, InfluxDB para persistência, Kapacitor para alertas e Firebase para autenticação, armazenamento e mensagens. SEnviro Connect foi usado em uma aplicação de agricultura inteligente usando dispositivos IoT chamados SEnviro Nodes. O *middleware* foi construído para uma aplicação específica. Dessa forma, ele não promove a comunicação por diferentes protocolos de comunicação. Além disso, não disponibiliza mecanismos de controle para as aplicações.

Zhao et al. (2018) apresentou um *middleware* orientado a serviços usando HTTP/REST, MQTT e CoAP compatível com oneM2M. As técnicas *Variable* e *McSugar* foram criadas para usar recursos de dispositivos heterogêneos e realizar interações uniformes entre as aplicações. Essas técnicas também fornecem a base para pesquisas na IoT, como raciocínio semântico e *Fog Computing*. O *middleware* foi projetado para facilitar a criação de novas aplicações e testes. No entanto, não foi possível identificar mecanismos de comunicação para facilitar essa construção. Além disso, não foi possível detectar a utilização de todos os protocolos mencionados.

No trabalho apresentado por Antonić et al. (2016), os autores apresentaram o *middleware* CUPUS. O *middleware* foi desenvolvido para adquirir dados de sensores de dispositivos móveis de forma flexível, com baixo consumo de energia e processamento em tempo real de *streams* de *Big Data*. A arquitetura consiste em dois componentes: o *broker* da nuvem e o *broker* móvel. O *middleware* usa TCP/IP e *Google Cloud Messaging* (GCM) para enviar dados ao agente de nuvem. O *broker* móvel se comunica com dispositivos

via Bluetooth. O *middleware* é usado em um aplicativo para verificar a qualidade do ar urbano. Observou-se que não foi definido nenhum aspecto para facilitar a construção de uma outra aplicação. O *middleware* foi desenvolvido para uma aplicação bem específica.

Xie et al. (2021) apresentou um *middleware* IoT multicamadas para fornecer acesso flexível ao dispositivo IoT. A solução proposta foi baseada em um gráfico de conhecimento IoT, que visa reduzir a heterogeneidade entre os dispositivos IoT. O gráfico de conhecimento de IoT consiste em três partes: uma ontologia de base da IoT; uma ontologia de domínio IoT; e uma instância IoT. O *middleware* foi dividido em dois *middleware* L1 e L2. O *middleware* L1 IoT armazena apenas o nó raiz e os atributos estáticos de um dispositivo, enquanto os dados dinâmicos do dispositivo são armazenados no *middleware* L2 IoT. Suporta MQTT e HTTP como protocolos da camada de aplicação para comunicação entre estruturas L1 e L2. O *middleware* foi utilizado no monitoramento remoto de estação de tratamento de esgoto rural.

O InterSCity foi apresentado por Esposte et al. (2019) e é uma plataforma de *middleware* desenvolvida para dar suporte aos desafios relacionados à infraestrutura de software para *Smart Cities*, segundo o autor, com foco em redes de alto desempenho e computação distribuída, engenharia de software, análise de dados e modelagem matemática. A plataforma oferece e disponibiliza os serviços baseados na Web, como gerenciamento de serviços, recursos IoT heterogêneos e suporta a descoberta de recursos da cidade com base em dados de contexto, para armazenar e processar dados e comandos de ação intermediários. O *middleware* usa os protocolos HTTP/REST e MQTT. Essa plataforma possui uma finalidade bem definida, devido a isso, não oferece formas para construção de aplicações em ambientes diferentes. Sendo assim, também não oferece a comunicação por diferentes protocolos.

O MiddleBridge foi apresentado por Cruz et al. (2019). Esse é um *middleware* que oferece uma solução para mediar a comunicação entre dispositivos que não adotam o mesmo protocolo. Ele funciona como uma ponte. Segundo o autor, o *middleware* transforma mensagens MQTT, CoAP, DDS e WebSocket para HTTP/REST e encaminha para os interessados. No entanto, não foi possível identificar a utilização de todos os protocolos. Além disso, o *middleware* não oferece uma comunicação de entrada e saída por diferentes protocolos, o que compromete a capacidade de interoperar.

Georgi, Corvol e Jeannes (2018) apresentaram o *middleware* na forma de uma biblioteca de *software*, capaz de se comunicar com um conjunto de sensores para acessar quaisquer dados. Esse *middleware* foi designado para ser usado por desenvolvedores terceirizados, aplicativos de saúde ou provedores para recuperar dados do paciente, permitindo ao usuário garantir a interoperabilidade com um número máximo de sensores aplicados à saúde com o mínimo de esforço. O *middleware* foi desenvolvido para Android para se comunicar com os dispositivos usando camadas físicas Wi-Fi e Bluetooth.

O CoIoT foi apresentado por Souza et al. (2019). Esse *middleware* foi desenvolvido para auxiliar no monitoramento de testes de sementes. O *middleware* fornece suporte para aplicativos e mecanismos onipresentes para o gerenciamento de dispositivos IoT. Foi desenvolvido baseado em outro *middleware* para agregar ao tratamento de dispositivos atuadores e sensores, além de utilizar os recursos de infraestrutura do ambiente ubíquo. O *middleware* usa o protocolo HTTP/REST. Todavia, não foi possível detectar formas padronizadas para auxiliar na construção de aplicações.

O M-Hub/CDDL foi apresentado por Gomes et al. (2017). Esse *middleware* foi elaborado para gerenciar dados de contexto (aquisição, processamento e distribuição) abstraindo a complexidade do contexto e fornecendo o serviço apropriado. É dividido em duas camadas principais: MHUB e CDDL e usa o protocolo de comunicação MQTT. É aplicado em um estudo de caso envolvendo o desenvolvimento de um aplicativo móvel para monitoramento remoto de pacientes. Percebe-se que o *middleware* foi desenvolvido para uma aplicação bem específica, reduzindo a generalização para construção de aplicações para cenários distintos.

O SWaMM foi apresentado por Alvisi et al. (2019) e é um *middleware* sem fio interoperável, baseado no paradigma *Edge Computing* para pré-processamento e ganho de eficiência no tempo. Seu foco principal é interagir com diversos tipos de medidores inteligentes de consumo de água, operando com alguns protocolos de comunicação. A plataforma usa o protocolo HTTP/REST na API e MQTT entre o *gateway* e o gerenciador de dados. O *middleware* foi desenvolvido para uma aplicação específica. Assim, não emprega outros protocolos de comunicação ou disponibiliza ferramentas que facilitem a construção de aplicações para IoT.

O Thinger.io foi apresentado por Alvaro, Miguel e Molina (2019). O *middleware* fornece um serviço de nuvem para conectar dispositivos à Internet para integração de dados. O *middleware* é exposto no trabalho como código aberto, mas no repositório o código-fonte não está disponível e apenas algumas APIs para acesso à nuvem estão disponíveis. Os autores afirmam que é independente de *hardware*, no entanto, não fornece acesso por *hardware* diferente (LoRa, por exemplo). Utiliza os protocolos HTTP/REST, MQTT e CoAP. O *middleware* é validado em uma aplicação meteorológica usando Wi-Fi e Arduino. Não foi possível identificar a utilização de todos os protocolos.

O Cilia foi apresentado por Lalanda, Morand e Chollet (2017) e foi projetado para fábricas inteligentes. A arquitetura é baseada em serviços. Proporciona dinâmica e autonomia de recursos, permitindo grande flexibilidade em tempo de execução com mínima intervenção humana. Foi aplicado em um estudo de caso real que envolveu a supervisão e manutenção de estações elevatórias. O *middleware* não foi desenvolvido para construir aplicações para outros ambientes. Dessa forma, ele não promove a comunicação por diferentes protocolos para integrar dispositivos distintos na IoT.

O DeXMS foi apresentado por Bouloukakis et al. (2019). Trata-se de um *middleware* construído para servir de ponte abstraindo a comunicação entre protocolos, que pode ser uma ponte direta de um protocolo para outro ou utilizando dois mediadores, para passar de um protocolo inicial a um protocolo final, passando por outro protocolo entre eles. Ele usa os protocolos de comunicação: MQTT, CoAP, HTTP/REST e WebSocket. Para isso, DeXMS cria um protocolo, denominado: DeX, que encapsula cada protocolo mencionado. No entanto, esse *middleware* não aborda todos os desafios de um *middleware* multiprotocolo para IoT (Seção 1.1.1). Não foi possível identificar mecanismos de controle para as aplicações ou formas de gerenciamento de recursos ou de dados. Além disso, protocolos como MQTT ou CoAP foram definidos para cenários distintos, encapsular esses protocolos por um outro protocolo não é a melhor solução, pois descaracteriza suas definições e possivelmente aumenta o tempo de processamento e tamanho de pacote.

O MSM foi apresentado por Benayache et al. (2019) e é um *middleware* inspirado na arquitetura de Redes Neurais Artificiais e microsserviços, obtendo uma interação dinâmica entre serviços e suportando alguns protocolos de comunicação: MQTT, HTTP, CoAP, WebSocket, DDS. MSM foi escrito em linguagem de programação Python. A solução foi comparada com outras soluções de *middleware* em termos de métricas qualitativas e quantitativas: inteligência, registro de serviço, descoberta de serviço e latência. Não foi possível identificar a utilização de todos os protocolos. Além disso, não foi possível identificar mecanismos que facilitem a construção de aplicação na IoT ou formas de controle para as aplicações.

O SmartCityAware foi apresentado por Mohamed et al. (2017). Essa é plataforma de *middleware* orientada a serviços projetada especificamente para usar CoT e *Fog Computing* a fim de apoiar o desenvolvimento e operação de aplicações de cidade inteligente. O principal objetivo do SmartCityWare é fornecer um ambiente virtual usado para desenvolver e implantar aplicativos de cidades inteligentes. SmartCityWare consiste em um conjunto de serviços e um ambiente de tempo de execução multi-agente. O *middleware* usa o protocolo HTTP/REST.

O MinT foi apresentado por Jeon e Jung (2017). Esse é um *middleware* distribuído focado na interação cooperativa que fornece os recursos básicos para operar um dispositivo IoT distribuído. Tem como objetivo a eficiência energética e o atendimento rápido às solicitações, permitindo uma interação eficiente entre os dispositivos IoT, levando em consideração o consumo de energia, os atrasos da rede e a confiabilidade. Ele usa os protocolos MQTT e CoAP.

Lilis e Kayal (2018) apresentaram um *middleware* sem *brokers* para servir de interface e interconectar o digital e o físico para sistemas em edifícios inteligentes usando a biblioteca de mensagens distribuídas ZeroMQ[7]. É orientado a mensagens entre *pu-*

[7]<https://zeromq.org/>

blish/subscribe com filas e descoberta de serviço. Os nós se comunicam de maneira P2P em redes: Wi-Fi e Ethernet. A solução foi implantada em *hardware* embarcado para validação das funcionalidades do *middleware* e avaliação de consumo de energia. O *middleware* foi desenvolvido para uma aplicação específica. Dessa forma, não promove a comunicação por diferentes protocolos, como também não facilita a construção de uma aplicação para um ambiente diferente.

O eVATAR+ foi apresentado por Dipsis e Stathis (2020) e é um *middleware* que busca incorporar recursos de Inteligência Artificial (IA) em redes de sensores-atuadores existentes ou infraestruturas IoT tornando os serviços de configuração oferecidos mais inteligentes. eVATAR+ usa tecnologias de servidor Java aprimoradas pela funcionalidade de mediador, fornecendo interoperabilidade e suporte de manutenção. A solução é um software de arquitetura monolítica que foi dividido em três camadas: camada de controlador, camada de negócios e camada de persistência. Atualmente oferece a API HTTP/REST como um protocolo de camada de aplicação para se comunicar com dispositivos que oferecem essa mesma interface de comunicação. eVATAR+ foi usado em um cenário de casa inteligente com o *Google Nest* e *Jade Agent*.

O MSOAH-IoT foi apresentado por Mesmoudi et al. (2018). O *middleware* é baseado em uma arquitetura orientada a serviços para lidar com os problemas de heterogeneidade. O *middleware* foi desenvolvido com a tecnologia Java e usa o protocolo HTTP/REST para se comunicar com os dispositivos. Percebe-se uma capacidade reduzida para interoperar com dispositivos que utilizam protocolos de comunicação diferentes, devido ao fato de não oferecer comunicação por outros protocolos. A solução foi validada em três cenários simulados: no primeiro, o *middleware* foi instalado em um Raspberry Pi como *gateway* para coleta de dados de dispositivos conectados via Wi-Fi; no segundo caso, o *middleware* instalado manipula dados provenientes de dispositivos conectados por Wi-Fi e Bluetooth; e, no terceiro foi utilizado um computador pessoal funcionando como *gateway*.

Merlino et al. (2019) apresentaram uma plataforma de *middleware* que estende a funcionalidade atual das plataformas existentes OpenStack e Stack4Things, por meio da qual contêineres de recursos nos níveis *edge*, *fog* e *cloud* podem ser descobertos, combinados e provisionados para usuários finais e aplicações para facilitar e orquestrar os processos de *download*. O *middleware* usa os protocolos de comunicação HTTP/REST e AMQP. A solução foi avaliada com um estudo de caso em um sistema de vigilância inteligente. Percebeu-se que o *middleware* possui uma finalidade bem definida. Diante disso, não foi possível identificar mecanismos que facilitam a construção de aplicações para IoT com objetivos diferentes.

O RDS foi apresentado por Shokrollahi e Shams (2017) e é um *middleware* baseado em uma abordagem derivada de uma arquitetura orientada a serviços, chamada *Rich Services*. RDS é uma abordagem que explora o *middleware* DDS para *publish/subscribe*,

centralizar dados, funcionar em tempo real e comunicação fracamente acoplada entre dispositivos para permitir uma integração escalável e dinâmica de dispositivos heterogêneos na IoT. No entanto, esse *middleware* possui uma capacidade reduzida para interoperar, devido ao fato de não utilizar outros protocolos de comunicação com mecanismos que facilitem essa comunicação.

3.10.1.1 Ferramenta Relacionada

Node-RED é uma ferramenta de programação baseada em fluxo (*Flow-based Programming*), originalmente desenvolvida por Nick O'Leary e Dave Conway-Jones do grupo de Serviços de Tecnologia Emergentes da IBM (JSFOUNDATION, 2021). Programação baseada em fluxo é uma maneira de descrever o comportamento de um aplicativo como uma rede de caixas pretas, ou "*nodes*". Cada *node* tem uma finalidade; ele recebe alguns dados, faz algo com esses dados e então os passa adiante. Dessa forma, essa ferramenta liga o fluxo de dados entre pontos de uma aplicação. Diferentemente dos middlewares, o Node-RED consiste em um *runtime* baseado em Node.js para o qual oferece uma interface no navegador da Web para acessar o editor de fluxo.

Um dos pontos positivos dessa abordagem está no fato de usuários poderem criar aplicações muito facilmente. No entanto, o Node-RED não oferece mecanismos de comunicação e controle para as aplicações ou gerenciamento de dados e recursos. As configurações dos dispositivos ou aplicações são de responsabilidade dos usuários. Assim, mesmo oferecendo facilidades na integração dos fluxos de dados, toda configuração do lado cliente (dispositivos e aplicações) deve ser realizada manualmente – salvo em casos em que o Node-RED consegue ler GPIOs dos dispositivos utilizando o protocolo Firmata, mas depende que esses dispositivos estejam conectados ao *Host* por uma USB. Entretanto, o Node-RED pode ser incorporado ou adaptado por um *middleware* para IoT para oferecer sua interface riquíssima na construção e ligação de fluxos.

3.10.2 QS1 - Quais arquiteturas foram utilizadas para construir esses middlewares?

Duas arquiteturas principais foram identificadas nas arquiteturas dos middlewares. Na Tabela 6, apresenta-se a arquitetura e o *middleware* que aborda essa arquitetura principal. Além disso, cada arquitetura é apresentada e discutida a seguir.

Baseadas na Nuvem

A maioria dos middlewares são baseados em nuvem, oferecendo uma *Platform as a Service* (PaaS) ou *Software as a Service* (SaaS). Essas soluções têm alguns recursos comuns a esse respeito. Assim, as plataformas oferecem mecanismos que não podem ser

Tabela 6 – Arquiteturas abordadas por middlewares para IoT

Arquitetura	***Middleware***
Baseadas na Nuvem	In.iot, CUPUS, InterSCity, MSM, Zhao *et al.*, M-Hub/CDDL, Thinger.io, SWaMM, Cilia, SmartCityAware, Merlino *et al.*
Baseadas na Borda	Xie *et al.*, eVATAR+, SEnviro Connect, MiddleBrigde, DeXMS, Lilis *et al.*, Georgi *et al.*, CoIoT, MinT, MSOAH-IoT, RDS

oferecidos nos dispositivos devido à natureza restrita dos recursos que estão presentes na IoT. Essas limitações impedem a oferta de serviços completos. Dessa forma, o *middleware* aborda essas dificuldades fornecendo provisionamento de serviço, capacidade de armazenamento, gerenciamento remoto de recursos, descoberta de serviço, processamento de dados e visualização. Os middlewares baseados na nuvem buscam principalmente oferecer interoperabilidade, escalabilidade e flexibilidade. Pode-se encontrar arquiteturas monolíticas, SOA ou arquitetura de microsserviços. No entanto, a arquitetura de microsserviços ainda não é predominante entre os middlewares IoT.

Baseadas na Borda

O *middleware* baseado na borda está estritamente interessado na interoperabilidade entre dispositivos na IoT. Por isso, buscam oferecer soluções de gestão e comunicação entre dispositivos com o mínimo da nuvem. Assim, alguns deles buscam apenas realizar uma tradução entre protocolos de comunicação para conseguir uma interação entre os dispositivos. Essa abordagem facilita a interoperabilidade dos dispositivos. Além disso, a confiabilidade é aprimorada pelo fato de estar em comunicação local, evitando problemas relacionados ao tráfego de dados com a nuvem. De fato, alguns middlewares são construídos para áreas específicas, como a indústria ou casas inteligentes, provendo interfaces para usuário e funcionalidades específicas do ambiente. Todavia, uma arquitetura baseada em microsserviços poderia ser uma adoção inteligente para middlewares de propósito geral. Assim, uma abordagem híbrida poderia ser facilitada com esse tipo de arquitetura. Dessa forma, dividir os componentes entre a borda e a nuvem é um caminho interessante.

3.10.3 QS2 - Quais requisitos foram atendidos por esses middlewares?

Sete requisitos principais abordados pelos middlewares foram citados. Todos os requisitos e quais middlewares estão tratando esses requisitos são exibidos na Tabela 7. Além disso, cada requisito é apresentado e discutido a seguir.

Interoperabilidade

Esse requisito é uma característica que possibilita a conexão, os serviços e a troca de informações com diversos dispositivos IoT utilizando e disponibilizando comunicação com

Tabela 7 – Requisitos abordados por middlewares para IoT

Requisito	**Middleware**
Interoperabilidade	In.iot, Xie *et al.*, SEnviro Connect, eVATAR+, Thinger.io, MSOAH-IoT, CoIoT, MiddleBrigde, DeXMS, M-Hub/CDDL, InterSCity, MinT, MSM, SWaMM, Georgi *et al.*, RDS
Flexibilidade	CUPUS, InterSCity, Merlino *et al.*, Cilia, SmartCityAware
Escalabilidade	In.iot, SEnviro Connect, InterSCity, RDS
Confiabilidade	In.iot, SEnviro Connect, MinT, InterSCity, Georgi *et al.*, Zhao *et al.*
Conectividade	Thinger.io
Gerenciamento de Contexto	CoIoT, M-Hub/CDDL
Segurança e Privacidade	MSM

diversos protocolos de comunicação. Ressalta-se que quase todos os trabalhos buscaram disponibilizar mecanismos para atender a essa exigência. Alguns middlewares usam mais de um protocolo de comunicação na camada de aplicação e também funcionam em diferentes tipos na camada de rede. Porém, não foi possível encontrar, nos estudos primários selecionados, um *middleware* multiprotocolo em que os autores descrevam como diferentes dispositivos, que utilizam protocolos incompatíveis, podem enviar, receber dados e controlar ações.

Flexibilidade

Esse requisito é inserido como forma de criar mecanismos que tornem o *middleware* adaptável e extensível. Ele foi introduzido para resolver um problema relacionado à reutilização de *middleware* em diferentes ambientes. Assim, a flexibilidade foi utilizada na definição da arquitetura a ser utilizada por esses middlewares, a fim de tornar as arquiteturas adaptáveis. Pode-se observar que os middlewares possuem uma arquitetura que funciona na nuvem e fornece mecanismos de comunicação com a infraestrutura. Metade dos middlewares que mencionam o requisito flexibilidade foram aplicados em problemas relacionados a cidades inteligentes.

Escalabilidade

Na definição proposta por Abbott e Fisher (2015) existem três tipos de escalabilidade que podem ser exemplificadas com o cubo de escala: i) a escala do eixo X que escala horizontalmente distribuindo a carga total por um certo número de nós; ii) a escala do eixo Y refere-se à divisão e distribuição em serviços, conhecida como decomposição funcional que se reflete em arquiteturas orientadas a serviços e microsserviços; e iii) a escala do eixo Z refere-se ao particionamento de dados, que é uma forma de distribuir dados entre

muitos nós ou blocos de dados para melhorar o desempenho. Portanto, o *middleware* que usa esse requisito está diretamente relacionado ao eixo Y. Ele busca compartilhar suas funcionalidades, mesmo se aplicar o *middleware* em diferentes ambientes.

Confiabilidade

A confiabilidade de um sistema está relacionada ao tempo médio entre as falhas. Além disso, este requisito também aborda a capacidade do sistema de se recuperar após uma falha e o tempo que leva para realizar essa recuperação (BAJPAI; GORTHI, 2012). O *middleware* que aborda esse requisito busca justificar o uso de ferramentas, bibliotecas e estruturas conhecidas e testadas para evitar falhas. Georgi, Corvol e Jeannes (2018) reforçaram a necessidade de dados confiáveis gerados por sensores em ambientes de *e-health*. Essa preocupação se deve ao risco de geração de dados imprecisos, ocasionando tomadas de decisão equivocadas.

Conectividade

Na IoT, diferentes tipos dispositivos são conectados por diferentes redes físicas. A conectividade está relacionada a esse fato. O *middleware* deve fornecer interfaces de comunicação, juntamente com interoperabilidade para gerenciar dispositivos de grupo no ecossistema em diferentes redes (Wi-Fi, Bluetooth, RFID, LoRa, ZigBee). Assim, alguns middlewares citaram e empregaram essas outras formas de conexão, além do Wi-Fi.

Gerenciamento de Contexto

Gerenciamento de contexto é gerenciar o conjunto de informações e inferir o estado atual de uma entidade para disponibilizar serviços aos usuários. Os sistemas que usam esta informação são considerados sensíveis ao contexto (ABOWD et al., 1999). Assim, o reconhecimento de contexto permite o armazenamento de informações de contexto vinculadas aos dados do dispositivo, permitindo uma interpretação fácil e significativa (PERERA et al., 2014).

Segurança e Privacidade

A segurança é um requisito importante em qualquer sistema. Isso traz grande responsabilidade para um *middleware* na IoT, o qual está em constante comunicação entre dispositivos e aplicações, principalmente quando o sistema utiliza dados sensíveis emitidos por domínios militares, de saúde ou industriais. Assim, o *middleware* precisa fornecer mecanismos de segurança para evitar possíveis ataques a dispositivos e dados. O *middleware* MSM (BENAYACHE et al., 2019) inclui uma camada de segurança que monitora todas as outras camadas da arquitetura. Não foi possível identificar mecanismos de segurança nos demais. Assim, percebe-se que ainda há necessidade de mais trabalhos

e possibilidade de evolução para proteger o *middleware* IoT, dispositivos e informações privadas.

3.10.4 QS3 - Quais protocolos de comunicação foram utilizados por esses middlewares?

Seis protocolos de comunicação principais foram utilizados. Todos os protocolos de comunicação e os middlewares que abordam esses protocolos são exibidos na Tabela 8. O protocolo HTTP foi usado por quase todos os middlewares por meio de abstrações diferentes: o REST ou *Google Cloud Messaging*, por exemplo. Além destes, os outros protocolos de comunicação são apresentados e discutidos a seguir.

Tabela 8 – Protocolos de comunicação utilizados por middlewares para IoT

Protocolo	**Middleware**
HTTP	In.iot, Xie *et al.*, SEnviro Connect, eVATAR+, CUPUS, Thinger.io, MSOAH-IoT, CoIoT, MiddleBrigde, Lilis *et al.*, DeXMS, InterSCity, MinT, MSM, SWaMM, Merlino *et al.*, Zhao *et al.*, Cilia, SmartCityAware
CoAP	In.iot, MiddleBrigde, DeXMS, MinT, MSM, Zhao *et al.*
MQTT	In.iot, Xie *et al.*, SEnviro Connect, Thinger.io, MiddleBrigde, DeXMS, M-Hub/CDDL, MinT, MSM, SWaMM, Zhao *et al.*
AMQP	Merlino *et al.*
DDS	MiddleBrigde, MSM, RDS
WebSocket	MiddleBrigde, DeXMS, MSM

- CoAP: é um protocolo de camada de aplicação feito pela IETF (SHELBY; HARTKE; BORMANN, 2014) para uso com nós restritos e redes restritas (baixa potência, com perdas). O protocolo é projetado para aplicações *Machine-to-Machine* (M2M), como *Smart Energy* e automação predial e utiliza o UDP para transporte. O CoAP fornece um modelo de interação de *resquest/response* entre terminais de aplicações, oferece suporte à descoberta integrada de serviços e recursos e inclui conceitos-chave da Web, como URIs e tipos de mídia da Internet. O CoAP foi projetado para interagir facilmente com HTTP para integração com a Web.

- MQTT: foi criado por Andy Stanford Clark da IBM e Arlen Nipper da Eurotech em 1999. O MQTT é um modelo de *publish/subscribe* para fornecer comunicações aprimoradas com flexibilidade. Assim, os dispositivos de publicação (*publishers*) geram os dados e os enviam ao *broker*. O *broker* encaminha esses dados às entidades interessadas. Os assinantes (*subscribers*) podem acessar esses dados por meio do *broker*. Além disso, o MQTT é um protocolo de mensagens leve, projetado para dispositivos restritos e redes de baixa largura de banda, alta latência ou não confiáveis. Assim, o protocolo é adequado para comunicação M2M. O MQTT funciona em cima do protocolo TCP.

- AMQP: foi criado por John O'Hara no JPMorgan Chase em 2003. AMQP é um protocolo de camada de aplicação com uma conexão *Peer-to-Peer* (P2P), mas amplamente usado de forma assíncrona com foco na troca de mensagens entre *publish* e *subscribe* por meio de um *broker* (OASIS, 2020). Os publicadores enviam as mensagens ao *broker* e o *broker* as organiza em filas para distribuição a seus assinantes. Ele fornece três tipos de garantias de entrega de mensagens: entrega no máximo uma vez, pelo menos uma vez e exatamente uma vez. Para isso, o protocolo usa TCP para trocar mensagens de forma confiável. Além disso, ele usa autenticação e/ou criptografia baseada em SASL e/ou TLS. Portanto, qualquer ferramenta que possa criar e interpretar mensagens de acordo com o formato de dados AMQP pode interoperar com qualquer outra ferramenta compatível, independentemente da linguagem de implementação.

- DDS: é um protocolo centrado em dados construído pelo *Object Management Group* (OMG). Ele integra os componentes do ecossistema. Além disso, as mensagens produzidas por essas entidades são enviadas por meio de tópicos para um local de armazenamento global gerenciado pelo DDS. Assim, segue o padrão *publish/subscribe*. As entidades interessadas nos dados podem se inscrever nos tópicos para receber os dados. O DDS trabalha com o UDP e/ou TCP na camada de transporte (OMG, 2020).

3.11 Trabalhos Relacionados

A IoT tornou-se uma área de pesquisa com possibilidade de desenvolvimento em outras áreas para a resolução de diversos problemas. Diante disso, esta pesquisa tem sido conduzida sobre *middleware* para auxiliar na construção de aplicações na IoT. Os tipos de middlewares podem ser diferentes, mas a grande maioria possui mecanismos para gerenciar dispositivos ou coletar dados. No entanto, os avanços tecnológicos nos produtos trazem novas oportunidades e desafios. Portanto, é essencial trabalhos que coletem novos resultados sobre middlewares, periodicamente. Nessa seção, a revisão foi relacionada com outros trabalhos que possuem propósito semelhante, a fim de explorar a literatura sobre *middleware* aplicado à IoT.

Os trabalhos Rathod e Chowdhary (2018), Razzaque et al. (2016), Ngu et al. (2017), Farahzadi et al. (2018) e Vikash, Mishra e Varma (2020) reúnem uma quantidade substancial de middlewares e explicam suas características. Os artigos abordam os requisitos de *middleware* para IoT, dos quais o requisito de interoperabilidade é o mais citado. Razzaque et al. (2016) forneceram uma visão geral mais abrangente, bem como uma visão de alto nível dos desafios de um *middleware* na IoT. Raza et al. (2021) apresenta um trabalho semelhante ao de Razzaque et al. (2016). Ambos os artigos enfocam requisitos não

funcionais. Observa-se que embora o artigo de Raza tenha sido publicado posteriormente, o autor não analisou e não incluiu o trabalho de Razzaque. Vikash, Mishra e Varma (2020) definiram a relação entre WSN e IoT, explicando os desafios de um *middleware* WSN na IoT. Ngu et al. (2017) abordou um exemplo de aplicação na coleta de dados de um ambiente de fácil conexão. Além disso, Farahzadi et al. (2018) abordou o uso de *middleware* na *Cloud of Things*.

Finalmente, os trabalhos relacionados não explicam, em profundidade, os protocolos da camada de aplicação usados por *middleware* para IoT. Além disso, os trabalhos relacionados não definem um protocolo de Revisão Sistemática da Literatura para pesquisar esses middlewares. Essa pesquisa está propondo um protocolo a ser replicado em novas pesquisas para atualizar o estado da arte sobre *middleware* para *Internet of Things*. Na Tabela 9, compara-se uma lista de pesquisas relacionadas, mostrando os principais aspectos que caracterizam cada trabalho.

Tabela 9 – Comparação entre trabalhos relacionados a RSL

Referência	Ano	Citações	*Middleware*	Arquitetura	Protocolos de Comunicação	Requisitos	Revisão Sistemática
Razzaque et al. (2016)	2016	430	✓	✓	✗	✓	✗
Ngu et al. (2017)	2017	152	✓	✓	✗	✗	✗
Farahzadi et al. (2018)	2018	52	✓	✓	✗	✓	✗
Rathod e Chowdhary (2018)	2018	1	✓	✓	✗	✓	✗
Vikash, Mishra e Varma (2020)	2020	1	✓	✓	✗	✓	✗
Raza et al. (2021)	2021	0	✓	✗	✗	✓	✗
our work	2021	-	✓	✓	✓	✓	✓

Esta pesquisa pretende ser um complemento às pesquisas já disponíveis na literatura. Dessa forma, foram reunidos os middlewares mais recentes que se relacionam e buscam solucionar problemas na área da IoT. Em complemento, foram identificados os requisitos que essas obras atribuem em sua definição de arquitetura. As arquiteturas foram analisadas com base nas definições, validação e resolução de problemas. Além disso, identifica-se os protocolos que foram usados no processo de validação dos middlewares.

3.12 Considerações Finais

Como contribuição, a RSL abordou os pontos mais importantes para *middleware* na IoT. Desse modo, a pesquisa detalhou os protocolos da camada de aplicação, as arquiteturas e os requisitos utilizados por middlewares na IoT. Foram analisados vinte e dois artigos sobre *middleware* para IoT. Alguns middlewares funcionavam na nuvem e outros na borda. Os requisitos para cada *middleware* foram identificados: interoperabilidade, escalabilidade, flexibilidade, gerenciamento de contexto, confiabilidade, segurança e privacidade. Além

disso, os protocolos de comunicação utilizados também foram identificados. O HTTP foi o protocolo mais utilizado pelos middlewares.

Além disso, percebeu-se uma oportunidade de pesquisa sobre os requisitos de segurança e privacidade para *middleware* IoT. Notou-se que os requisitos de segurança e privacidade foram os elementos mais negligenciados. As abordagens existentes tendem a adaptar a segurança e a privacidade tardiamente. A questão permanece em aberto no que diz respeito à inserção de segurança e privacidade em todas as fases do ecossistema durante o desenvolvimento do *middleware* para IoT. No entanto, cada abordagem apresentou pontos fortes e fracos. Nenhuma abordagem solucionará todos os problemas da IoT, portanto, há uma necessidade de uma abordagem híbrida. Os middlewares desenvolvidos nos últimos cinco anos adotaram a nuvem como parte integrante. As tendências recentes estão mudando para usar *Fog* e *Edge Computing* para pequenos processamentos e para responder a solicitações sensíveis à latência na IoT.

O requisito de usabilidade do *middleware* para IoT não foi abordado por nenhum *middleware* exposto. Assim, mecanismos que facilitem o uso de um *middleware* para IoT por pessoas com pouco ou nenhum conhecimento precisam ser definidos na composição do *middleware*, tornando-se assim um possível requisito a ser abordado para a evolução de um *middleware* para IoT mais acessível à comunidade.

A maioria dos trabalhos propuseram agregar interoperabilidade às suas soluções. No entanto, houve uma redução da capacidade de comunicação entre os dispositivos. Esses trabalhos não agregaram diversos protocolos da camada de aplicação para comunicação entre *middleware*, dispositivos e aplicações. Portanto, a capacidade de interoperar fica comprometida. Outra lacuna nessas propostas está na camada física, uma vez que os middlewares não trazem uma gama de opções de comunicação entre as diferentes redes. Alguns trabalhos apenas mencionam que estão utilizando tais redes, mas não foi possível identificar nenhum tipo de validação utilizando a tecnologia em uma aplicação real. Além disso, o método de validação de algumas das propostas foi a simulação. Em alguns estudos, entretanto, não foi possível identificar como essas simulações foram realizadas.

Por fim, existem características essenciais que um *middleware* para IoT deve atender. No entanto, os middlewares identificados não mostraram com clareza as formas de comunicação do *middleware* com os dispositivos e aplicações; ou, não definiram mecanismos padronizados para comunicação e adaptação das formas de comunicação. Além disso, existe a necessidade da utilização de uma arquitetura escalável que torne e possibilite a distribuição do *middleware* entre a *cloud*, *fog* e *edge* provendo uma maior capacidade de interoperar. Dessa forma, o *middleware* proposto neste trabalho (Capítulo 4) busca promover a interoperabilidade em alto nível para auxiliar na construção, autonomia e dinamismo para aplicações na IoT. Em complemento, o *middleware* reúne requisitos de escalabilidade, flexibilidade, aspectos de segurança e privacidade.

4 MORE4IoT - Requisitos, Arquitetura e Implementação

Neste capítulo, apresenta-se o *middleware* multiprotocolo baseado em microsserviços para habilitar aplicações na IoT. A primeira versão do MORE4IoT (***M**ultipr**O**tocol Middlew**RE for I**nternet **of T**hings*) foi projetada para atender as lacunas de pesquisa discutidas no capítulo anterior e os desafios de um *middleware* multiprotocolo para IoT. Com isso, ela cobre os principais recursos para oferecer suporte a diversas aplicações e comunicação com diferentes tipos de dispositivos encontrados na IoT. O MORE4IoT oferece serviços implantados localmente ou em nuvem para gerenciar recursos IoT heterogêneos, controle de atuações, armazenamento e recuperação de dados, gerenciamento e descoberta de serviços.

O capítulo está dividido da seguinte forma: na Seção 4.1, apresenta-se os requisitos funcionais e não funcionais utilizados para definição da arquitetura; na Seção 4.2 é apresentada uma definição sobre arquitetura baseada em microsserviços, uma comparação entre arquitetura baseada em microsserviços e arquitetura monolítica, e ainda, descreve-se as vantagens e desvantagens da arquitetura orientada a microsserviços; na Seção 4.3, detalha-se a arquitetura do MORE4IoT baseada em microsserviços, a distribuição das responsabilidades e as formas de comunicação; na Seção 4.4, apresenta-se os detalhes das funcionalidades de cada microsserviço; e por fim, na Seção 4.5, mostra-se as tecnologias utilizadas para construir o MORE4IoT e as ferramentas para facilitar a construção das aplicações usando o MORE4IoT.

4.1 Requisitos

São descritos um conjunto de requisitos para o MORE4IoT, baseados nas características essenciais discutidas anteriormente e nos desafios de um *middleware* multiprotocolo para IoT. Com isso, aborda-se as funcionalidades necessárias para facilitar a construção de aplicações que usam e conectam dispositivos com protocolos diferentes. Os requisitos de gerenciamento de recursos e de dados são requisitos fundamentais, detectados a partir dos resultados da RSL. A maioria dos middlewares abordaram esses requisitos funcionais. No entanto, não foi possível detectar mecanismos de controle para as aplicações, de tal forma que a aplicação possua total controle dos seus dispositivos. Além disso, outro requisito importante está relacionado com a capacidade de construção de aplicações, por isso é fundamental disponibilizar as ferramentas para auxiliar na construção das aplicações, reduzindo a complexidade no desenvolvimento, principalmente na comunicação por diferentes

protocolos. Diante disso, os requisitos funcionais citados estão diretamente relacionados com a capacidade de interoperar, atribuída como um requisito não funcional. Portanto, destaca-se que ao cumprir os requisitos funcionais, a arquitetura proposta também atende a esse requisito não funcional.

4.1.1 Requisitos Funcionais

- **Gerenciamento de Recursos** - o MORE4IoT deve integrar vários recursos heterogêneos que se comunicam por protocolos de comunicação distintos para permitir trocas de dados sobre os ambientes que estão inseridos e disponibilizar mecanismos para controlar os recursos físicos. Dessa forma, ele deve encapsular dispositivos e aplicações em estruturas únicas que abstraem as características individuais, como funcionalidades, representação dos dados produzidos e protocolos de comunicação utilizados.

- **Gerenciamento de Dados** - a integração de um grande número de dispositivos e fontes de dados, exige que o MORE4IoT gerencie estes dados, fornecendo serviços de armazenamento e processamento de dados históricos. Portanto, ele dispensa que as aplicações tenham que implementar uma infraestrutura necessária para essas tarefas.

- **Mecanismos de Controle** - o MORE4IoT deve fornecer mecanismos para intermediar e disponibilizar o controle dos dispositivos para as aplicações. Assim, ele deve fornecer uma forma abstrata para que as aplicações possam controlar os dispositivos utilizando um modelo único com a capacidade de inserção de comandos e designação dos dados. Dessa forma, o MORE4IoT deve traduzir essas especificações para controlar o tráfego de dados e comandos de controle.

- **Disponibilização das Funcionalidades** - o MORE4IoT deve fornecer meios padronizados de acesso aos recursos, aos dados gerenciados e aos mecanismos de controle por meio de abstrações bem definidas, oferecendo modelos que facilitam essa integração, seguindo os padrões definidos na Engenharia de *Software*.

- **Ferramentas para Desenvolvimento** - o MORE4IoT deve fornecer um conjunto de ferramentas de Engenharia de *Software* para facilitar o desenvolvimento das aplicações e utilização dos dispositivos, entre as quais estão: *Software Development Kit* (SDK), *Application Programming Interface* (API) e interface visual para auxiliar os usuários com as funcionalidades do *middleware* em forma de painel para mapeamento dos recursos e os mecanismos de controle.

4.2 Arquitetura Baseada em Microsserviços

Um dos objetivos dessa abordagem é planejar uma arquitetura com a possibilidade de adesão de novos requisitos sem comprometer a infraestrutura do MORE4IoT. Desse modo, adotar uma arquitetura de microsserviços é uma estratégia eficiente. Assim, é possível adicionar novas funcionalidades sem comprometer o que está definido reduzindo custos de manutenção e disponibilidade para uso.

Um estilo arquitetônico baseado em microsserviços auxilia na construção de um sistema complexo como um conjunto de componentes (microsserviços) independentes, distribuídos e com responsabilidades bem definidas, que colaboram utilizando protocolos de comunicação (FOWLER; LEWIS, 2014). O modelo de microsserviços surgiu dos esforços da indústria de *software* para construir sistemas distribuídos em larga escala reunindo as diretrizes da *Service-Oriented Architecture* (SOA) tradicional, orientada por *design* de domínio, entrega contínua, virtualização sob demanda, automação de infraestrutura e pequenas equipes autônomas (NEWMAN, 2015).

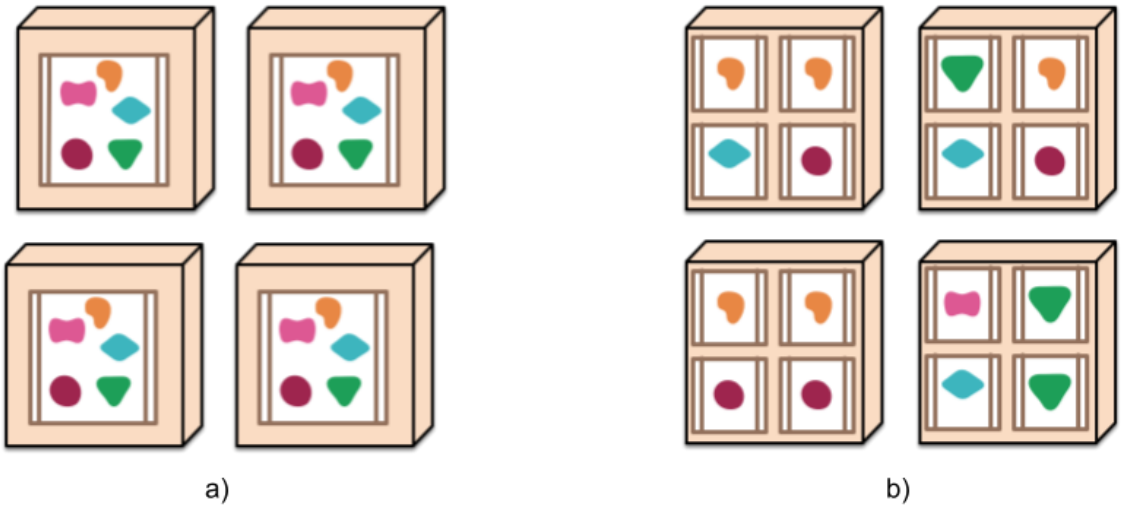
Cada microsserviço deve implementar um sistema altamente coeso, construído em volta dos recursos de negócios com limites explícitos (DRAGONI et al., 2017). O limite de um microsserviço é geralmente definido por sua API que promove a utilização por outros microsserviços com chamadas síncronas ou mensagens assíncronas, independentemente das tecnologias em uso. Como consequência, esse isolamento promove uma arquitetura de baixo acoplamento, permitindo uma implantação independente e o gerenciamento de versões, alavancando o uso das técnicas de *DevOps* (BALALAIE; HEYDARNOORI; JAMSHIDI, 2016), além de princípios ágeis, como entrega contínua e aceitação de mudanças (SCHWABER, 2012). Por outro lado, as principais desvantagens dessa abordagem estão no aumento da complexidade geral do sistema e no uso extensivo de chamadas remotas, que tendem a ser mais custosas do que as chamadas em memória local.

A implementação de uma quantidade limitada de funcionalidades torna a base de códigos dos microsserviços pequena, assim, facilitando a manutenção, testes isolados e limitando o escopo de alguma falha. Além disso, os microsserviços levam ao gerenciamento descentralizado de dados, permitindo que cada serviço gerencie seu banco de dados, o que é conhecido como abordagem de persistência poliglota. Com isso, diferentes tecnologias de Sistema de Gerenciamento de Banco de Dados (SGBD) podem ser utilizadas para diferentes tipos de dados, sejam eles relacionais ou *Not Only SQL* (NoSQL). Na mesma linha de raciocínio, as arquiteturas baseadas em microsserviços promovem uma heterogeneidade da tecnologia, visto que cada serviço pode ser implementado utilizando tecnologias mais apropriadas para atender adequadamente aos requisitos do produto (FOWLER; LEWIS, 2014).

4.2.1 Microsserviços *vs* Monolitos

Uma aplicação monolítica é construída como uma única unidade, formada por um único executável lógico. Quaisquer mudanças no sistema envolvem a construção e implantação de uma nova versão. Ao longo do tempo, essa abordagem torna-se difícil de manter e apresenta custos elevados para evoluir. Por outro lado, os microsserviços podem possuir múltiplas bases de código autônomas com o seu próprio ciclo de vida (Figura 12). Assim, apenas os serviços modificados precisam ser atualizados e reinicializados (NEWMAN, 2015).

Figura 12 – Esboço de implantação de uma aplicação monolítica *vs* microsserviços. a) As funcionalidades do monolito reunidas e replicadas em conjunto. b) As funcionalidades separadas em microsserviços e replicadas quando necessárias



Fonte: adaptado de Fowler e Lewis (2014).

A base de código de um monolito bloqueia as tecnologias para os programadores, qualquer alteração na linguagem ou estrutura pode causar um impacto em todo o sistema em forma de cascata. Entretanto, as interfaces simples de comunicação com acoplamento fraco e independentes da tecnologia favorece aos desenvolvedores na escolha das tecnologias mais adequadas para implementar um microsserviço.

4.2.2 Microsserviços e SOA

Os microsserviços formam um estilo arquitetônico derivado do modelo *Service-Oriented Architecture* (SOA)(NIKNEJAD et al., 2020). Sua definição parte da ideia que os sistemas são compostos de pequenas partes construídas independentemente e com a comunicação via passagem de mensagens. Eles herdam alguns recursos de SOA, mas tomam as definições com uma granularidade muito mais fina. Diferentemente do SOA e similar ao princípio da responsabilidade única da programação orientada a objetos (MARTIN, 2003), os microsserviços destacam a importância dos serviços serem pequenos, portanto,

facilmente reutilizáveis, facilmente compreendidos e até mesmo facilmente reconstruídos (DRAGONI et al., 2017).

4.2.3 Vantagens e Desvantagens dos Microsserviços

Uma arquitetura baseada em microsserviços traz alguns benefícios e algumas desvantagens (FOWLER; LEWIS, 2014). Assim, é necessário entendê-las para aplicar esse estilo aos domínios específicos. A seguir, são detalhadas as vantagens e desvantagens da utilização de uma arquitetura baseadas em microsserviços.

Vantagens

- Implantação independente: a distribuição das funcionalidades em serviços simples facilita a implantação, e como são autônomos, possuem uma menor probabilidade de causar problemas ao sistema quando ocorrem erros.
- Diversidade de tecnologia: o sistema completo formado por microsserviços pode combinar várias linguagens, estruturas de desenvolvimento e tecnologias de armazenamento de dados.

Desvantagens

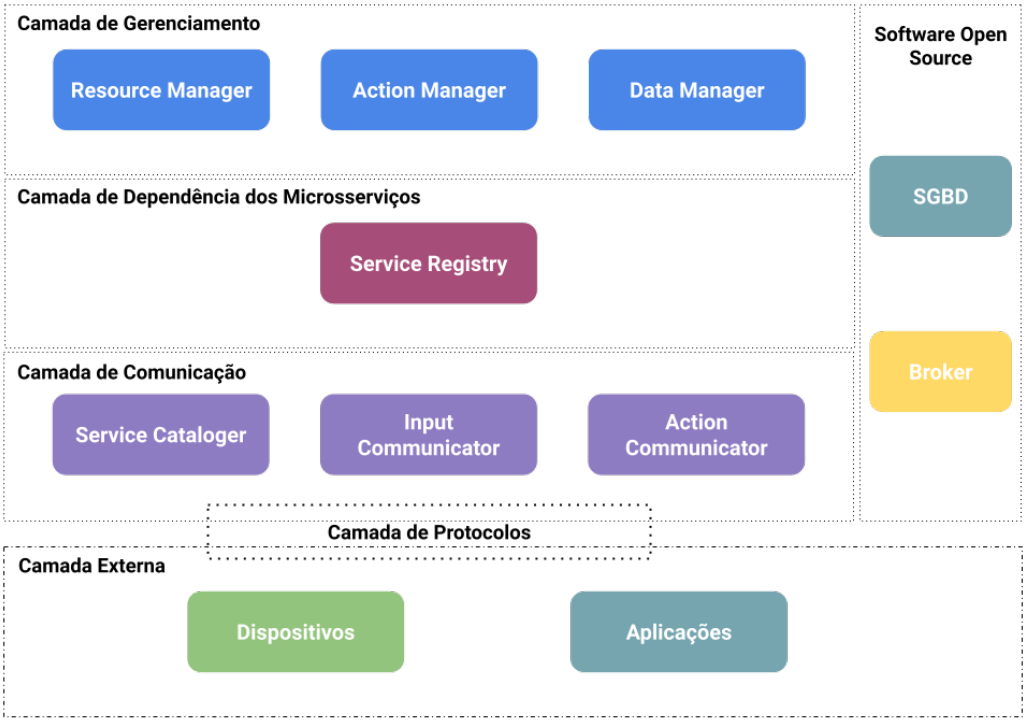
- Distribuição: os sistemas distribuídos tendem a ser difíceis de programar. Além disso, as chamadas remotas são mais lentas e podem correr o risco de falhar.
- Complexidade operacional: uma complexidade maior em manter e gerenciar vários microsserviços que estão sendo reimplantados regularmente.

4.3 Arquitetura do MORE4IoT

A arquitetura do MORE4IoT foi definida a partir dos requisitos descritos anteriormente. A arquitetura é baseada em microsserviços para solucionar os desafios de um *middleware* multiprotocolo para IoT. Percebe-se que a arquitetura orientada a microsserviços ainda não é predominantemente utilizada pelos middlewares na IoT. No entanto, entende-se que essa arquitetura pode auxiliar e abordar as funcionalidades necessárias para aplicações e dispositivos na IoT, fornecendo serviços de comunicação por protocolos diferentes, gerenciamento de dados, mecanismos de controle para as aplicações e gerenciamento de recursos. Dessa forma, o MORE4IoT facilita e habilita a construção de aplicações para IoT reduzindo a complexidade no gerenciamento de dispositivos heterogêneos, gerenciamento e tráfego de dados e desenvolvimento de aplicações. Na Figura 13, apresenta-se a arquitetura de microsserviços completa do MORE4IoT.

O MORE4IoT é constituído por quatro camadas de abstração para delimitar as funcionalidades dos microsserviços. As camadas são divididas entre a camada de gerenciamento, a camada de dependência dos microsserviços, a camada de comunicação e a camada externa ao *middleware* – que possui abstrações que facilitam a comunicação com o MORE4IoT. Essas abstrações encontradas na camada externa buscam oferecer a capacidade de comunicação por diferentes protocolos, tais como: o MQTT, o CoAP, o HTTP, AMQP e LoRaWAN. O *protocol adapter* é o componente fundamental para que os dispositivos possam enviar e receber dados, assim como as aplicações possam receber, enviar e controlar os dispositivos. Observa-se que este componente fica do lado do dispositivo para facilitar a comunicação com os microsserviços do MORE4IoT. Esse componente é abordado com mais detalhes ao longo desta seção.

Figura 13 – Arquitetura do MORE4IoT baseada em microsserviços



Fonte: autoria própria.

Atualmente, o MORE4IoT é composto por sete microsserviços que possuem funcionalidades bem definidas e que fornecem a capacidade de registrar e gerenciar recursos (*resource manager*), registrar e gerenciar mecanismos de controle (*action manager*), salvar e gerenciar dados (*data manager*), descoberta de microsserviços (*service registry*), disponibilização dos serviços para camada externa de forma síncrona (*service cataloger*), integração por diferentes protocolos para receber dados dos recursos de forma síncrona e assíncrona (*input communicator*) e o microsserviço (*action communicator*) responsável por resolver e enviar os dados por diferentes protocolos para os recursos (dispositivos ou aplicações) de forma síncrona ou assíncrona. Além disso, todos os microsserviços expõem uma API HTTP/REST para comunicação síncrona. No entanto, a comunicação predominante entre

os microsserviços do MORE4IoT é de forma assíncrona, utilizando um *broker* para entregar uma fila de mensagens por meio de um barramento.

Os detalhes de cada camada e seus respectivos microsserviços são apresentados na Seção 4.4. A seguir, descreve-se as formas de comunicação e a distribuição das responsabilidades no MORE4IoT.

4.3.1 Distribuição das Responsabilidades

Uma arquitetura baseada em microsserviços oferece a distribuição das responsabilidades para pequenos serviços, com funcionalidades específicas que se comunicam com mecanismos leves para atingir o objetivo designado. Utilizando uma arquitetura baseada em microsserviços, o MORE4IoT distribui as responsabilidades entre os serviços que implementam e colaboram por mecanismos de comunicação leves que podem ser síncronos ou assíncronos. A arquitetura do MORE4IoT oferece suporte ao gerenciamento dos recursos e dos dados, dividindo o gerenciamento, a comunicação e persistência de dados entre os microsserviços.

Existem três microsserviços responsáveis que formam e fornecem a comunicação com a camada externa: o *service cataloger*, *input communicator* e *action communicator*. O *service cataloger* é um microsserviço que fornece uma API síncrona que busca facilitar a comunicação das aplicações com o *middleware*, externando todas as funcionalidades que podem ser utilizadas pelas aplicações e validando as solicitações. O *input communicator* fornece um único ponto de entrada de dados para persistir os dados dos recursos por diferentes protocolos de comunicação. Esse serviço faz parte da responsabilidade de fornecer a interoperabilidade para dispositivos heterogêneos. Ao receber um novo dado, esse microsserviço valida a origem do dado e insere no fluxo de persistência.

O *action communicator* é responsável por receber o fluxo de atuação e resolver o envio das atuações por diferentes protocolos de comunicação, completando a responsabilidade de interoperar com dispositivos heterogêneos. A distribuição de responsabilidades das tarefas de enviar e receber por diferentes protocolos está relacionada com o fato que eles podem entrar em picos distintos de utilização. Dessa forma, pode-se introduzir um balanceador de carga de trabalho (*load balancer*) (STEEN; TANENBAUM, 2017) para escalonar os microsserviços horizontalmente, quando necessário, sendo uma abordagem importante no projeto de sistemas distribuídos quando existem serviços sem estado (*stateless service*) (BURNS, 2018).

A camada de gerenciamento é formada por três microsserviços que possuem papéis vitais para o funcionamento e gerenciamento dos dispositivos na IoT: o *resource manager*, o *action manager* e *data manager*. Essa divisão garante as responsabilidades únicas para cada microsserviço de gerenciamento. Além disso, cada microsserviço pode implementar

formas de persistência utilizando SGBDs ou banco de dados diferentes. Com isso, qualquer modificação, indisponibilidade ou migração pode ser facilmente resolvida de forma distribuída sem comprometer o funcionamento dos outros microsserviços.

O *resource manager* gerencia as informações sobre cada *resource* inserido em locais distintos. Esse recurso pode ser um dispositivo que possui sensores ou atuadores, ou pode ser uma aplicação que fornece dados ou está interessada nos dados que estão sendo produzidos pelos dispositivos. Dessa forma, as informações de cada recurso são armazenadas para serem utilizadas pelos microsserviços para validação, localização e comunicação. O processo de validação está relacionado com a verificação do *Universally Unique IDentifier* (UUID). Esse identificador foi desenvolvido para garantir a exclusividade da identificação com mecanismos que protegem a identidade do emissor (SCHAFFER; SCHARTNER; RASS, 2007). Todo novo recurso recebe um UUID para ser utilizado na validação da inserção de um novo dado e na inserção de uma nova atuação de controle (*action*). Assim, as aplicações e os *protocol adapter* devem utilizar o UUID para interagir com MORE4IoT para obter dados, inserir dados e utilizar as demais funcionalidades. A garantia de autenticidade auxilia no aspecto de segurança dos dispositivos gerenciados.

O *action manager* fornece as funcionalidades para intermediar todas as solicitações de controle aos recursos pelas aplicações. Esse microsserviço é responsável por receber e validar as novas *actions* (mais detalhes em 4.4.5) e notificar de forma assíncrona o *action communicator* para resolver o envio dessas *actions* por diferentes protocolos. Além disso, esse microsserviço registra o histórico e guarda o *status* das *actions* que estão disponíveis para acesso pela própria API em futuras análises e auditorias.

O *data manager* armazena dados de sensores, atuadores e processamento das aplicações por meio da inserção desses dados produzidos pelo *input communicator* devidamente identificado com o UUID. Os dados produzidos consistem nas informações geradas pelas capacidades de cada sensor, atuador ou aplicações em diferentes momentos do tempo e localização. Esse microsserviço fornece uma API para permitir o acesso aos dados atuais e históricos através do *service cataloger*.

A camada de dependência dos microsserviços é formada pelo microsserviço que possui função específica na arquitetura baseada em microsserviços. Na versão atual, o microsserviço *service registry* é importante e possui a função de registrar os endereços dos microsserviços do MORE4IoT. Por fim, a camada externa é formada pelos dispositivos e aplicações que utilizam o MORE4IoT *protocol adapter* para facilitar a comunicação por diferentes protocolos e auxiliar no desenvolvimento das regras de negócio nos dispositivos utilizando os dados obtidos por dispositivos distintos inseridos na IoT.

Como resultado da distribuição das funcionalidades, a carga de trabalho do MORE4IoT é resolvida por microsserviços distribuídos que possuem responsabilidades bem específicas, cooperando para fornecer todas as funcionalidades que são necessárias

para promover interoperabilidade entre dispositivos heterogêneos na IoT. A distribuição da capacidade de trabalho do MORE4IoT se reflete na camada do banco de dados, uma vez que cada microsserviço tem seu banco de dados e gerencia um conjunto de dados. Diferentes aplicações podem trabalhar de maneiras diferentes ocasionando cargas de trabalho distintas no MORE4IoT. Por exemplo, em uma aplicação com uma grande quantidade de sensores e um número reduzidos de atuadores, um microsserviço do *middleware* pode ser usado com uma maior carga de trabalho: o *input communicator*. Em outra aplicação, que possui muitos atuadores, o *action communicator* pode ser acionado e possuir uma carga de trabalho maior em um tipo de relacionamento 1:M, no qual poucos dispositivos produzem dados, mas ocorrem diversos processamentos de *actions*.

4.3.2 Formas de Comunicação

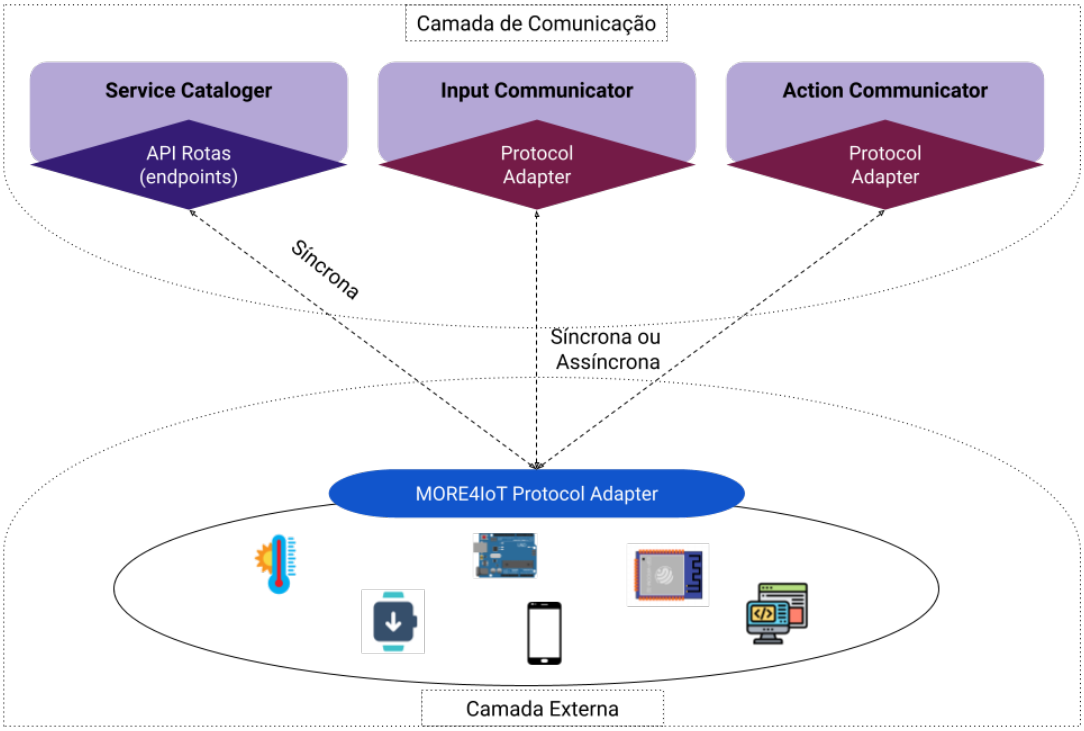
Os microsserviços do MORE4IoT interagem com mecanismos de comunicação bem definidos (Figura 14) utilizando comunicação síncrona ou assíncrona por meio de um adaptador de protocolos (*protocol adapter*) para acesso externo ou interno utilizando mensagens no formato *JavaScript Object Notation* (JSON). Os microsserviços cooperam constantemente para fornecer os serviços para aplicações e para permitir a interoperabilidade entre dispositivos na IoT. Essas decisões sobre como os microsserviços interagem podem incluir a escolha de protocolos de comunicação, a descoberta de serviços, o tratamento de mensagens assíncronas e o formato das mensagens. O MORE4IoT implementa a comunicação externa com auxílio do *protocol adapter*, que pode se comunicar de forma síncrona ou assíncrona com o *middleware* dependendo da escolha do protocolo a ser utilizado. O *protocol adapter* foi implementado e mais detalhes podem ser obtidos na Seção 4.5.

O *protocol adapter* pode ser implementado por diferentes ferramentas (SDK, API, *plugin*) que abstraem essa comunicação, auxiliando e facilitando os desenvolvedores durante o processo de construção das aplicações. Os dispositivos e as aplicações são capazes de utilizar as ferramentas que são disponibilizadas para realizar a comunicação com o MORE4IoT. O MORE4IoT também possui adaptadores de protocolo dentro dos microsserviços. Os dispositivos proprietários ou aplicações de terceiros devem disponibilizar formas de configurar esses mecanismos para comunicação e conexão com o *middleware*. O MORE4IoT expõe os endereços e portas públicas da camada de comunicação para esse tipo de configuração, os mesmos utilizados pelo *protocol adapter*.

Os protocolos MQTT e AMQP são protocolos baseados em mensagens assíncronas utilizando um padrão de *design publish/subscribe*. Esse tipo de comunicação evita a latência adicional do bloqueio de interações síncronas de um padrão *request/response*. O MORE4IoT provê a comunicação com esses protocolos por intermédio de um *broker* RabbitMQ para garantir o encaminhamento das mensagens publicadas. O RabbitMQ[1] é

[1]<https://www.rabbitmq.com/>

Figura 14 – Formas de comunicação entre a camada de comunicação e a camada externa



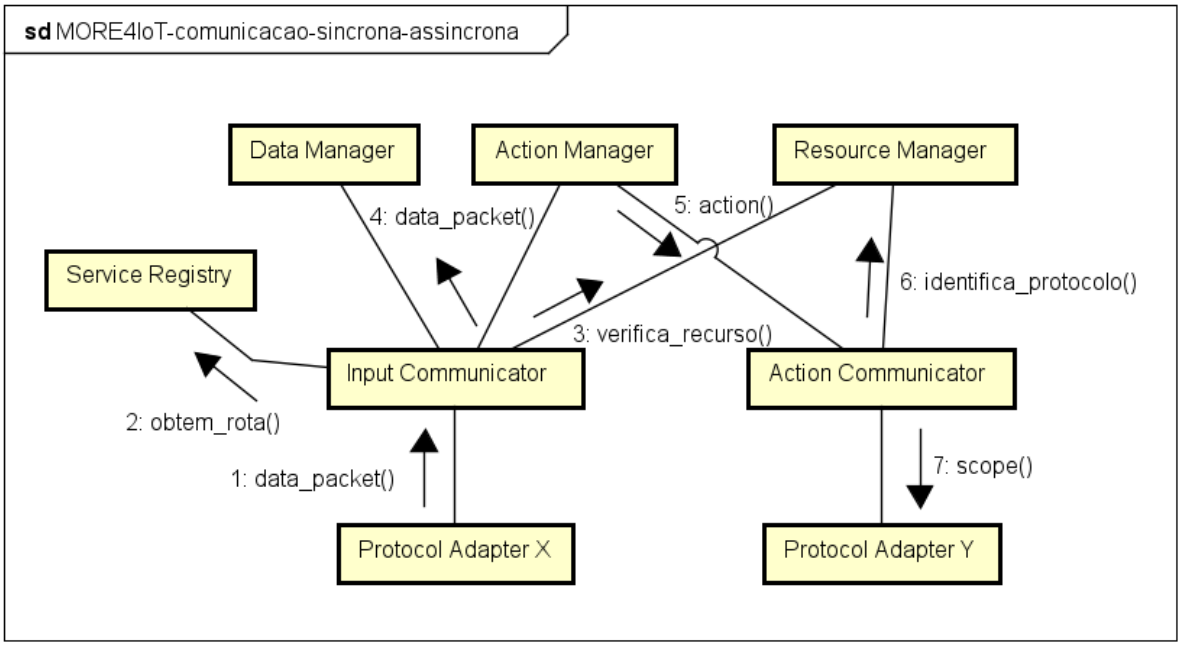
Fonte: autoria própria.

um *broker* leve e *open source* amplamente adotado pela comunidade. O *broker* garante o envio das mensagens para os assinantes (*subscribers*). O MORE4IoT utiliza o *broker* de forma completamente ampla para atender os microsserviços e também para receber os dados dos dispositivos através do *protocol adapter* utilizando os protocolos MQTT ou AMQP.

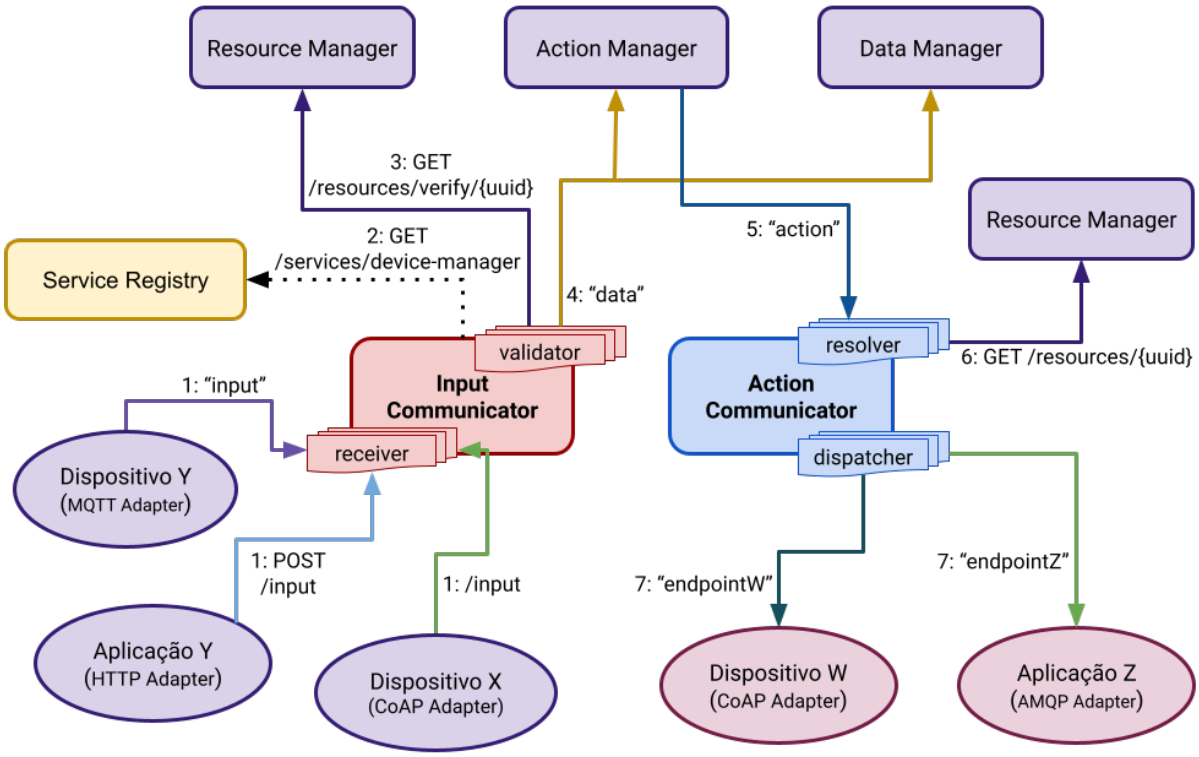
Vários eventos podem ocorrer e precisam ser transmitidos para os microsserviços da camada de comunicação ou realizar troca de mensagens internamente. Cada tipo de evento é mapeado para um tópico, ao qual os serviços interessados podem se inscrever para receber novas mensagens por meio de filas mantidas pelo RabbitMQ. Por padrão, cada assinatura criará uma nova fila de mensagens.

Como exemplo, na Figura 15a, apresenta-se o diagrama de comunicação do MORE4IoT para recebimento de dados com o processamento das atuações. Essa comunicação pode ser síncrona ou assíncrona. Esse diagrama é utilizado pelo exemplo apresentado na Figura 15b. Para compreensão, deve-se utilizar a numeração do diagrama para visualizar as mesmas etapas no exemplo. Assim, é possível visualizar a troca de mensagens durante o (1) recebimento de um novo dado de um dispositivo utilizando o tópico "*input*"; (2) e (3) o processo de validação desses dados; (4) a persistência dos dados e notificação para verificação das atuações utilizando o tópico "*data*"; (5) o envio das ações de controle para o *action communicator* utilizando o tópico "action". (6) O *action communicator* realiza uma requisição ao *resource manager* para identificar os protocolos

Figura 15 – Forma de comunicação assíncrona e síncrona dos microsserviços do MORE4IoT com a camada externa. a) Diagrama de comunicação. b) Exemplificação do diagrama de comunicação



(a)



(b)

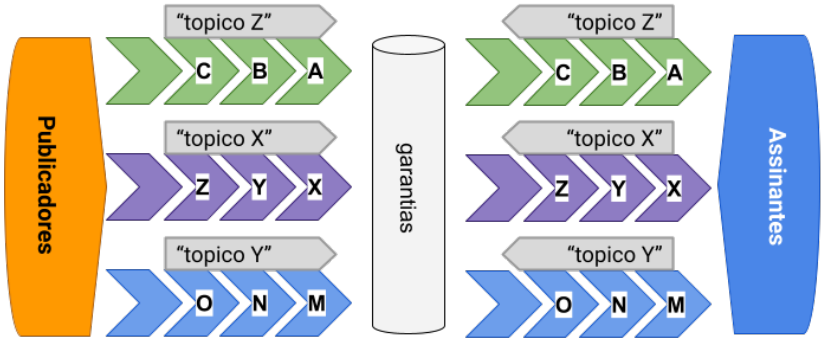
Fonte: autoria própria.

que os recursos entendem. Com isso, o *action communicator* envia pelos protocolos que

os dispositivos atuadores ou as aplicações escutam, com o objetivo de (7) enviar os dados e comandos (*scope*) pelos "endpoints" dos recursos de forma assíncrona para os dispositivos que utilizam os protocolos MQTT ou AMQP, também, de forma síncrona para os dispositivos que utilizam o HTTP ou CoAP.

As mensagens enviadas para um tópico específico são enviadas para todas as suas filas inscritas em ordem garantida pelo *broker* (Figura 16). Seguindo esse padrão pub/sub, um *protocol adapter* (comunicação externa) ou microsserviço (comunicação interna) envia ou recebe por um tópico, e o *broker* garante a entrega por ordem de envio. Vale ressaltar que os outros protocolos podem ou utilizam outras formas de comunicação, sejam síncronas ou assíncronas. Portanto, não são intermediadas pelo *broker*. Dessa forma, o MORE4IoT implementa diferentes formas de comunicação para atender diferentes dispositivos e aplicações para IoT.

Figura 16 – Padrão *publish/subscribe* mediado por um *broker*



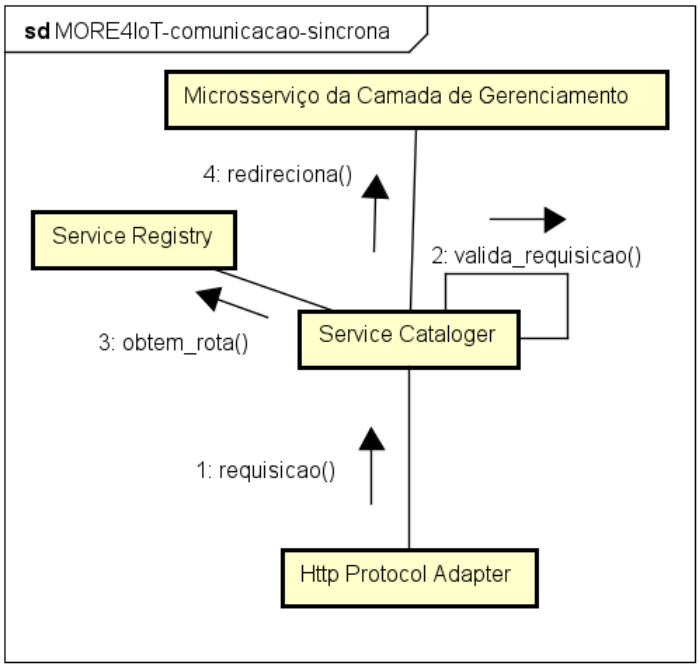
Fonte: baseado em Jacobsen (2009).

Além disso, o MORE4IoT também implementa a comunicação interna de forma síncrona utilizando o HTTP/REST, expondo APIs por todos os microsserviços. Dessa forma, o *service cataloger* pode expor o conjunto das funcionalidades utilizando um *proxy design pattern* para camada externa a ser utilizada pelos adaptadores do HTTP/REST usando uma API cliente ou outras ferramentas que implementam o protocolo. O API *Proxy* reúne todas as funcionalidades expostas pelos microsserviços da camada de gerenciamento e implementa mecanismo de autorização e validação.

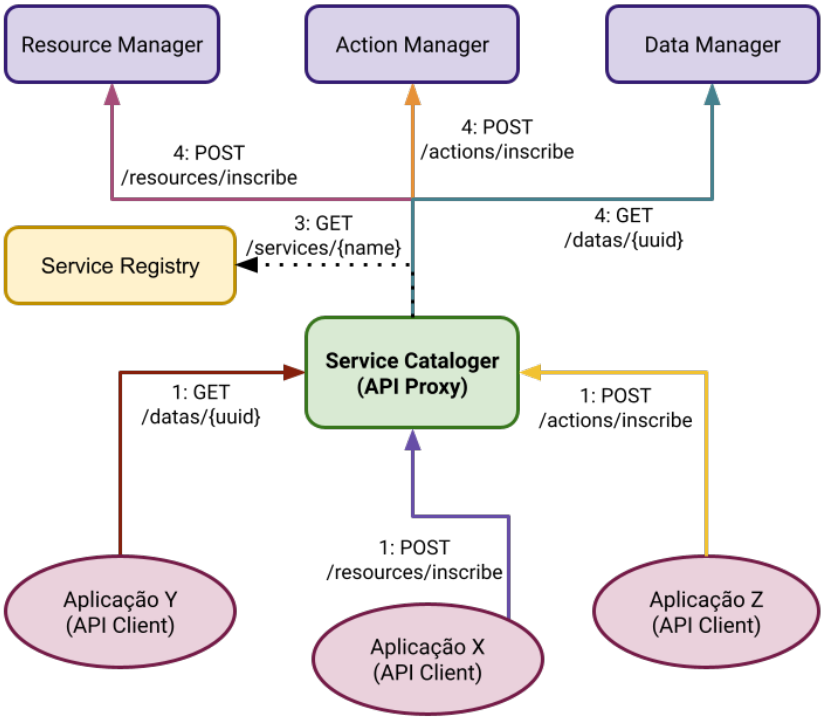
Na Figura 17a, apresenta-se o diagrama de comunicação síncrona com o *service cataloger*, expondo as funcionalidades da camada de gerenciamento. Para exemplificar, a Figura 17b mapeia esse diagrama. Uma API *Proxy* intermedeia as requisições para controlar o acesso as funcionalidades e mantendo alguns aspectos de segurança e privacidade dos microsserviços internos do sistema. No exemplo, o *protocol adapter* que implementa o HTTP/REST (API *Client*) realiza algumas requisições de inserção e recuperação de dados aos microsserviços gerenciadores intermediados pela API *Proxy*. Essa forma de comunicação

é replicada para todos os microsserviços da camada de gerenciamento e suas respectivas funcionalidades. Mais detalhes sobre as funcionalidades são descritas na próxima seção.

Figura 17 – Forma de comunicação síncrona com o MORE4IoT. a) Diagrama de comunicação. b) Exemplificação do diagrama de comunicação



(a)



(b)

Fonte: autoria própria.

4.4 Detalhes dos Microsserviços

Nesta seção, os microsserviços que formam a arquitetura do MORE4IoT são detalhados. Essa primeira versão foi desenvolvida abordando os principais desafios que um *middleware* multiprotocolo para IoT deve resolver. Todavia, a utilização dessa arquitetura baseada em microsserviços proporciona uma capacidade evolutiva. Dessa forma, o MORE4IoT pode e deve continuar evoluindo para disponibilizar novas funcionalidades para as aplicações, melhorando os mecanismos de interoperabilidade e empregando outros requisitos para IoT. Para mais informações sobre o código ou documentação, o MORE4IoT está utilizando o GitHub[2] para o versionamento de código. Os microsserviços, os SDKs, APIs, as aplicações ou outras ferramentas também são disponibilizadas nesse repositório[3].

4.4.1 *Service Cataloger*

O *service cataloger* utiliza uma abordagem baseada em um *proxy design pattern* para disponibilizar as funcionalidades dos microsserviços do MORE4IoT para as aplicações por meio de uma API *Proxy*. Essas funcionalidades devem ser utilizadas pelas aplicações para interagir com os microsserviços internos do MORE4IoT. Nessa primeira versão, as funcionalidades estão relacionadas com a capacidade dos dispositivos e aplicações interoperarem. As principais funções dos microsserviços da camada de gerenciamento são agrupadas por esse microsserviço – integrando, encapsulando e controlando o acesso dessas funcionalidades do MORE4IoT, entre as quais estão: inserir um novo recurso (*resource*) para gerenciamento; criar um novo mecanismo de controle pela aplicação utilizando os dispositivos (*action*); obter dados atuais e históricos dos dados gerados pelos recursos.

Tabela 10 – *Endpoints* expostos pelo *service cataloger*

GET /resources	obter os *resources* gerenciados
POST /resources/inscribe	inscrição de um novo *resource*
PUT /resources/update	atualização de um *resource*
GET /resources/{uuid}	solicitar informações do *resource*
GET /resources/verify/{uuid}	verificar existência de um *resource*
DELETE /resources/delete/{uuid}	remover um *resource*
GET /data	solicitar os dados gerados pelos *resources*
GET /data/last/{uuid}	solicitar o último *data packet* inserido
GET /data/{uuid}	obter os dados produzidos por *resource*
DELETE /data/delete/{uuid}	remover os dados produzidos por *resource*
GET /actions	obter as *actions*
POST /actions/inscribe	inscrever uma nova ação de controle
GET /actions/{uuid}	solicitar as *actions* de uma aplicação
DELETE /actions/delete/{id}	remover uma *action*

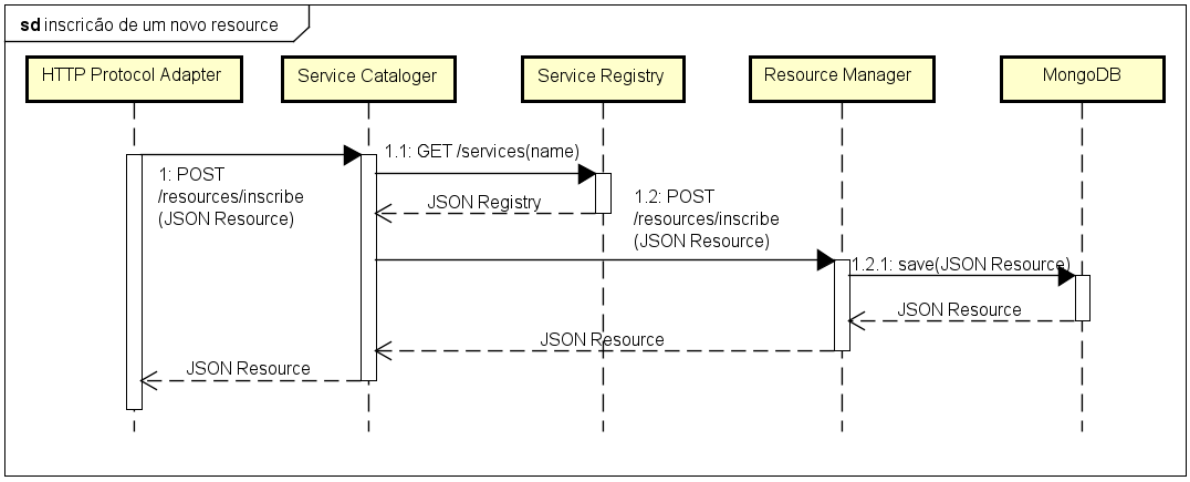
[2] <https://github.com/iotufersa/more4iot>
[3] <https://github.com/iotufersa>

Na Tabela 10, mostra-se os *endpoints* disponibilizados por esse microsserviço para comunicação síncrona, por meio de uma API *Client* (*HTTP protocol adapter*). Além disso, uma breve explicação para cada *endpoint* exposto. Todavia, mais detalhes das funcionalidades podem ser obtidos nos detalhes dos microsserviços: *resource manager*, *data manager* e *action manager*.

4.4.2 *Resource Manager*

O microsserviço *resource manager* é responsável por gerenciar, persistir e controlar o acesso às informações dos recursos inseridos na IoT. Dessa forma, ele recebe os novos recursos enviados via *service cataloger* pelas aplicações utilizando a HTTP/REST API exposta. Portanto, nessa primeira versão, esse microsserviço fornece os *endpoints* para registrar um novo recurso, obter as informações do recurso gerenciado, verificar a existência de algum recurso, fornecer todos os recursos gerenciados e remover recurso que não está mais disponível. A seguir são explicados os principais *endpoints* disponibilizados.

Figura 18 – Diagrama de sequência da inscrição de um novo *resource*



Fonte: autoria própria.

Dentre as funcionalidades, o microsserviço disponibiliza inscrição de um novo recurso (Figura 18) inserido na IoT pelas aplicações por meio de uma API, usando o comando "POST" para o *endpoint* "/resources/inscribe". As aplicações devem enviar uma requisição "POST" por meio de uma API *Client* através do *service cataloger* que irá redirecionar a requisição para esse microsserviço. As aplicações precisam enviar um JSON *Resource* com, pelo menos, os campos obrigatórios desse objeto. Ao final, o *resource* é salvo no banco de dados (MongoDB).

Figura 19 – Formato do JSON *Resource*

```
{
    "uuid": {
        type: String,
        required: true,
        unique: true,
    },
    "name": {
        type: String
    },
    "lat": {
        type: Number,
    },
    "lon": {
        type: Number,
    },
    "resource": {
        type: Array,
        items: String
    },
    "uri": {
        type: String,
        required: true
    },
    "protocol": {
        type: String,
        required: true
    },
    "describe": {
        type: String
    },
    "isDevice": {
        type: Boolean,
        required: true
    }
}
```

Fonte: autoria própria.

Na Figura 19, apresenta-se os campos do JSON *Resource* e as designações de tipos e obrigatoriedade. Os campos “uuid”, “uri”, “protocol” e “isDevice” são as informações mínimas/obrigatórios para gerenciar o recurso. Ademais, os outros campos podem ser utilizados em novos serviços que utilizam geolocalização (“lat” e “lon”) ou busca por um recurso específico (“resource”). Em complemento, os campos “name” e “describe” auxiliam na identificação do recurso.

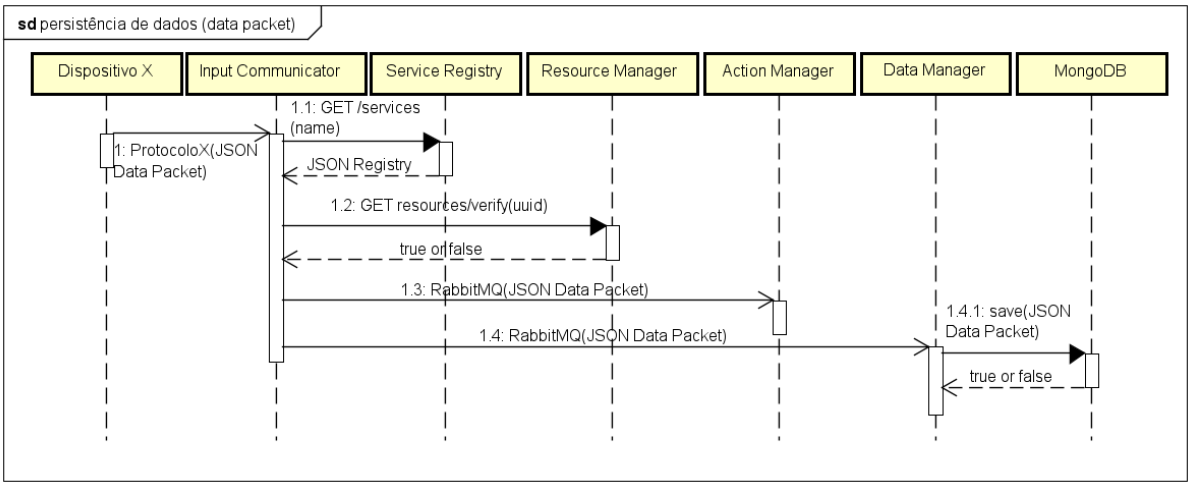
Os microserviços do MORE4IoT precisam verificar a existência de um *resource* gerenciado em diferentes momentos do funcionamento de cada microsserviço. Dessa forma, o *resource manager* disponibiliza um *endpoint* “GET /resources/verify/{uuid}” para ser utilizado nessas verificações. Além disso, esse microsserviço guarda em *cache* essas

informações para reduzir a quantidade de transações com o banco de dados, reduzindo a latência da resposta na comunicação síncrona. Para essa primeira versão, o microsserviço ainda possui os *endpoints*: "GET /resources/{uuid}" para buscar e retornar um *resource* para o microsserviço durante a descoberta de protocolo ou URI, por exemplo. Também pode realizar uma busca por todos os *resources* disponíveis usando "GET /resources". Além disso, uma aplicação pode remover um *resource* que não está mais disponível para gerenciamento utilizando o *endpoint* "DELETE /resources/delete/{uuid}". Por fim, uma aplicação pode atualizar um *resource* através do *endpoint* "PUT /resources/update".

4.4.3 *Data Manager*

O *data manager* é um microsserviço fundamental para gerenciar os dados produzidos pelos dispositivos na IoT. Ele também pode criar um quadro histórico de dados para ser utilizado em novos serviços providos pelo MORE4IoT, ou até mesmo, a recuperação de dados para as aplicações e também para auditorias ou resolução de problemas com dispositivos em ambientes distintos. Dessa forma, os *resources* inseridos na IoT utilizam o *protocol adapter* para enviar os dados gerados para o *input communicator* para, primeiramente, realizar uma validação e posteriormente enviar esses dados pelo fluxo interno de comunicação por meio do RabbitMQ, conforme apresentado na Figura 20.

Figura 20 – Diagrama de sequência da persistência de dados por um dispositivo



Fonte: autoria própria.

Assim, o serviço recebe um *data packet* no formato de um JSON contendo os metadados desse pacote, formado pelo UUID do recurso ("uuid"), o "data" gerado pelo recurso e a localização atual desse recurso ("lat" e "lon"), como apresentado na Figura 21. Vale ressaltar que os dispositivos e aplicações que estejam inscritos nas *actions* receberão os dados que estão sendo persistidos. O envio dos dados é resolvido pelo *action manager* e, posteriormente, pelo *action communicator*. Mais detalhes desse envio são apresentados nos respectivos microsserviços.

Figura 21 – Formato do JSON *Data Packet*

```
{
    "uuid": {
        type: String,
        required: true
    },
    "lat": {
        type: Number,
    },
    "lon": {
        type: Number,
    },
    "data": {
        type: Object,
        required: true
    }
}
```

Fonte: autoria própria.

O *input communicator* insere o JSON *data packet* no tópico "data". O *data manager* está inscrito (*subscribe*) nesse tópico para receber os *data packets* que são inseridos pelo *input communicator*. Até este ponto, o *data packet* continua sendo o mesmo inserido pelo *input communicator*, no entanto, esse microsserviço recebe e realiza uma nova validação de campos antes de persistir esses dados. Durante a persistência, são inseridos mais alguns campos para auditoria, como mostrado na Figura 22. Além dos campos já existentes, são inseridos os campos: _id, createdAt, updatedAt e ___v.

Figura 22 – Formato do JSON *Data Packet* com campos adicionais após persistência

```
{
    "_id":"60b549f4ee515800139de8c9",
    "uuid":"1b6a9020-c1fc-11eb-834a-51fd5324303e",
    "data":{
        "temperature":31,
        "humidity":30
    },
    "createdAt":1622493684673
    ,
    "updatedAt":1622493684673,
    "__v":0
}
```

Fonte: autoria própria.

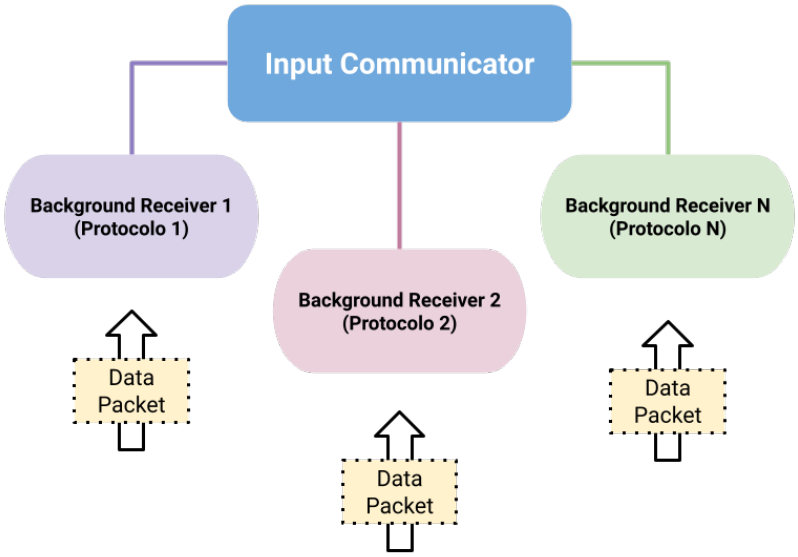
Todavia, o *data manager* expõe uma HTTP/REST API para utilização síncrona

com as aplicações e os demais microsserviços do MORE4IoT interessados nesses dados. Nessa primeira versão, os *endpoints* são bem definidos para expor as funcionalidades que o *data manager* possui como responsabilidade. Esse microsserviço busca gerenciar e disponibilizar os dados que foram gerados pelos *resources*. Sendo assim, são disponibilizados os *endpoints*: "GET /data" para obter todos os dados gerenciados pelo microsserviços inseridos pelos *resources*; "GET /data/{uuid}" para obter os dados gerados por um *resource* específico; "DELETE /data/delete/{uuid}" para remover os dados gerados por um *resource* específico; por fim, para obter o último *data packet* produzido por um *resource* utilizando o *endpoint* "GET /data/last/{uuid}". Com a arquitetura baseada em microsserviços, novos microsserviços podem ser construídos e funcionar em conjunto ao *data manager* para trazer novas funcionalidades, até mesmo, em novas versões, outras funcionalidades podem ser inseridas para oferecer melhores mecanismos de buscas ou descoberta de informação nesse microsserviço.

4.4.4 *Input Communicator*

O *input communicator* disponibiliza a comunicação com a camada externa por diferentes protocolos. Esses protocolos são implementados e possuem características próprias para comunicação. Na Figura 23, apresenta-se a forma de comunicação com esse microsserviço. O microsserviço cria diferentes *background receivers* para permitir o recebimento do dado (*data packet*), a ser persistido, do recurso (*resource*) gerenciado pelo MORE4IoT.

Figura 23 – Mecanismo de comunicação do *input communicator*



Fonte: autoria própria.

Cada *background receiver* funciona como uma porta para recebimento de dados bem definido pela arquitetura e implementa os mecanismos de comunicação para cada protocolo. Dessa forma, o *input communicator* também utiliza um *protocol adapter* para

disponibilizar a comunicação por diferentes protocolos. Assim, novos protocolos podem ser acrescentados sem quebrar o funcionamento dos outros, proporcionando um mecanismo evolutivo para novas versões desse microsserviço.

O microsserviço disponibiliza o endereço e porta para ser utilizado pelo *protocol adapter* da camada externa para comunicação. No caso do HTTP, é disponibilizado um *endpoint* "POST /input" para ser utilizado pelos adaptadores. O mesmo ocorre com os outros protocolos. No caso do CoAP, o *data packet* é enviado utilizando uma rota similar ao HTTP ("/input"). Diferentemente do MQTT ou AMQP, que utilizam o tópico "input" para envio do *data packet* através do *protocol adapter*. O MORE4IoT disponibiliza as ferramentas do lado do cliente (aplicações e dispositivos) para resolver o processo de comunicação com o *input communicator*.

No *data packet*, apenas o identificador ("uuid") e os dados ("data") são campos obrigatórios. Um exemplo do pacote é apresentado na Figura 24. Os pacotes recebidos pelos *background receivers* são direcionados para o processo de validação desse pacote. Nesse ponto, um *background validator* é acionado para validar os campos que formam o *data packet*. Assim, o *protocol adapter* não fica bloqueado e retorna a funcionar normalmente. O pacote sendo validado (de acordo com o formato definido na Figura 21), entra no fluxo de persistência e controle de ação, conforme mostrado anteriormente na Figura 15. O *protocol adapter* deve construir o pacote seguindo a estrutura definida. O *input communicator* seleciona tais campos, qualquer outro campo é descartado. Além disso, o pacote é descartado quando não possui os campos obrigatórios ("uuid" e "data"), pelo menos.

Figura 24 – Um exemplo do JSON *data packet* gerado por um *protocol adapter*

```
{
    "uuid":"d4a45c32-3957-480e-9570-5aab4599a45c",
    "data": {
        "temperature":31,
        "humidity":30
    }
}
```

Fonte: autoria própria.

4.4.5 *Action Manager*

O *action manager* é um microsserviço que está relacionado com os desafios da distribuição de dados e controle dos dispositivos pertencentes as aplicações. Dessa forma, esse microsserviço busca auxiliar com uma abordagem para solucionar esses desafios. Nessa primeira versão, o microsserviço disponibiliza as funcionalidades essenciais para que as

aplicações possam controlar o tráfego de dados entre seus dispositivos. O *action manager* fornece um mecanismo de controle que pode ser utilizado pelas aplicações durante os direcionamentos e regras que essas aplicações precisam resolver. Ele oferece uma estrutura chamada *Action*. O JSON *Action* é formado por campos bem definidos para que as aplicações tenham a autonomia de controle dos seus dispositivos.

Os campos que formam o JSON *Action* são apresentados na Figura 25. A *action* é formada pelos campos: *creator*, *origin*, *receiver*, *scope*, *lifetime* e *status*. O "creator" guarda o identificador do criador da *action*, assim, uma aplicação deve ser o criador dessa *action*. O campo "origin" guarda os identificadores dos *resources* que geram os *data packet*. O "receiver" é formado pelos campos: *identifiers*, *protocol* e URI. O "identifiers" guarda os *resources* de destino do *scope*, além disso, o "protocol" e URI são os padrões para envio pelo mesmo protocolo direcionado ao *identifiers*, mesmo não sendo um campo obrigatório. O "scope" guarda os comandos para os dispositivos (*commands*) e possui um campo de "data" para a inserção do *data packet* recebido da origem ou a inserção de um *default data packet* pela aplicação, durante a construção do JSON *Action*.

O "lifetime" possui as informações necessárias para duração da *action*, o *action manager* vai decrementar o tempo de duração (*count*) ao longo dos disparos de dados recebidos dos *resources* e verificar o campo "validity". O campo "validity" é utilizado para os três estados possíveis de uma *action*: primeiro, para uma *action* que é executada instantaneamente, por exemplo, um comando de ligar uma luz da sala naquele momento; segundo, para uma *action* que é executada por um período de tempo, por exemplo, realizar uma ação de atuação por dez vezes, dessa forma, o campo "count" é utilizado para mapear essa quantidade de vezes; por fim, uma *action* permanente, ou seja, uma ação que é executada infinitamente, e é apenas desativada por uma aplicação modificando o "status" da *action*. O campos "status" informa para o MORE4IoT ou para as aplicações que a *action* está ativa ou inativa. Em *actions* ativas o "status" é *true* e em *actions* inativas é *false*. O repositório desse microsserviço possui mais informações para configuração das *actions*.

A *action* é construída pelas aplicações utilizando um HTTP *protocol adapter* que disponibiliza um *builder* (*builder pattern*) para a construção das *actions*. As requisições síncronas são autenticadas e redirecionadas pelo *service cataloger* para o *action manager*. Dessa forma, o *action manager* recebe o JSON *Action*, realiza o processo de validação dos campos, conforme apresentado na Figura 26. As *actions* entram em processo de execução instantânea/síncrona ou ficam aguardando o disparo de algum *data packet* recebido a partir de um *resource* (*origin*). Como resultado, esse microsserviço oferece uma abordagem para evitar a distribuição de dados entre aplicações desconhecidas, e também oferece autonomia para aplicações. Todavia, novas abordagens podem ser inseridas por meio de novos microsserviços para oferecer funcionalidades, outras formas de controle ou mecanismo

Figura 25 – Formato do JSON *Action*

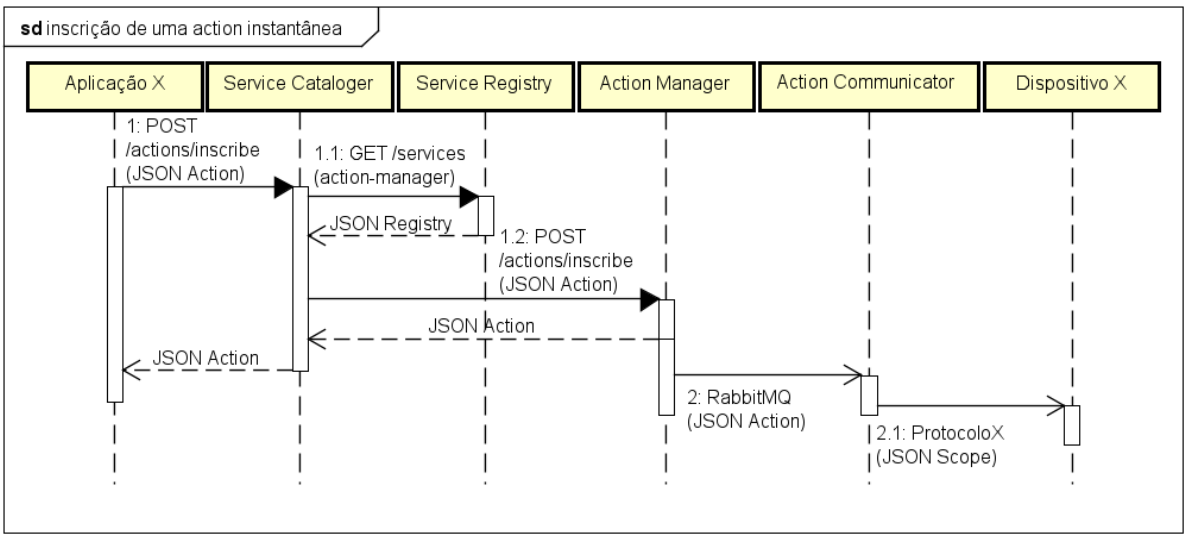
```
{
    "creator":{
        type: String,
        required: true
    },
    "origin": {
        type: Array,
        items: String
    },
    "receiver": {
        "identifiers": {
            type: Array,
            items: String
        },
        "protocol": String,
        "uri": String
    },
    "scope": {
        "data": {
            type: Object
        },
        "commands": {
            type: Object
        }
    },
    "lifetime": {
        "validity": {
            type: Boolean,
            required: true,
        },
        "count": {
            type: Number,
            required: true
        }
    },
    "status": {
        type: Boolean,
        required: true
    }
}
```

Fonte: autoria própria.

de obtenção de dados.

Em complemento, políticas de segurança e privacidade podem ser inseridas para que esses dados possam trafegar entre aplicações sem comprometer o funcionamento ou vazamento de dados entre aplicações e dispositivos. Muito ainda pode ser pesquisado e elaborado para essas funcionalidades. Entretanto, essa arquitetura baseada em microsserviços permite a inserção de novos microsserviços sem quebrar as funcionalidades existentes. Na

Figura 26 – Diagrama de sequência de uma *action* instantânea/síncrona



Fonte: autoria própria.

versão atual, o MORE4IoT está direcionado a oferecer interoperabilidade entre dispositivos e aplicações na IoT.

4.4.6 *Action Communicator*

O *action communicator* recebe as *actions* enviadas pelo *action manager* por meio do fluxo de mensagens assíncronas intermediado pelo RabbitMQ. A função desse microsserviço é receber essas *actions* e resolver as formas de envio para diferentes protocolos de comunicação (conforme foi apresentado na Figura 15). O formato da JSON *Action* está descrito no tópico sobre o microsserviço *action manager* (Subseção 4.4.5). O *action communicator* complementa as funcionalidades para promover interoperabilidade entre dispositivos na IoT. Esse microsserviço, juntamente do *input communicator*, implementa diferentes protocolos de comunicação para receber e enviar pacotes de dados. Esses protocolos possuem características próprias para comunicação. Sendo assim, o *action communicator* torna-se uma espécie de cliente para diferentes protocolos provendo o envio desses pacotes utilizando *background dispatchers*.

Primeiramente, antes do envio da *action*, esse microsserviço realiza uma verificação de protocolo e *endpoint* para envio. Essa verificação se dá por meio da verificação dos campos "protocol" e "uri" disponibilizados no objeto JSON da *action*. Em caso da inexistência dessas informações, o microsserviço realiza uma requisição para o *resource manager* para obter as informações de envio para cada recurso pertencente a essa *action*. Em caso de sucesso, o *action communicator* seleciona apenas os campos do "scope" da *action* constituídos pelos campos "data" e "commands" para envio pelos protocolos designados. Por exemplo, "data" possui os dados do dispositivo gerador e o "commands" possui os

comandos inseridos pelas aplicações para controlar seus dispositivos atuadores, no exemplo, os "commands" podem conter dados para configuração de cálculos a serem realizados no dispositivo (*edge computing*), conforme apresentado na Figura 27. Por fim, o "data", os "commands" e as informações sobre protocolo e *endpoint* do dispositivo são repassados para um *background dispatcher* por meio do *protocol adapter*.

Figura 27 – Exemplo do *scope* extraído de uma *action* preparado para envio aos dispositivos atuadores

```
{
    "data": {
        "soil-moisture":15
    },
    "commands": {
        "dry-soil": 30
    }
}
```

Fonte: autoria própria.

Diferentemente do *input communicator*, o *action commmunicator* implementa as formas de envio por cada protocolo por meio do *background dispatcher*. Dessa forma, o microsserviço cria diferentes *background dispatcher* para permitir o envio do pacote por diferentes protocolos. Cada *background dispatcher* funciona como um cliente bem definido pela arquitetura e implementa os mecanismos de comunicação para cada protocolo. Assim como o *input communicator*, diferentes protocolos podem ser acrescentados sem quebrar o funcionamento, dessa maneira, proporcionando um mecanismo evolutivo para novas versões desse microsserviço. Os microsserviços *input* e *action communication* atuam como facilitadores da comunicação com os dispositivos por diferentes protocolos. As tomadas de decisão e regras de negócio são de responsabilidade das aplicações. Dessa forma, o MORE4IoT busca habilitar a construção das aplicações. Com isso, disponibiliza as ferramentas para essas aplicações, sem interferir como elas funcionam mas provendo a interoperabilidade entre os dispositivos.

4.4.7 *Service Registry*

O *service registry* é fundamental para o funcionamento dos microsserviços do MORE4IoT. Sua remoção pode ser considerada quando os microsserviços são implantados no mesmo servidor, assim, reduzindo a quantidade de chamadas remotas para descoberta dos microsserviços em execução. Todavia, com a advento da *Cloud*, *Fog* e *Edge* os sistemas tendem a ficar completamente distribuídos entre esses locais. Dessa forma, necessitam

de mecanismos que façam esse gerenciamento para que os serviços possam realizar essa descoberta.

Na versão atual, o *service registry* possui uma responsabilidade bem definida para funcionar com os microsserviços do MORE4IoT. Ele reúne os endereços dos microsserviços em funcionamento. Assim, os microsserviços devem, periodicamente, enviar as informações de acesso para serem disponibilizadas para os demais. Esse mecanismo utiliza um padrão chamado *server-side discovery pattern*. Ele dispobiliza uma HTTP/REST API para receber um JSON *Registry*. Para isso, expõe-se dois *endpoints*: "POST /registry" para persistência das informações para acesso e "GET /registry/{service-name}" para solicitar as informações de acesso de um microsserviço através do nome. Nessa versão, o formato JSON *Registry* utilizado para enviar essas informações é composto pelos campos: *name*, IP e *port*.

4.5 Pilha de Tecnologias

Os microsserviços do MORE4IoT podem ser mantidos em repositórios separados e implantados separadamente, podem utilizar diferentes formas de persistência e tecnologias para desenvolvimento. Os microsserviços podem ser desenvolvidos com as tecnologias mais adequadas para cada finalidade. Na versão atual[4], os microsserviços foram desenvolvidos com as tecnologias apresentadas na Tabela 11.

Tabela 11 – Tecnologias utilizadas no desenvolvimento do MORE4IoT

Microsserviço	*Linguagem*	*SGBD*	*Cache*	*Broker*
Resource Manager	JavaScript/Node.js	MongoDB	Redis	-
Data Manager	JavaScript/Node.js	MongoDB	-	RabbitMQ
Action Manager	JavaScript/Node.js	MongoDB	-	RabbitMQ
Service Cataloger	JavaScript/Node.js	-	Redis	-
Input Communicator	JavaScript/Node.js	-	-	RabbitMQ
Action Communicator	JavaScript/Node.js	-	-	RabbitMQ
Service Registry	JavaScript/Node.js	Redis	-	-

O JavaScript/Node.js foi adotado como a linguagem para construção dos microsserviços devido a curva de aprendizagem rápida, permitindo que novos estudantes da linguagem iniciem na contribuição do desenvolvimento das novas versões dos microsserviços do MORE4IoT. Além disso, o MongoDB é amplamente utilizado por causa da flexibilidade oferecida por ser um banco de dados baseado em documentos – o que ajuda na persistência de estruturas heterogêneas requeridas para aplicações na IoT. O Redis foi adotado como mecanismo de armazenamento para *cache*. Além da adoção do RabbitMQ para

[4] <https://github.com/iotufersa/more4iot>

comunicação assíncrona entre os microsserviços do MORE4IoT. Alguns microsserviços não utilizam nenhum SGBD devido ao fato que esses possuem responsabilidades que não estão relacionadas ao armazenamento de dados.

Outras tecnologias e linguagens foram adotadas para desenvolver as ferramentas que promovem a capacidade de comunicação com os microsserviços do MORE4IoT. Na Tabela 12, mostra-se os SDKs e linguagens utilizadas para o desenvolvimento das ferramentas para auxiliar na construção de aplicações para IoT. Foram construídos os SDKs para as linguagens: C++ (MORE4IoT Arduino SDK[5]), JavaScript (MORE4IoT JS SDK[6]) e Dart (MORE4IoT Dart SDK[7]).

Tabela 12 – Ferramentas do MORE4IoT desenvolvidas para auxiliar na construção de aplicações para IoT

Protocol Adapter	*Linguagem*
MORE4IoT Arduino SDK	C++
MORE4IoT JS SDK	JavaScript
MORE4IoT Dart SDK	Dart

Essas ferramentas implementam o componente chamado de *protocol adapter*. Dessa forma, a materialização do *protocol adapter* auxilia na construção da comunicação entre os dispositivos, as aplicações e o MORE4IoT. O SDK escrito em C++ é utilizado para construção da comunicação em dispositivos baseados em Arduino ou placas similares que utilizam a mesma linguagem de programação. O SDK escrito em JavaScript é utilizado pelos microsserviços do MORE4IoT para proporcionar a comunicação por diferentes protocolos. Por fim, o SDK escrito em Dart é para auxiliar na construção da comunicação do aplicativo construído com Flutter.

4.6 Considerações Finais

Neste capítulo, definiu-se o *middleware* MORE4IoT, desenvolvido para facilitar o desenvolvimento de aplicações para IoT. Como contribuição, os requisitos e a arquitetura do *middleware* foram definidos, os componentes da arquitetura de microsserviços foram detalhados e desenvolvidos e, por fim, os componentes para facilitar o desenvolvimento foram construídos.

Como forma de validar a infraestrutura construída pela pesquisa, o MORE4IoT foi aplicado em uma aplicação para agricultura de precisão. Com isso, o MORE4IoT e as ferramentas auxiliaram na construção da comunicação entre os dispositivos inseridos

[5] <https://github.com/iotufersa/more4iot-arduino-sdk>
[6] <https://github.com/iotufersa/more4iot-js-sdk>
[7] <https://github.com/iotufersa/more4iot-dart-sdk>

no ambiente rural. Mais detalhes da utilização das do MORE4IoT são apresentados no próximo capítulo.

5 Estudo de Caso

Neste capítulo, apresenta-se uma aplicação para agricultura de precisão que utiliza as funcionalidades e tecnologias disponibilizadas pelo MORE4IoT. Portanto, trata-se de um caso para validar e mostrar que a construção de aplicações com dispositivos que usam diferentes protocolos pode ser facilitada com o uso do MORE4IoT. As ferramentas que o *middleware* disponibiliza são fundamentais para auxiliar na construção das aplicações para IoT. Desse modo, o capítulo aborda teoria e prática da aplicação para agricultura de precisão com aprofundamento nas funcionalidades que o MORE4IoT oferece para auxiliar na construção de aplicações na *Internet of Things*.

5.1 Germina - Uma Aplicação para Agricultura de Precisão

A agropecuária usa 70% da água no país, porém quase metade desse montante é jogado fora (ANA, 2019). Entre os motivos do desperdício estão irrigações mal-executadas e falta de controle do agricultor na quantidade usada e no processamento dos produtos. Os impactos recaem sobre o ecossistema, já que lençóis freáticos e rios sofrem com a falta de chuvas e correm o risco de secar ao longo dos anos. A agricultura necessita de uma elevada quantidade de água para irrigar o cultivo. Dessa forma, evitar o desperdício pode resultar em melhores resultados para o cultivo e economia para o agricultor. No entanto, destaca-se a quantidade de água disponível superior a necessidade da planta para um determinado tamanho (idade da planta), com isso, ocasionando uma perda de água desnecessária, além do desperdício de nutrientes, energia e outros produtos. No fim, o produtor possui um produto muito caro para produzir e comercializar. Esse problema é intensificado em regiões que possuem pouca disponibilidade de água para produção, como no nordeste brasileiro.

Em cultivos controlados, que possuem um manejo adequado, nenhum problema descrito anteriormente ocorre com grande intensidade. No entanto, é importante investigar e mensurar o quanto a irrigação é eficiente em tempo e quantidade. Além disso, quando aplicada ao semiárido, os tempos e quantidade da irrigação podem variar. Dessa forma, tecnologias aplicadas à agricultura de precisão podem fornecer dados para realizar um controle avançado da irrigação, para ajudar a otimizar e reduzir o excesso de irrigação e dos produtos utilizados no cultivo (GAO, 2019).

A Agricultura de Precisão (AP) é uma estratégia de gestão que considera a variabilidade temporal e espacial para melhorar a sustentabilidade da produção agrícola (ISPA, 2021). Para isso, reúne, processa e analisa dados temporais, individuais e espaciais, combinados com outras informações para apoiar as decisões de gerenciamento, de acordo com a variabilidade estimada para melhorar a eficiência no uso de recursos, a produtividade,

a qualidade, a rentabilidade e a sustentabilidade da produção agropecuária (Grego et al., 2020).

Diante dos problemas relacionados ao desperdício de água e falta de controle foi proposto o Germina (**Ger**enciamento e **M**oni**t**oramento **n**a **A**gricultura), um sistema para auxiliar a viabilização do cultivo rural, auxiliando o produtor na criação de um histórico de produção e monitorando os custos do cultivo baseado nos itens utilizados no período de produção, a fim de reduzir o desperdício, aumentar a lucratividade e precificar seus produtos com maior precisão. Além disso, a aplicação permite: monitorar o cultivo utilizando sensores na captura de dados; inserir atuadores no controle dinâmico das ferramentas disponíveis; gerenciar e auxiliar o agricultor no processo de irrigação agrícola; e, disponibilizar relatórios quantitativos dos itens utilizados no período de irrigação. Para isso, o MORE4IoT torna-se uma ferramenta importante na construção dessa solução. Assim, o MORE4IoT traz as funções de gerenciar os sensores e os dispositivos atuadores, intermediando a comunicação entre os dispositivos e o GerminaApp. Nas próximas subseções são apresentados mais detalhes dessa aplicação utilizando o MORE4IoT.

5.2 Requisitos

São descritos um conjunto de requisitos para a aplicação Germina, baseados nas características essenciais para resolver os problemas descritos anteriormente. Com isso, aborda-se as funcionalidades necessárias para oferecer flexibilidade, histórico e organização. O agrônomo será o responsável por todas as decisões, mas terá um aplicativo para auxiliá-lo durante as etapas de produção. Os requisitos de gerenciamento e de relatórios são essências para que o agrônomo possa definir e realizar procedimentos no manejo do cultivo. Diante disso, o aplicativo do Germina deve ser responsável por guardar as informações que serão geradas pelo agrônomo e pelas informações geradas durante o monitoramento dos sensores ao longo do cultivo. Durante esse processo, diversos relatórios devem ficar disponíveis para o agrônomo, por exemplo: quantitativo de nutrientes, de água, energia e produtos utilizados. Além disso, o agrônomo poderá analisar qualitativamente as informações observadas durante o dia a dia do cultivo e inseridas no aplicativo.

5.2.1 Requisitos Funcionais

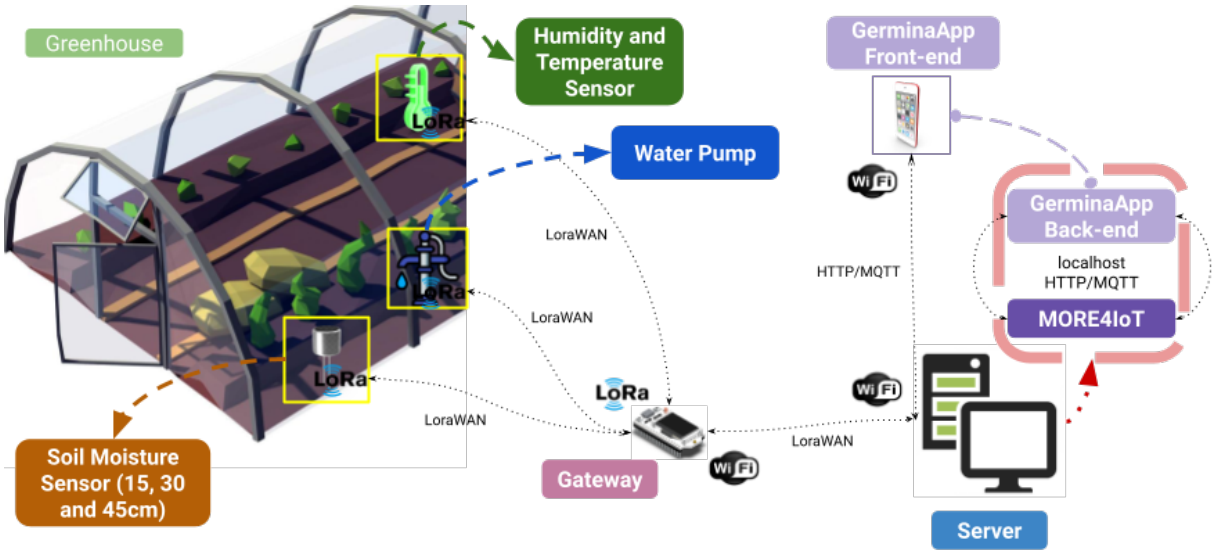
- **Gerenciamento de Cultivos:** o Germina deve fornecer formas de inserção, atualização, visualização e remoção de cultivos.
- **Gerenciamento de Nutrientes:** o Germina deve disponibilizar a inserção de nutrientes, assim como a listagem, atualização da quantidade e remoção de nutrientes.

- **Visualização dos Dispositivos:** o Germina deve disponibilizar os dispositivos para visualização dos dados gerados obtidos através da comunicação disponibilizada pelo MORE4IoT. O dispositivos devem ser acessados a partir do MORE4IoT.
- **Gerenciamento de Irrigações:** o Germina deve disponibilizar a inserção, atualização, visualização e remoção de irrigações. As irrigações devem utilizar os cultivos, os nutrientes e dispositivos disponíveis, escolhidos pelo agrônomo.
- **Geração e Visualização de Relatórios:** o Germina deve disponibilizar relatórios com as informações gerenciadas pela aplicação.

5.2.2 Arquitetura do Germina

O Germina é formado por sensores, atuadores e aplicativo (GerminaApp) para auxiliar o produtor rural no gerenciamento e monitoramento do cultivo na agricultura de precisão. Na Figura 28, apresenta-se uma visão geral do Germina utilizando o MORE4IoT como o intermediário no gerenciamento dos dispositivos inseridos nessa aplicação. Essa versão é formada pelos sensores de umidade do solo (*soil moisture sensor*), de temperatura (*temperature sensor*), umidade do ar (*humidity sensor*), um dispositivo atuador para controlar o acionamento da válvula solenoide da bomba de irrigação (*water pump*).

Figura 28 – Visão geral da aplicação Germina



Fonte: autoria própria.

Os dispositivos foram conectados a um *gateway* e mediados pelo MORE4IoT. Além disso, possui um aplicativo chamado GerminaApp (composto por *front-end* e *back-end*) para auxiliar o produtor no gerenciamento e monitoramento do manejo do cultivo, irrigação, produtos, nutrientes utilizados, geração de relatórios e notificações. O MORE4IoT é instanciado em um servidor local (*server*) para gerenciamento desses dispositivos inseridos nessa

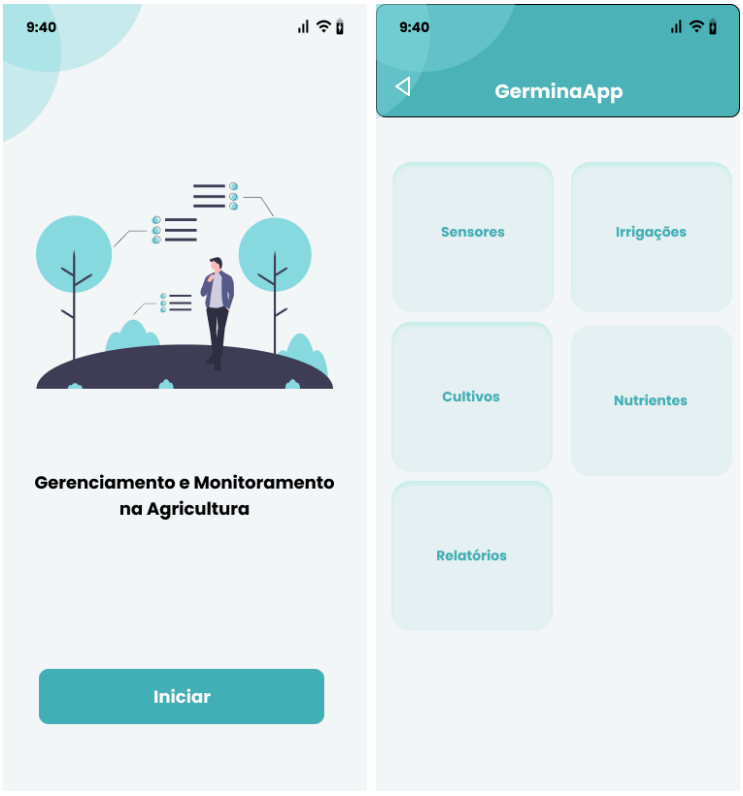
aplicação, conforme mencionado anteriormente. O MORE4IoT comunica-se localmente com o *back-end* do aplicativo GerminaApp, além de gerenciar os dispositivos LoRa. No entanto, o MORE4IoT poderia estar na nuvem. Um *gateway* LoRa permite a comunicação de longo alcance com os dispositivos com interface LoRa, principalmente para aplicações na zona rural. Todavia, outras camadas físicas poderiam ser empregadas, entre quais pode-se citar: ZigBee, GSM ou Z-Wave. A escolha da tecnologia está atrelada ao ambiente da aplicação. A comunicação entre o *gateway* e o MORE4IoT é via Wi-Fi. A comunicação com GerminaApp é com o MQTT (comunicação assíncrona) ou o HTTP (comunicação síncrona).

O MORE4IoT possui um papel fundamental no gerenciamento da comunicação entre os sensores e o aplicativo, oferecendo as ferramentas necessárias para configuração dos pacotes de dados gerados pelos sensores, comunicação por diferentes protocolos e ações de controle realizadas a partir do aplicativo. Sem o MORE4IoT, essas responsabilidades seriam da aplicação, aumentando o custo e complexidade da mesma. Mais detalhes dos componentes do Germina são apresentados nas subseções seguintes, que abordam: o aplicativo GerminaApp, os dispositivos necessários e um cenário de uso adaptado do Germina para validação do MORE4IoT no auxílio da construção do Germina.

5.2.3 GerminaApp

O GerminaApp disponibiliza a interface gráfica das funcionalidades necessárias para gerenciar e monitorar o cultivo do produtor rural, em especial o processo de irrigação, cultivos e nutrientes necessários. Na Figura 29, apresenta-se as telas principais do GerminaApp. A tela inicial é apresentada na Figura 29a. O aplicativo é constituído por cinco funcionalidades principais que podem ser acessadas pela tela principal: cultivos, irrigações, nutrientes, sensores e relatórios. A tela principal (Figura 29b) da aplicação disponibiliza os botões para cada funcionalidade. Ao clicar em algum botão, o usuário é redirecionado para tela principal de cada funcionalidade. Mais detalhes de cada funcionalidade são apresentados a seguir.

Figura 29 – Telas principais do GerminaApp. a) Tela inicial. b) Tela principal



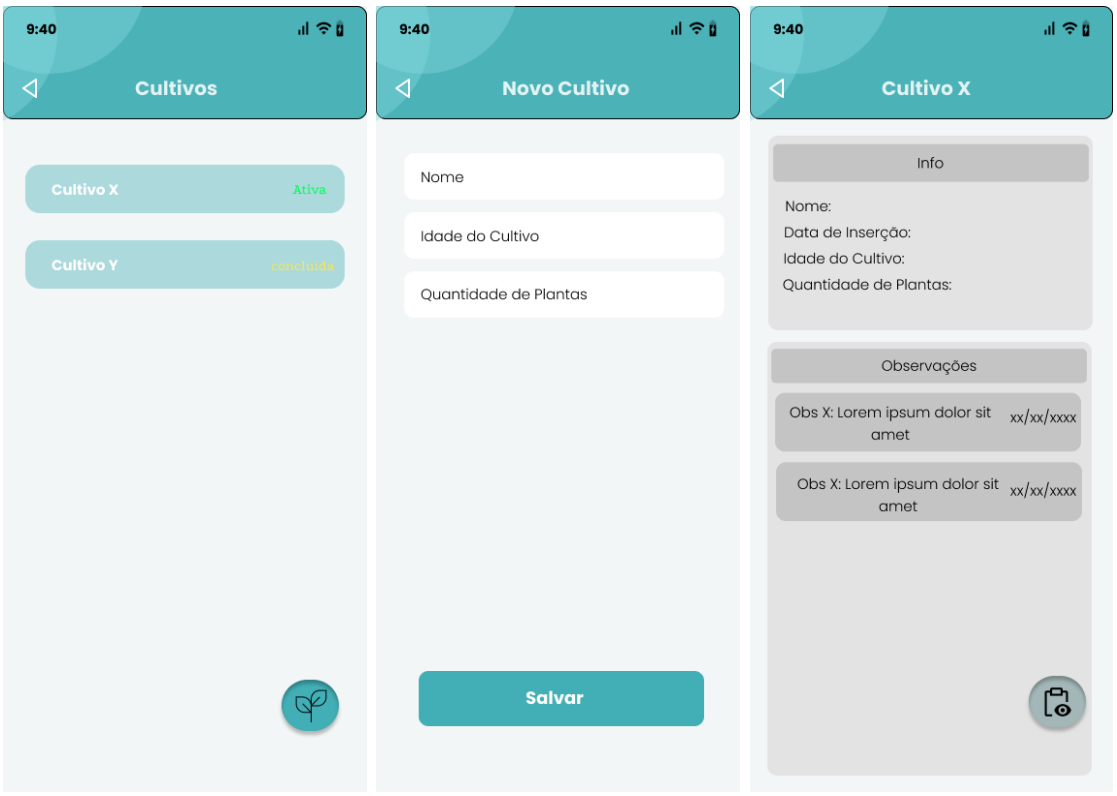
(a) (b)

Fonte: autoria própria.

A funcionalidade para cultivos é responsável por gerenciar os cultivos ativos e concluídos do produtor rural (Figura 30a). Além disso, disponibiliza-se as funcionalidades para inserir um novo cultivo (Figura 30b) e obter as informações sobre o cultivo com possibilidade de atualização (Figura 30c). Ao clicar no botão inferior direito (Figura 30a), o usuário acessa a tela para inserção de um novo cultivo. Para inserção de um novo cultivo, o usuário informa o nome, a idade do cultivo e a quantidades de plantas. Com todas as informações inseridas, o usuário pode clicar no botão salvar na mesma tela.

Depois que o usuário salva o novo cultivo, ele é redirecionado para a tela principal de cultivos. Nessa tela, ele pode clicar em qualquer um cultivo para entrar nas informações cadastradas desse cultivo. Ao clicar em um cultivo cadastrado, o usuário é redirecionado para tela de informações (Figura 30c). Nessa tela, disponibiliza-se as informações de cadastro e mais a data de inserção desse cultivo. Além disso, o usuário pode inserir observações sobre o cultivo. Essas observações servem como um histórico para futuras análises qualitativas do manejo desse cultivo. Para isso, o usuário deve clicar no botão inferior direito da tela de informações e inserir as observações, depois clicar em salvar. O aplicativo captura a data e o horário da inserção de uma nova observação.

Figura 30 – Telas das funcionalidades de cultivos a) Tela dos cultivos gerenciados. b) Tela para cadastrar um novo cultivo. c) Tela para visualizar as informações de um cultivo



(a) (b) (c)

Fonte: autoria própria.

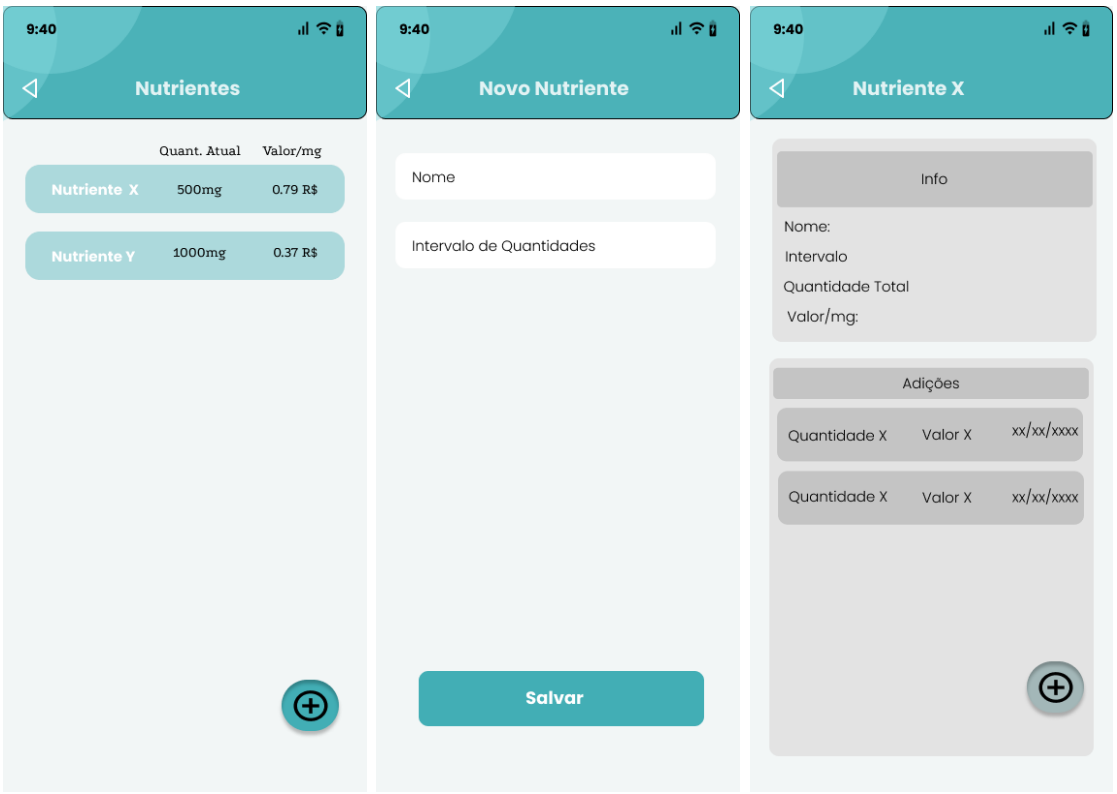
A funcionalidade de nutrientes está relacionada com os produtos utilizados durante o período de irrigação. Esses nutrientes são inseridos no processo de irrigação. Assim, para que o produtor consiga compreender os custos de produção, os nutrientes são partes fundamentais para esse gerenciamento de custos, no qual o produtor pode inserir novos nutrientes (Figura 31b) e gerenciar as quantidades de cada nutriente disponíveis (Figura 31c). Os nutrientes são utilizados durante o processo de irrigação. Eles são inseridos na água e utilizados em quantidades que podem variar por causa de alguns fatores, como: idade ou saúde da planta, por exemplo.

A tela principal de nutrientes (Figura 31a) lista todos os nutrientes que foram inseridos pelo usuário. Nessa tela, o usuário pode visualizar a quantidade atual de cada nutriente e valor pago por cada medida de peso. Além disso, disponibiliza-se mais duas funcionalidades na tela principal de nutrientes. A primeira funcionalidade é a possibilidade da inserção de novo nutriente. Para isso, o usuário deve clicar no botão inferior direito. Ao clicar, o usuário será redirecionado para tela de cadastro de um novo nutriente (Figura 31b). Nessa tela, o usuário deve informar o nome do nutriente e o intervalo de quantidades desse nutriente. Esse intervalo é utilizado durante o processo de irrigação pelo produtor rural

para identificar a quantidade a ser inserida na irrigação. Com essas informações inseridas, o usuário deve clicar em salvar e será redirecionado para tela principal de nutrientes.

Outra funcionalidade disponibilizada na tela principal de nutrientes é de visualizar as informações de cada nutriente. O usuário, ao clicar em cada nutriente será redirecionado para a tela de informações de nutrientes (Figura 31c). Nessa tela, o usuário pode identificar todas as informações nutrientes, tais como: nome, intervalo de quantidades, quantidade total desse nutriente e o valor por cada medida de peso. Além disso, o usuário pode, periodicamente, inserir novas quantidades de cada nutriente. Para isso, deve-se clicar no botão inferior direito da tela do nutriente. Com isso, um *popup* para adições das quantidades dos nutrientes é liberado. Nesses campos, o usuário pode inserir novas quantidades dos nutrientes e valor total pago por essa quantidade, depois o usuário deve salvar essa informação. Ao salvar, o usuário poderá visualizar as adições na tela de cada nutriente. Cada adição possui uma quantidade total, um valor total e uma data de inserção. Assim, o produtor rural pode gerenciar os custos desses nutrientes ao longo do manejo desse cultivo.

Figura 31 – Telas das funcionalidades dos nutrientes a) Tela dos nutrientes b) Tela para cadastrar um novo nutriente. c) Tela para visualizar as informações do nutriente



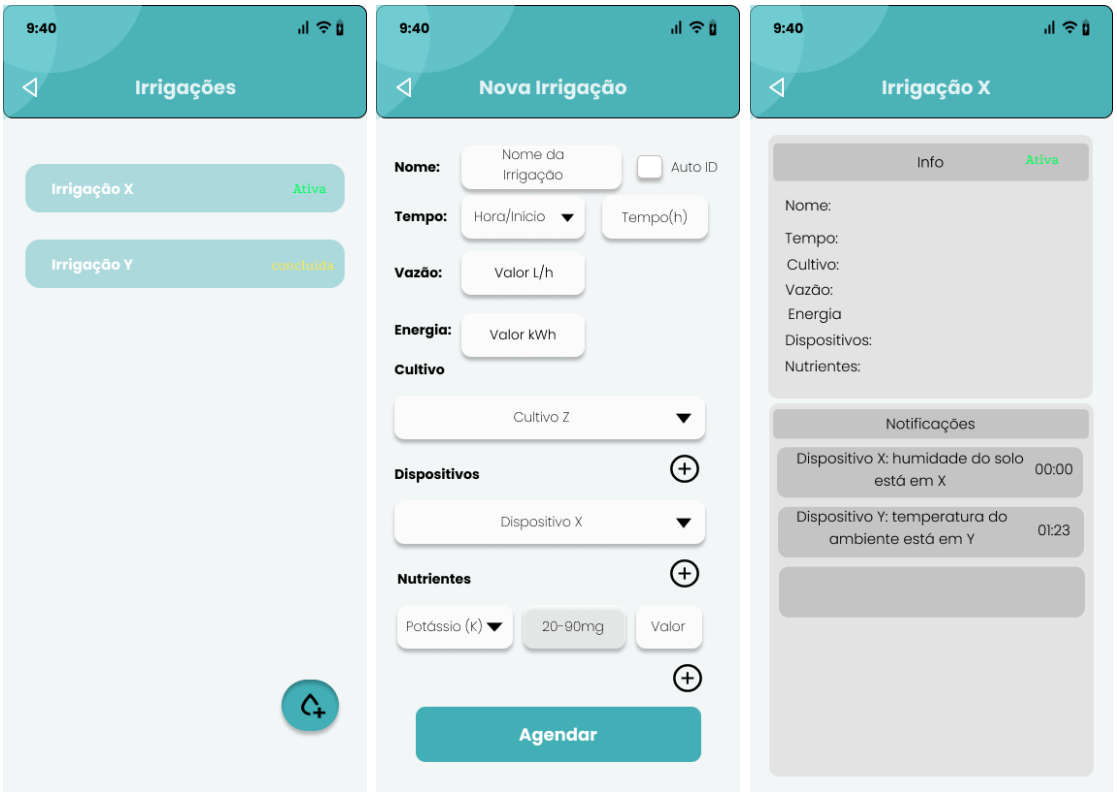
(a) (b) (c)

Fonte: autoria própria.

A funcionalidade de irrigações reúne as irrigações que estão sendo realizadas para um determinado tipo de cultivo (Figura 32a), com adição das quantidades de nutrientes

desejadas. Para isso, foram disponibilizadas as funcionalidades de inserção de uma nova irrigação com a seleção do cultivo e nutrientes desejados (Figura 32b), além da visualização e opções de observação para cada cultivo a partir da quantidade de água utilizada na irrigação (Figura 32c).

Figura 32 – Telas das funcionalidades das irrigações a) Tela das irrigações b) Tela para cadastrar uma nova irrigação. c) Tela para visualizar as informações da irrigação



(a) (b) (c)

Fonte: autoria própria.

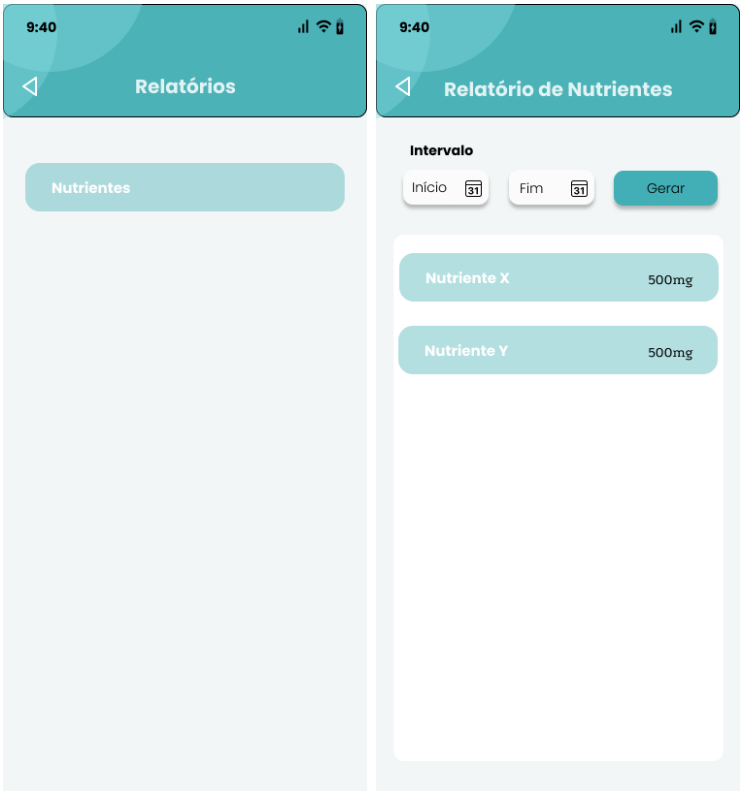
Na tela principal de irrigações (Figura 32a), o usuário pode visualizar as irrigações que estão ativas e as que estão concluídas, para um determinado período. Além disso, disponibiliza-se um botão para inserção de uma nova irrigação. Para isso, o usuário deve clicar no botão inferior direito. Assim, ele é redirecionado para a tela de nova irrigação (Figura 32b). Nessa tela, são disponibilizados os campos necessários para inserir uma nova irrigação, tais como: nome da irrigação ou *auto id*, horário de inicio, tempo total de irrigação, vazão da irrigação, custo de energia da bomba, o cultivo, os dispositivos e os nutrientes. Ao clicar em salvar, o usuário é redirecionado para tela principal de irrigação, o mesmo poderá visualizar a irrigação inserida. Além disso, o usuário pode visualizar as informações de cada irrigação. Ao clicar na irrigação, o usuário é redirecionado para a tela de informações dessa irrigação (Figura 32c). Nessa tela, aparecem as informações da irrigação. Além disso, notificações são geradas, periodicamente, devido ao funcionamento dos dispositivos nessa irrigação. Assim, tudo que está acontecendo no momento da irrigação

é notificado. O usuário pode acompanhar os dados que estão sendo gerados da irrigação com os dispositivos selecionados.

A funcionalidade de relatórios reúne e disponibiliza para o produtor todos os relatórios necessários para monitorar os custos de produção do cultivo. Com isso, outros relatórios podem ser inseridos para auxiliar o produtor. Por exemplo, pode ser inserido o relatório dos custos dos nutrientes ou dos custos das irrigações.

Na tela principal dos relatórios (Figura 33a), apresenta-se os relatórios disponíveis. O relatório para nutrientes mostra o custo de cada nutriente num intervalo de tempo selecionado. Esse relatório pode ser acessado ao clicar no item com respectivo nome do relatório. Assim, o usuário será redirecionado para a tela de informações do relatório (Figura 33b). Na tela do relatório, disponibiliza-se os campos para selecionar o intervalo de tempo: um campo para início e outro campo de fim. Ao clicar no botão "Gerar", lista-se os nutrientes e suas quantidades utilizadas para o intervalo de tempo selecionado.

Figura 33 – Telas das funcionalidades dos relatórios a) Tela dos relatórios b) Tela para visualizar as informações de um relatório



(a) (b)

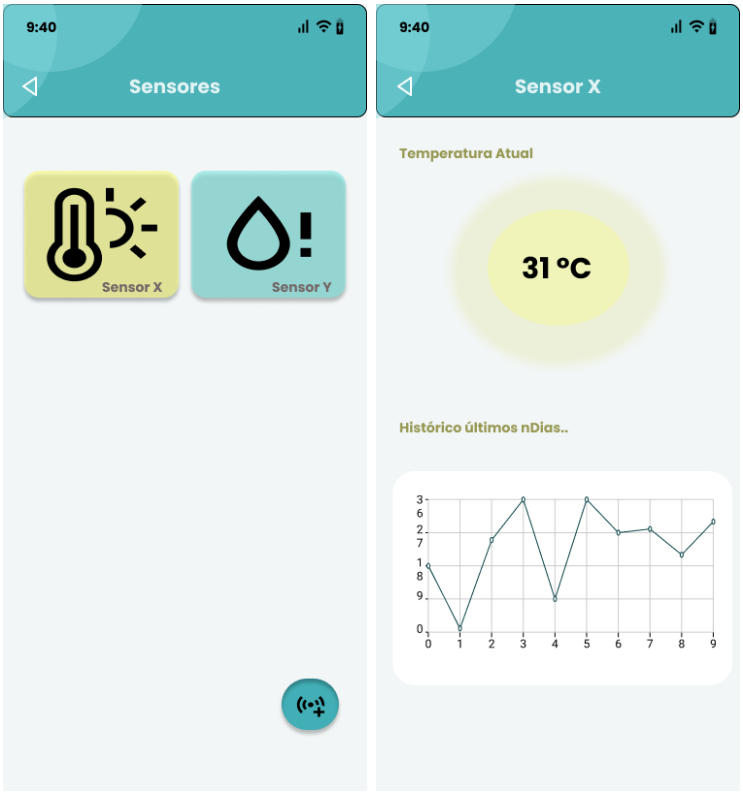
Fonte: autoria própria.

A funcionalidade de sensores é utilizada para visualizar os sensores e atuadores disponíveis para aplicação, no qual o produtor poderá visualizar os dados que são sendo gerados pelos dispositivos que estão sendo intermediados pelo MORE4IoT. Dessa forma, o

GerminaApp utiliza a ferramenta disponibilizada pelo MORE4IoT para receber os dados gerados e para controlar os dispositivos remotamente.

Na tela principal de sensores (Figura 34a), lista-se os sensores e atuadores da aplicação. Ao clicar em algum deles, o usuário será redirecionado para tela do dispositivo (Figura 34b). Nessa tela, é possível visualizar os dados que estão sendo gerados pelo sensor, em tempo real. Além disso, um histórico dos dados gerenciados também é apresentado para o usuário. Esses dados são obtidos com o intermédio do MORE4IoT. A aplicação não precisa gerenciar os dispositivos, assim, reduzindo a complexidade.

Figura 34 – Telas das funcionalidades dos sensores a) Tela dos sensores b) Tela para visualizar as informações do sensor



(a) (b)

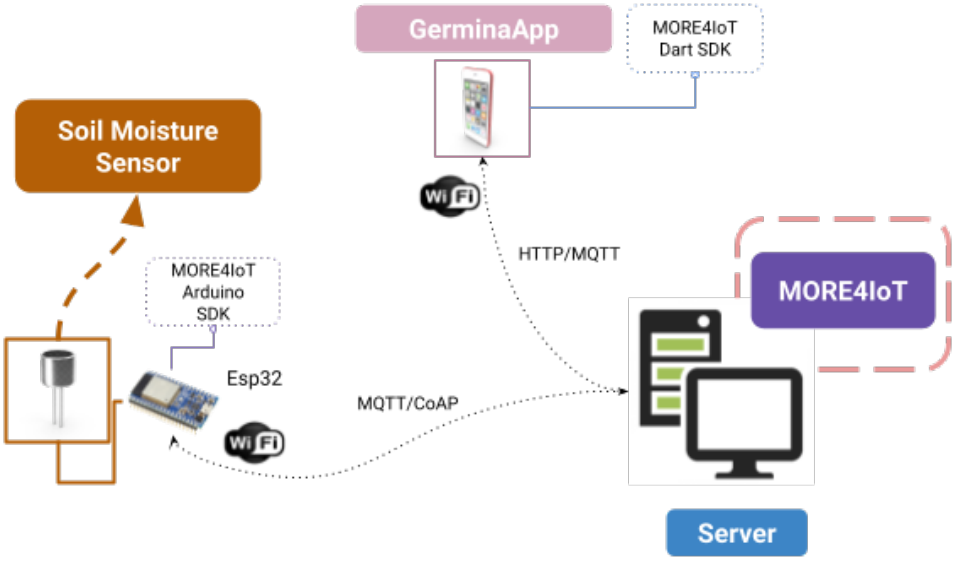
Fonte: autoria própria.

5.2.4 Aplicação Adaptada do Germina

A arquitetura do Germina, descrita anteriormente, foi adaptada para ser utilizada na validação das funcionalidades que o MORE4IoT oferece. Essa adaptação foi necessária devido a falta de recurso para construção da aplicação completa. Todavia, as funcionalidades para validação do MORE4IoT não foram afetadas, pois as partes necessárias foram utilizadas e mostradas. Nessa aplicação, foram utilizados os protocolos MQTT, CoAP e HTTP. A visão geral da arquitetura foi remodelada para os dispositivos disponíveis, conforme apresentado na Figura 35. Mesmo assim, um dispositivo (Esp32), um sensor

de umidade do solo (*soil moisture sensor*) e um atuador para ativar/desativar a válvula solenoide do sistema de irrigação (mapeado por um LED azul contido no Esp32) foram utilizados nesse cenário com o MORE4IoT. O MORE4IoT foi executado em um servidor local e uma versão reduzida do GerminaApp foi utilizada nesse cenário. A seguir é apresentada e explicada a construção da aplicação dessa arquitetura adaptada com auxílio de imagens de trechos do código e do funcionamento da aplicação.

Figura 35 – Adaptação da aplicação Germina



Fonte: autoria própria.

O Esp32 foi configurado utilizando o MORE4IoT Arduino SDK. Esse SDK é fundamental para oferecer as funcionalidades necessárias na construção e configuração desse dispositivo. Sem ele, o desenvolvedor da aplicação precisaria entender as configurações, entender os protocolos de comunicação e realizar a construção do pacote de dados manualmente. Nas Figuras 36 e 37, mostra-se a construção de envio de dados na utilização do MORE4IoT e de um outro *middleware* (ThingSpeak[1]) por meio de alguns trechos do código necessário para mostrar a diferença das duas abordagens. Na Figura 36, configura-se o Esp32 utilizando o MORE4IoT. Na Figura 37, utiliza-se um outro *middleware* para configuração.

Para o exemplo, por falta de recurso, apenas um Esp32 foi utilizado. Devido a isso, o Esp32 foi configurado para enviar os dados utilizando o MQTT e também para receber comandos pelo CoAP. O Esp32 foi configurado para enviar um dado sobre umidade do solo capturado pelo sensor. Na construção (como mencionado anteriormente) o protocolo de comunicação utilizado com o MORE4IoT é o MQTT. Todavia, poderia ser outro protocolo – o MORE4IoT disponibiliza outros exemplos com outros protocolos

[1] <https://thingspeak.com/>

no repositório do MORE4IoT Arduino SDK[2]. O ThingSpeak[3] apenas disponibiliza um exemplo de comunicação utilizando o HTTP. Para o MORE4IoT, isso não é um problema, pois ele também pode prover a comunicação utilizando o HTTP.

Mostra-se um comparativo entre as duas abordagens para facilitar a construção de aplicações na IoT (Tabela 13). Como mencionado anteriormente, o MORE4IoT reduz a complexidade no desenvolvimento. Dessa forma, os desenvolvedores apenas necessitam construir as regras de negócio, pois a comunicação é gerenciada pelo MORE4IoT.

Tabela 13 – Comparação entre middlewares em relação ao desenvolvimento no dispositivo

Característica/*Middleware*	MORE4IoT	ThingSpeak
Fornece componente para o dispositivo	✓	✗
Simplifica a utilização do protocolo	✓	✗
Auxilia na construção das regras de controle	✓	✗
Tamanho do código	simplificado	extenso

Como resultado, é perceptível a redução da complexidade do código utilizando MORE4IoT. A responsabilidade de resolver a construção do JSON *data packet* é totalmente do MORE4IoT (linhas 32-34 da Figura 36), diferentemente do outro *middleware*, que necessitou construir o envio dos dados manualmente (linhas 39-55 da Figura 37).

Em complemento, o MORE4IoT disponibiliza a adição de novos campos dinamicamente. Além disso, o desenvolvedor não precisa realizar nenhum tipo de configuração de envio pelo protocolo selecionado, o MORE4IoT abstrai os mecanismos de envio. Assim, apenas é necessário chamar o método *send* do MORE4IoT para realizar o envio do *data packet* para o MORE4IoT no servidor. Da mesma forma, ocorre com os outros protocolos disponibilizados pelo MORE4IoT. O MORE4IoT disponibiliza uma interface única para configuração dos dispositivos por diferentes protocolos. As diferenças estão encapsuladas em cada instância. Dessa forma, o desenvolvedor necessitará apenas entender a interface geral e informar a camada de transporte, o endereço IP e a porta do protocolo, conforme pode ser visualizado nas linhas 6-11 da Figura 36, como também nas linhas 6-11 da Figura 38.

O dispositivo também foi configurado para ser controlado remotamente pela aplicação. Dessa forma, o MORE4IoT disponibiliza os mecanismos de controle para construção das regras de controle no dispositivo atuador. Na aplicação do Germina, existe um atuador para ativar ou desativar a válvula solenoide da bomba d'água do sistema de irrigação. Na Figura 38, apresenta-se o trecho do código utilizando o MORE4IoT com o CoAP para auxiliar na construção do controle do dispositivo, no caso, para controlar a *water pump*.

[2]<https://github.com/iotufersa/more4iot-arduino-sdk/tree/main/examples>
[3]<https://www.mathworks.com/help/thingspeak/MoistureMonitor.html>

O MORE4IoT necessita receber uma *callback* (linha 28 da Figura 38) para ser utilizada durante o recebimento de algum pacote de dados. O MORE4IoT oferece os mecanismos de tradução do pacote para o tipo de dado desejado (linhas 17-19 da Figura 38). Com isso, a aplicação pode utilizar esses dados nas suas regras de controle.

Figura 36 – Exemplo de código utilizando o MORE4IoT para envio de dados pelo dispositivo

```
1  #include "more4iot.h"
2
3  // device default config
4
5  // MORE4IoT Config
6  IPAddress more4iotIp(MORE4IOT_IP);
7  int more4iotMqttPort = MQTT_PORT;
8
9  WiFiClient tcpClient;
10 More4iot more4iot =
11   More4iotMqtt(tcpClient, more4iotIp, more4iotMqttPort);
12 // ---- //
13
14 void setup()
15 {
16   // pin mode config
17   // wifi config
18   // more4iot connect
19   more4iot.connect();
20 }
21
22 void loop()
23 {
24   // wifi connection verify
25   // more4iot connection verify
26   if(!more4iot.connected()){
27     more4iot.connect();
28     return;
29   }
30   // soil moisture calc
31   // create data packet with uuid, soil moisture data and send
32   more4iot.newDataPacket(UUID);
33   more4iot.addField("soil-moisture",soilMoisture);
34   more4iot.send();
35   // delay time
36 }
```

Fonte: autoria própria.

Figura 37 – Exemplo de código utilizando outro *middleware* para envio de dados

```
1
2  // device default config
3
4  // ThingSpeak information.
5  // To update more fields, increase this number and add a field label below.
6  #define NUM_FIELDS 2
7  // ThingSpeak field for soil moisture measurement.
8  #define SOIL_MOISTURE_FIELD 1
9  // ThingSpeak field for elapsed time from startup.
10 #define ELAPSED_TIME_FIELD 2
11 #define THING_SPEAK_ADDRESS "api.thingspeak.com"
12 // Change this to the write API key for your channel.
13 String writeAPIKey= "XXXXXXXXXXXXXXXX";
14
15 WiFiClient client;
16
17 void setup()
18 {
19   // pin mode config
20   // wifi config
21 }
22
23 void loop()
24 {
25   // wifi connection verify
26   // Write to successive fields in your channel by filling fieldData with up to 8 values.
27   String fieldData[ NUM_FIELDS ];
28   // You can write to multiple fields by storing data in the fieldData[] array, and changing numFields.
29   // Write the moisture data to field 1.
30   fieldData[ SOIL_MOISTURE_FIELD ] = String( readSoil( numMeasure ) );
31   // Write the elapsed time from startup to Field 2.
32   fieldData[ ELAPSED_TIME_FIELD ] = String( millis() );
33   HTTPPost( NUM_FIELDS , fieldData );
34   delay( 1000 );
35   Serial.print( "Goodnight for "+String( SLEEP_TIME_SECONDS ) + " Seconds" );
36   // delay time
37 }
38
39 int HTTPPost( int numFields , String fieldData[] ){
40   if (client.connect( THING_SPEAK_ADDRESS , 80 )){
41       // Build the postData string.
42       // If you have multiple fields, make sure the sting does not exceed 1440 characters.
43       String postData= "api_key=" + writeAPIKey ;
44       for ( int fieldNumber = 1; fieldNumber < numFields+1; fieldNumber++ ){
45           String fieldName = "field" + String( fieldNumber );
46           postData += "&" + fieldName + "=" + fieldData[ fieldNumber ];
47         }
48       client.println( "POST /update HTTP/1.1" );
49       client.println( "Host: api.thingspeak.com" );
50       client.println( "Connection: close" );
51       client.println( "Content-Type: application/x-www-form-urlencoded" );
52       client.println( "Content-Length: " + String( postData.length() ) );
53       client.println();
54       client.println( postData );
55       String answer = getResponse();
56   }
57 }
58
59 String getResponse(){
60   String response;
61   long startTime = millis();
62   delay( 200 );
63   while ( client.available() < 1 && (( millis() - startTime ) < TIMEOUT ) ){
64     delay( 5 );
65   }
66   if( client.available() > 0 ){ // Get response from server.
67     char charIn;
68     do {
69       // Read a char from the buffer.
70       charIn = client.read();
71       // Append the char to the string response.
72       response += charIn;
73     } while ( client.available() > 0 );
74   }
75   client.stop();
76   return response;
77 }
```

Fonte: adaptado de ThingSpeak (2021)

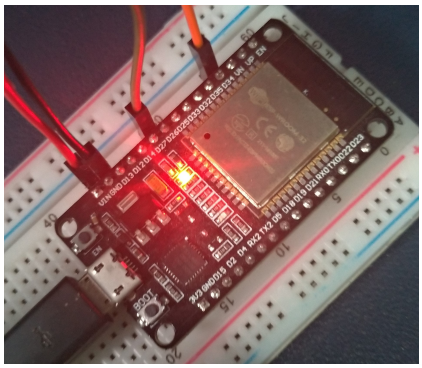
Figura 38 – Exemplo de código utilizando o MORE4IoT para auxiliar no controle do dispositivo

```ino
#include "more4iot.h"

// device default config

// MORE4IoT Config
IPAddress more4iotIp(MORE4IOT_IP);
int more4iotCoapPort = COAP_PORT;

WiFiUDP udpClient;
More4iot more4iot =
  More4iotCoap(udpClient, more4iotIp, more4iotCoapPort);
// ---- //

// callback for water-pump
void waterPump(CoapPacket &packet, IPAddress ip, int port) {
  // get dry soil value from scope
  drySoil = more4iot.getCommand<int>(packet.payload, "dry-soil");
  // get soil moisture value from scope
  soilMoisture = more4iot.getData<int>(packet.payload, "soil-moisture");
  // water pump on / off
}

void setup()
{
  // pin mode config
  // wifi config
  // more4iot connect and callback
  more4iot.callback(waterPump, "soil-moisture");
  more4iot.connect();
}

void loop()
{
  // wifi connection verify
  // more4iot packet verify
  more4iot.loop();
  // delay time
}
```

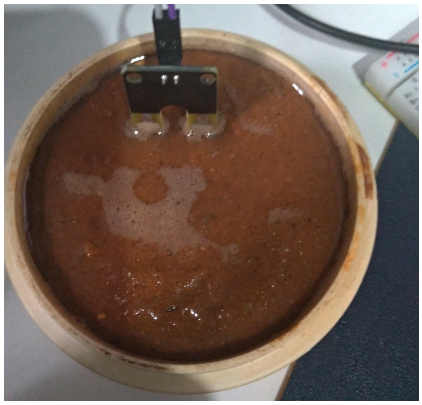
Fonte: autoria própria.

Na Figura 39, apresenta-se o resultado da configuração do dispositivo para envio do dados produzidos pelo sensor de umidade do solo (*soil moisture sensor*) com o MORE4IoT Arduino SDK utilizando os códigos apresentados anteriormente (Figuras 36 e 38). No dispositivo, os dados do sensor estão sendo enviados para o MORE4IoT que distribui para os recursos da aplicação. As mesmas configurações são realizadas no GerminaApp para receber os dados que estão sendo intermediados pelo MORE4IoT. No GerminaApp é utilizado o MORE4IoT Dart SDK (*protocol adapter*) para disponibilizar a utilização do MQTT na construção da comunicação no aplicativo com o MORE4IoT. O Esp32 envia o *data packet* para o MORE4IoT. O MORE4IoT distribui para os recursos, no caso, o GerminaApp e dispositivo atuador do *water pump*. Nesse primeiro cenário, o dispositivo mantém a *water pump* desligada (representado por um LED azul desligado do Esp32 na Figura 39a), pelo fato da umidade do solo está acima da desejada (Figura 39b). Da mesma forma, o GerminaApp mostra a umidade do solo atual (*soil moisture* 59% na Figura 39c).

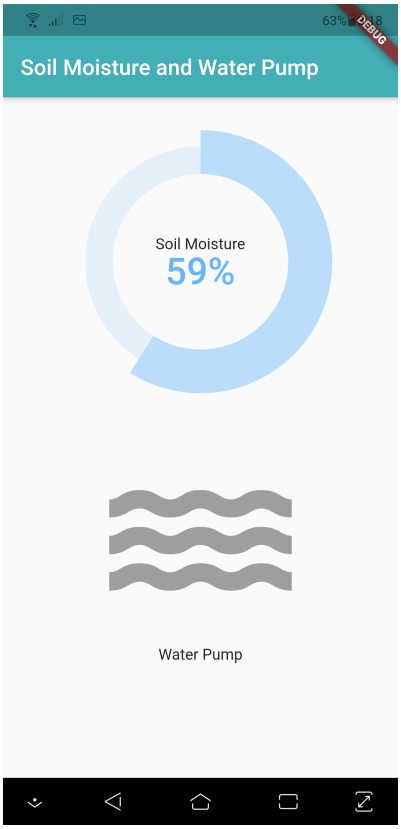
Figura 39 – Estado da aplicação quando o solo está úmido. a) Esp32 com *water-pump* desligada. b) Sensor com solo úmido. c) Tela do GerminaApp mostrando a umidade do solo



(a)



(b)



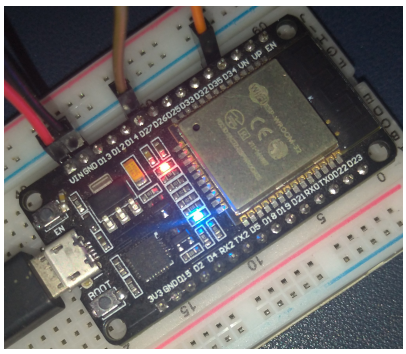
(c)

Fonte: autoria própria.

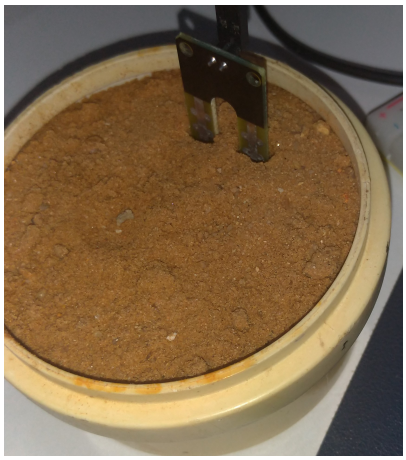
Em um segundo cenário (Figura 40), quando o solo está seco ou com umidade inferior a desejada (Figura 40b), utilizando o MORE4IoT para receber os dados na

construção das regras de controle do dispositivo (*water pump*). Assim, o ato de ligar e desligar a válvula solenoide é mapeado por um LED azul no Esp32 (Figura 40a). Dessa forma, a *water pump* está ligada, conforme mostrado no GerminaApp com umidade do solo em 0% e *water pump* ligada (representada por um ícone em azul na Figura 40c). As regras de controle são definidas pelas aplicações. Observa-se que o MORE4IoT auxilia na transmissão e gerenciamento dos recursos da aplicação provendo os mecanismos necessários, conforme visto nessa subseção.

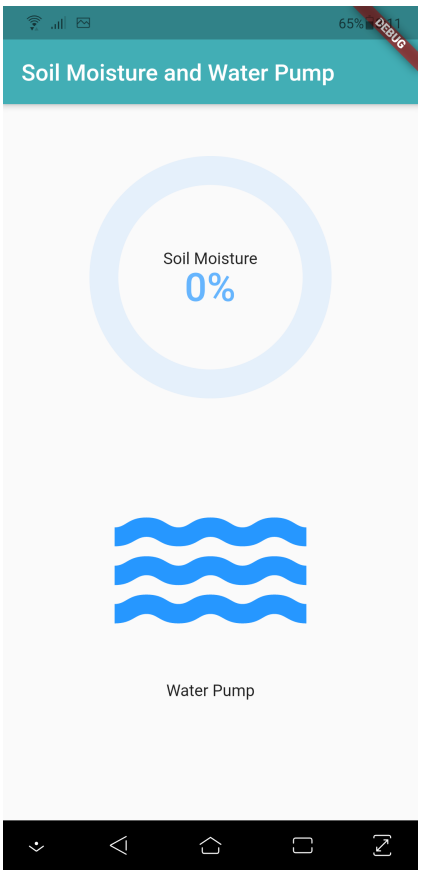
Figura 40 – Estado da aplicação quando o solo está com baixa umidade. a) Esp32 com *water-pump* ligada. b) Sensor e solo com baixa umidade c) Tela do GerminaApp mostrando a umidade do solo e a *water-pump* ligada



(a)



(b)



(c)

Fonte: autoria própria.

5.3 Considerações Finais

Neste capítulo, apresentou-se o Germina, uma aplicação para agricultura de precisão definida para auxiliar o agrônomo no manejo do cultivo. Para isso, foram inseridos dispositivos para realizar capturas dados do cultivo, em tempo real. Com isso, o MORE4IoT foi utilizado para gerenciar e auxiliar na comunicação dos dispositivos inseridos no ambiente da aplicação, facilitando a construção e gerenciamento dos dispositivos. Além disso, o

middleware foi comparado com outras soluções.

Como resultado, observou-se que o MORE4IoT solucionou alguns problemas relacionados a construção de uma aplicação IoT. O MORE4IoT forneceu componentes que foram inseridos nos dispositivos para construir a comunicação. Além disso, simplificou a utilização dos protocolos de comunicação nos dispositivos e no aplicativo do Germina. Em complemento, auxiliou na construção das regras de controle dos dispositivos e reduziu a complexidade de código para o desenvolvedor. Por fim, solucionou o gerenciamento da comunicação e distribuição dos dados gerados pelas partes da aplicação.

6 Conclusão

Este trabalho foi iniciado a partir de uma revisão bibliográfica com o objetivo de reunir e capturar informações relevantes sobre *middleware* para IoT. Ao término da revisão bibliográfica, um protocolo da revisão sistemática foi definido para organizar o estado da arte sobre *middleware* usado em aplicações na internet das coisas, bem como protocolos de comunicação utilizados, as arquiteturas e requisitos. Com o protocolo definido, a revisão sistemática foi conduzida. Como resultado, observou-se que a maioria dos trabalhos propuseram agregar interoperabilidade às suas soluções. No entanto, houve uma redução da capacidade de comunicação entre os dispositivos. Esses trabalhos não agregaram diversos protocolos para comunicação entre *middleware*, dispositivos e aplicações. Assim, a capacidade de interoperar ficou comprometida. Além disso, não definiram mecanismos padronizados para comunicação e adaptação das formas de comunicação por diferentes protocolos.

Dessa forma, este trabalho definiu uma arquitetura baseada em microsserviços e construiu um *middleware* multiprotocolo para auxiliar na construção de aplicações para internet das coisas, provendo interoperabilidade com mecanismos de comunicação bem definidos. Em complemento, o MORE4IoT reúne requisitos de escalabilidade, flexibilidade, aspectos de segurança e privacidade. Além disso, outros requisitos importantes foram agregados ao *middleware* para enfrentar os desafios de um *middleware* multiprotocolo na internet das coisas. Entre os requisitos, pode-se destacar o gerenciamento de recursos, gerenciamento de dados, mecanismos de controle, disponibilização de funcionalidades e ferramentas para auxiliar no desenvolvimento das aplicações.

Além disso, um estudo de caso com o objetivo de validar e utilizar o MORE4IoT como mecanismo auxiliar na construção da aplicação foi definido e desenvolvido. Dessa forma, as ferramentas construídas para facilitar a construção das aplicações para IoT foram expostas e exemplificadas durante a construção da aplicação Germina.

Por fim, destaca-se que as questões de pesquisa levantadas no Capítulo 1 foram respondidas ao longo do texto. A questão QC1 foi respondida com a identificação dos middlewares utilizados em aplicações na IoT. A questão QC1.1 foi respondida com a identificação dos requisitos para *middleware* IoT e a definição dos requisitos do *middleware* proposto. A questão QC1.2 foi respondida com a identificação dos protocolos de comunicação utilizados por middlewares IoT. A questão QC1.3 foi respondida com a definição da arquitetura do MORE4IoT baseada nas arquiteturas de middlewares na IoT, nos desafios do *middleware* multiprotocolo e nas lacunas de pesquisa. As questões QE(1,2,3), QPS1 e QSR1 foram respondidas com a definição e implementação do MORE4IoT. A questão

QT1 foi respondida a partir da definição do componente do MORE4IoT que auxilia na construção da aplicação no lado do dispositivo.

As contribuições deste trabalho são descritas a seguir, assim como os trabalhos futuros.

6.1 Contribuições do Trabalho

A principal contribuição deste trabalho foi construir um *middleware* multiprotocolo para facilitar a construção de aplicações que usam e conectam dispositivos com protocolos de comunicação diferentes na internet das coisas, por meio de mecanismos bem definidos de comunicação. Além disso, pode-se adicionar as seguintes contribuições:

- Definição e condução de uma RSL, que reúne os middlewares mais recentes usados em aplicações na IoT, além da identificação dos requisitos, das arquiteturas e dos protocolos de comunicação utilizados.
- Definição de uma arquitetura de *middleware* multiprotocolo baseada em microsserviços capaz de gerenciar diferentes dispositivos inseridos na internet das coisas.
- Implementação de um *middleware* multiprotocolo que promove a comunicação por diferentes protocolos de comunicação com a definição e disponibilização de mecanismos bem definidos de comunicação para auxiliar na construção das aplicações na IoT, que usam e conectam dispositivos com protocolos de comunicação diferentes.
- Desenvolvimento de uma aplicação para Agricultura de Precisão que usa e conecta dispositivos por protocolos de comunicação diferentes.

6.2 Trabalhos Futuros

Como possibilidade de trabalhos futuros decorrentes deste trabalho, é possível citar:

- Pesquisar e adaptar o MORE4IoT para gerenciar dispositivos que geram áudio e vídeo utilizando protocolos específicos para transmissão e disponibilização em tempo real, como o RTP ou versões adaptadas desse protocolo (IoT-RTP e IoT-RTCP), como proposto por Said et al. (2017). O CoAP foi avaliado para esse fim, mas necessitaria de algumas configurações para melhorar o processo de transmissão sem perdas consideradas, segundo Rahman, Choi e Chung (2019).
- Pesquisar e agregar novos requisitos não funcionais e/ou funcionais para o MORE4IoT. Por exemplo, adicionar novos mecanismos de segurança e privacidade.

- Investigar e avaliar a utilização do Node-RED com MORE4IoT, principalmente na utilização da interface gráfica. Assim, por exemplo, as aplicações poderiam arrastar e interligar fluxos na criação de novas ações de controle, a partir de *nodes* do MORE4IoT para o Node-RED. Além disso, agregando as funcionalidades providas pelo MORE4IoT, sobretudo para comunicação do lado do cliente, que não são disponibilizadas pelo Node-RED.

6.3 Produções Científicas

Os resultados obtidos ao longo deste trabalho foram submetidos nos seguintes periódicos e conferências:

- – Título: *Middleware for the Internet of Things: a systematic literature review.*
  – Veículo: *Journal of Universal Computer Science.*
  – Situação: submetido.
- – Título: MORE4IoT – *Multiprotocol Middleware for the Internet of Things.*
  – Veículo: a ser definido.
  – Situação: a ser submetido.

Referências

ABBOTT, M. L.; FISHER, M. T. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise*. 2nd. ed. [S.l.]: Addison-Wesley Professional, 2015. ISBN 0134032802. Citado na página 50.

ABDI. *Indústria 4.0*. 2020. Acessado: 26-02-2020. Disponível em: <http://www.industria40.gov.br/>. Citado na página 15.

ABINC. *Agricultura de precisão com a Internet das Coisas*. 2020. Acessado: 03-06-2021. Disponível em: <https://abinc.org.br/agricultura-de-precisao-com-a-internet-das-coisas/>. Citado na página 15.

ABOWD, G. D.; DEY, A. K.; BROWN, P. J.; DAVIES, N.; SMITH, M.; STEGGLES, P. Towards a better understanding of context and context-awareness. In: *Proc. 1st Int. Symp. Handheld Ubiquitous Computing*. Berlin, Heidelberg: Springer-Verlag, 1999. (HUC '99, v. 1707), p. 304–307. ISBN 3540665501. Citado na página 51.

ALLIANCE, A. *The Agile Practice Guide*. Newtown Square, Pennsylvania: The Project Management Institute, 2017. ISBN 978-1-62825-199-9. Citado 2 vezes nas páginas 24 e 26.

ALVARO, L. B.; MIGUEL, P.; MOLINA, J. Thinger.io: An Open Source Platform for Deploying Data Fusion Applications in IoT Environments. *Sensors*, v. 19, n. 5, p. 1044, mar 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/5/1044>. Citado 2 vezes nas páginas 39 e 45.

ALVISI, S.; CASELLATO, F.; FRANCHINI, M.; GOVONI, M.; LUCIANI, C.; POLTRONIERI, F.; RIBERTO, G.; STEFANELLI, C.; TORTONESI, M. Wireless Middleware Solutions for Smart Water Metering. *Sensors*, v. 19, n. 8, p. 1853, apr 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/8/1853>. Citado 2 vezes nas páginas 39 e 45.

ANA. *Quase metade da água usada na agricultura é desperdiçada — Agência Nacional de Águas e Saneamento Básico*. 2019. <https://www.ana.gov.br/noticias-antigas/quase-metade-da-a-gua-usada-na-agricultura-a-c.2019-03-15.2354987174>. (Acessado on 07/07/2021). Citado na página 83.

ANTONIĆ, A.; MARJANOVIĆ, M.; PRIPUŽIĆ, K.; Podnar Žarko, I. A mobile crowd sensing ecosystem enabled by CUPUS: Cloud-based publish/subscribe middleware for the Internet of Things. *Future Generation Computer Systems*, v. 56, p. 607–622, mar 2016. ISSN 0167739X. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0167739X15002575>. Citado 2 vezes nas páginas 39 e 43.

BAJPAI, V.; GORTHI, R. On non-functional requirements: A survey. In: . [S.l.: s.n.], 2012. p. 1–4. ISBN 978-1-4673-1516-6. Citado na página 51.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, v. 33, n. 3, p. 42–52,

May 2016. ISSN 1937-4194. Disponível em: <https://doi.org/10.1109/MS.2016.64>. Citado na página 58.

BANDYOPADHYAY, S.; SENGUPTA, M.; MAITI, S.; DUTTA, S. Role of middleware for internet of things. *International Journal of Computer Science Engineering Survey (IJCSES)*, v. 2, n. 3, p. 94–105, 2011. Citado na página 16.

BENAYACHE, A.; BILAMI, A.; BARKAT, S.; LORENZ, P.; TALEB, H. MsM: A microservice middleware for smart WSN-based IoT application. *Journal of Network and Computer Applications*, Elsevier Ltd, v. 144, n. October 2018, p. 138–154, 2019. ISSN 10958592. Disponível em: <https://doi.org/10.1016/j.jnca.2019.06.015>. Citado 3 vezes nas páginas 39, 46 e 51.

BIOLCHINI, J.; MIAN, P. G.; NATALI, A. C. C.; TRAVASSOS, G. H. *Systematic review in software engineering*. Systems Engineering and Computer Science Department, UFRJ, RJ, 2005. Citado na página 25.

BIOLCHINI, J. C. de A.; MIAN, P. G.; NATALI, A. C. C.; CONTE, T. U.; TRAVASSOS, G. H. Scientific research ontology to support systematic review in software engineering. *Adv. Eng. Inform.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 21, n. 2, p. 133–151, abr. 2007. ISSN 1474-0346. Disponível em: <http://dx.doi.org/10.1016/j.aei.2006.11.006>. Citado na página 25.

BOULOUKAKIS, G.; GEORGANTAS, N.; NTUMBA, P.; ISSARNY, V. Automated synthesis of mediators for middleware-layer protocol interoperability in the IoT. *Future Generation Computer Systems*, v. 101, p. 1271–1294, dec 2019. ISSN 0167739X. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0167739X18323586>. Citado 2 vezes nas páginas 39 e 46.

BRERETON, P.; KITCHENHAM, B. A.; BUDGEN, D.; TURNER, M.; KHALIL, M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, v. 80, n. 4, p. 571–583, 2007. ISSN 0164-1212. Software Performance. Disponível em: <https://doi.org/10.1016/j.jss.2006.07.009>. Citado na página 24.

BUNZ, M.; MEIKLE, G. *The Internet of Things*. Medford, MA, USA: Polity Press, 2017. ISBN 978-1-5095-1746-6. Citado na página 16.

BURNS, B. *Designing distributed systems: patterns and paradigms for scalable, reliable services*. First edition. Sebastopol, CA: O'Reilly Media, 2018. ISBN 9781491983645. Citado na página 62.

CHAQFEH, M. A.; MOHAMED, N. Challenges in middleware solutions for the internet of things. *International Conference on Collaboration Technologies and Systems (CTS)*, p. 21–26, 2012. Citado na página 17.

CRUZ, M. A. A. da; RODRIGUES, J. J. P. C.; LORENZ, P.; KOROTAEV, V. V.; ALBUQUERQUE, V. H. C. de. In.iot—a new middleware for internet of things. *IEEE Internet of Things Journal*, v. 8, n. 10, p. 7902–7911, 2021. Citado 2 vezes nas páginas 39 e 43.

CRUZ, M. A. da; RODRIGUES, J. J.; LORENZ, P.; SOLIC, P.; AL-MUHTADI, J.; ALBUQUERQUE, V. H. C. A proposal for bridging application layer protocols to HTTP on IoT solutions. *Future Generation Computer Systems*, Elsevier B.V., v. 97, n. 2019, p. 145–152, 2019. ISSN 0167739X. Disponível em: <https://doi.org/10.1016/j.future.2019.02.009>. Citado 2 vezes nas páginas 39 e 44.

DIPSIS, N.; STATHIS, K. A restful middleware for ai controlled sensors, actuators and smart devices. *Journal of Ambient Intelligence and Humanized Computing*, v. 11, n. 7, p. 2963–2986, Jul 2020. ISSN 1868-5145. Disponível em: <https://doi.org/10.1007/s12652-019-01439-3>. Citado 2 vezes nas páginas 39 e 47.

DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. Microservices: Yesterday, today, and tomorrow. In: _____. *Present and Ulterior Software Engineering*. Cham: Springer International Publishing, 2017. p. 195–216. ISBN 978-3-319-67425-4. Disponível em: <https://doi.org/10.1007/978-3-319-67425-4_12>. Citado 2 vezes nas páginas 58 e 60.

ECK, N. J. van; WALTMAN, L. Software survey: VOSviewer, a computer program for bibliometric mapping. v. 84, n. 2, p. 523–538, 2010. ISSN 1588-2861. Disponível em: <https://doi.org/10.1007/s11192-009-0146-3>. Citado na página 36.

ESPOSTE, A. d. M. D.; SANTANA, E. F.; KANASHIRO, L.; COSTA, F. M.; BRAGHETTO, K. R.; LAGO, N.; KON, F. Design and evaluation of a scalable smart city software platform with large-scale simulations. *Future Generation Computer Systems*, Elsevier B.V., v. 93, p. 427–441, apr 2019. ISSN 0167-739X. Disponível em: <https://doi.org/10.1016/j.future.2018.10.026>. Citado 2 vezes nas páginas 39 e 44.

FARAHZADI, A.; SHAMS, P.; REZAZADEH, J.; FARAHBAKHSH, R. Middleware technologies for cloud of things: a survey. *Digital Communications and Networks*, v. 4, n. 3, p. 176 – 188, 2018. ISSN 2352-8648. Disponível em: <https://doi.org/10.1016/j.dcan.2017.04.005>. Citado 2 vezes nas páginas 53 e 54.

FELIZARDO, K. R.; NAKAGAWA, E. Y.; FABBRI, S. C. P. F.; FERRARI, F. C. *Revisão sistemática da literatura em engenharia de software: teoria e prática*. Rio de Janeiro, RJ: Elsevier, 2017. v. 1. 144 p. ISBN 978-85-352-8641-0. Citado na página 25.

FOWLER, M.; LEWIS, J. *Microservices: a definition of this new architectural term*. 2014. Acessado: 11/11/2019. Disponível em: <https://www.martinfowler.com/articles/microservices.html>. Citado 3 vezes nas páginas 58, 59 e 60.

GAO. *Irrigated Agriculture: Technologies, Practices, and Implications for Water Scarcity*. 2019. <https://www.gao.gov/products/gao-20-128sp>. (Acessado on 07/07/2021). Citado na página 83.

GEORGI, N.; CORVOL, A.; JEANNES, R. L. B. Middleware Architecture for Health Sensors Interoperability. *IEEE Access*, v. 6, p. 26283–26291, 2018. ISSN 2169-3536. Disponível em: <https://ieeexplore.ieee.org/document/8358685/>. Citado 3 vezes nas páginas 39, 44 e 51.

GOMES, B.; MUNIZ, L.; da Silva e Silva, F.; SANTOS, D. dos; LOPES, R.; COUTINHO, L.; CARVALHO, F.; ENDLER, M. A Middleware with Comprehensive Quality of Context Support for the Internet of Things Applications. *Sensors*, v. 17, n. 12, p. 2853, dec

2017. ISSN 1424-8220. Disponível em: <http://www.mdpi.com/1424-8220/17/12/2853>. Citado 2 vezes nas páginas 39 e 45.

GONZALEZ, C. *What Is Hannover Messe Fair and How Does It Play a Role in the U.S.?* 2016. Acessado: 26-02-2020. Disponível em: <https://www.machinedesign.com/automation-iiot/article/21834704/what-is-hannover-messe-fair-and-how-does-it-play-a-role-in-the-us>. Citado na página 15.

Grego et al. *Alice: Tecnologias desenvolvidas em Agricultura de Precisão*. 2020. <https://www.alice.cnptia.embrapa.br/alice/handle/doc/1126469?mode=full>. (Acessado em 07/07/2021). Citado na página 84.

HASSAN, Q. F. *Internet of Things A to Z: Technologies and Applications*. [S.l.]: John Wiley Sons Ltd., 2018. Citado na página 16.

HASSAN, Q. F.; KHAN, A. R.; MADANI, S. A. *Internet of Things : Challenges, Advances, and Applications*. Boca Raton, FL, USA: Chapman Hall/CRC, 2018. ISBN 978-1-4987-7851-0. Citado na página 16.

HEVNER, A. R.; MARCH, S. T.; PARK, J.; RAM, S. Design science in information systems research. *MIS Q.*, Society for Information Management and The Management Information Systems Research Center, USA, v. 28, n. 1, p. 75–105, mar. 2004. ISSN 0276-7783. Citado na página 23.

IBGE. *Agropecuária*. 2020. Acessado: 26-02-2020. Disponível em: <https://brasilemsintese.ibge.gov.br/agropecuaria.html>. Citado na página 15.

ISPA. *Precision Ag Definition | International Society of Precision Agriculture*. 2021. <https://www.ispag.org/about/definition>. (Acessado em 07/07/2021). Citado na página 83.

JACOBSEN, H.-A. Topic-based publish/subscribe. In: _____. *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009. p. 3127–3129. ISBN 978-0-387-39940-9. Disponível em: <https://doi.org/10.1007/978-0-387-39940-9_1208>. Citado na página 67.

JEON, S.; JUNG, I. MinT: Middleware for Cooperative Interaction of Things. *Sensors*, v. 17, n. 6, p. 1452, jun 2017. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/17/6/1452http://www.mdpi.com/1424-8220/17/6/1452>. Citado 2 vezes nas páginas 39 e 46.

JSFOUNDATION. *About : Node-RED*. 2021. <https://nodered.org/about/>. (Acessado em 07/06/2021). Citado na página 48.

KITCHENHAM, B. *Procedures for Performing Systematic Reviews*. Department of Computer Science, Keele University, UK, 2004. Citado na página 25.

KRAMP, T.; KRANENBURG, R. van; LANGE, S. Introduction to the internet of things. In: _____. *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 1–10. ISBN 978-3-642-40403-0. Disponível em: <https://doi.org/10.1007/978-3-642-40403-0_1>. Citado na página 16.

LALANDA, P.; MORAND, D.; CHOLLET, S. Autonomic Mediation Middleware for Smart Manufacturing. *IEEE Internet Computing*, v. 21, n. 1, p. 32–39, 2017. ISSN 10897801. Citado 2 vezes nas páginas 39 e 45.

LILIS, G.; KAYAL, M. A secure and distributed message oriented middleware for smart building applications. *Automation in Construction*, Elsevier, v. 86, n. November 2017, p. 163–175, feb 2018. ISSN 09265805. Disponível em: <https://doi.org/10.1016/j.autcon.2017.10.030https://linkinghub.elsevier.com/retrieve/pii/S0926580517304867>. Citado 2 vezes nas páginas 39 e 46.

MAFRA, S.; TRAVASSOS, G. *Estudos primários e secundários apoiando a busca por evidência em engenharia de software*. Programa de Engenharia de Sistemas e Computação, UFRJ, RJ, 2006. Citado na página 25.

MARTELLO, L. S. Zootecnia de precisão (zp): Conceitos, aplicações e desafios. In: ZOOTEC. *Anais do Zootec*. Santos, SP: Zootec, 2017. ISSN 2358-2030. Citado na página 15.

MARTIN, R. C. *Agile Software Development: Principles, Patterns, and Practices*. USA: Prentice Hall PTR, 2003. ISBN 0135974445. Citado na página 59.

MERLINO, G.; DAUTOV, R.; DISTEFANO, S.; BRUNEO, D. Enabling Workload Engineering in Edge, Fog, and Cloud Computing through OpenStack-based Middleware. *ACM Transactions on Internet Technology*, v. 19, n. 2, 2019. ISSN 15576051. Citado 2 vezes nas páginas 39 e 47.

MESMOUDI, Y.; LAMNAOUR, M.; KHAMLICHI, Y. E.; TAHIRI, A.; TOUHAFI, A.; BRAEKEN, A. A middleware based on service oriented architecture for heterogeneity issues within the internet of things (msoah-iot). *Journal of King Saud University - Computer and Information Sciences*, 2018. ISSN 2213-1248. Disponível em: <https://doi.org/10.1016/j.jksuci.2018.11.011>. Citado 2 vezes nas páginas 39 e 47.

Ministério da Agricultura. *Ministério da Agricultura eleva estimativa de VBP 2021 a recorde de R$ 1,076 tri*. 2021. Acessado: 03-06-2021. Disponível em: <https://www.noticiasagricolas.com.br/noticias/agronegocio/287784-ministerio-da-agricultura-eleva-estimativa-de-vbp-2021-a-recorde-de-r-1076-tri.html>. Citado na página 15.

MOHAMED, N.; AL-JAROODI, J.; JAWHAR, I.; LAZAROVA-MOLNAR, S.; MAHMOUD, S. SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services. *IEEE Access*, v. 5, p. 17576–17588, 2017. ISSN 2169-3536. Disponível em: <http://ieeexplore.ieee.org/document/7990214/>. Citado 2 vezes nas páginas 39 e 46.

MULLIGAN, G. H. *Portal: An Interaction Independence Middleware Framework*. Dissertação (Mestrado) — Virginia Polytechnic Institute and State University, Falls Church, Virginia, USA, 2009. Citado na página 17.

NEWMAN, S. *Building Microservices: Designing Fine-Grained Systems*. [S.l.]: O'Reilly Media, 2015. ISBN 978-1-4919-5035-7. Citado 2 vezes nas páginas 58 e 59.

NGU, A. H.; GUTIERREZ, M.; METSIS, V.; NEPAL, S.; SHENG, Q. Z. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, v. 4, n. 1, p. 1–20, Feb 2017. ISSN 2372-2541. Disponível em: <https://doi.org/10.1109/JIOT.2016.2615180>. Citado 2 vezes nas páginas 53 e 54.

NIKNEJAD, N.; ISMAIL, W.; GHANI, I.; NAZARI, B.; BAHARI, M.; HUSSIN, A. R. B. C. Understanding service-oriented architecture (soa): A systematic literature review and directions for further investigation. *Information Systems*, v. 91, p. 101491, 2020. ISSN 0306-4379. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306437920300028>. Citado na página 59.

OASIS. *AMQP is an open Internet (or "wire") Protocol standard for message-queuing communications. | AMQP*. 2020. <https://www.amqp.org/product/overview>. Acessado em 16/11/2020. Citado na página 53.

OMG. *What is DDS?* 2020. <https://www.dds-foundation.org/what-is-dds-3/>. Acessado em 17/11/2020. Citado na página 53.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, Institute of Electrical and Electronics Engineers (IEEE), v. 16, n. 1, p. 414–454, 2014. Disponível em: <https://doi.org/10.1109/surv.2013.042313.00197>. Citado na página 51.

PRESSMAN, R. *Software engineering : a practitioner's approach*. New York, NY: McGraw-Hill Education, 2015. ISBN 978-0-0780-2212-8. Citado na página 28.

RAHMAN, W. U.; CHOI, Y.-S.; CHUNG, K. Performance evaluation of video streaming application over coap in iot. *IEEE Access*, v. 7, p. 39852–39861, 2019. Citado na página 102.

RATHOD, D.; CHOWDHARY, G. Survey of middleware for internet of things. In: *2018 International Conference on Recent Trends in Advance Computing (ICRTAC)*. [S.l.: s.n.], 2018. p. 129–135. Citado 2 vezes nas páginas 53 e 54.

RAZA, H. W.; KAMAL, M. A.; ALAM, M.; SU'UD, M. M. A Review Of Middleware Platforms In Internet Of Things: A Non – Functional Requirements Approach. Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology, 2021. Citado 2 vezes nas páginas 53 e 54.

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, Feb 2016. ISSN 2372-2541. Disponível em: <https://doi.org/10.1109/JIOT.2015.2498900>. Citado 2 vezes nas páginas 53 e 54.

RUNESON, P. *Case study research in software engineering : guidelines and examples*. Hoboken, N.J: Wiley, 2012. ISBN 9781118104354. Citado na página 24.

SAID, O.; ALBAGORY, Y.; NOFAL, M.; RADDADY, F. A. Iot-rtp and iot-rtcp: Adaptive protocols for multimedia transmission over internet of things environments. *IEEE Access*, v. 5, p. 16757–16773, 2017. Citado na página 102.

SCHAFFER, M.; SCHARTNER, P.; RASS, S. Universally unique identifiers: How to ensure uniqueness while protecting the issuer's privacy. In: *Proceedings of the 2007 International Conference on Security Management*. [S.l.: s.n.], 2007. Citado na página 63.

SCHWABER, K. *Agile project management with Scrum*. Redmond, Wash: Microsoft Press, 2004. ISBN 978-0-7356-1993-7. Citado na página 26.

SCHWABER, K. *Software in 30 days : how Agile managers beat the odds, delight their customers, and leave competitors in the dust*. Hoboken, N.J: Wiley, 2012. ISBN 978-1-118-26574-1. Citado 2 vezes nas páginas 27 e 58.

SHELBY, Z.; HARTKE, K.; BORMANN, C. *The Constrained Application Protocol (CoAP)*. RFC Editor, 2014. RFC 7252. (Request for Comments, 7252). Disponível em: <https://rfc-editor.org/rfc/rfc7252.txt>. Citado na página 52.

SHOKROLLAHI, S.; SHAMS, F. Rich Device-Services (RDS): A Service-Oriented Approach to the Internet of Things (IoT). *Wireless Personal Communications*, Springer US, v. 97, n. 2, p. 3183–3201, nov 2017. ISSN 0929-6212. Disponível em: <http://link.springer.com/10.1007/s11277-017-4669-2>. Citado 2 vezes nas páginas 39 e 47.

SOMMERVILLE, I. *Software engineering*. Boston: Pearson, 2016. ISBN 978-0-1339-4303-0. Citado na página 28.

SOUZA, R. Santos de; Barbará Lopes, J. L.; Resin Geyer, C. F.; da Rosa Silveira João, L.; Afonso Cardozo, A.; Corrêa Yamin, A.; GADOTTI, G. I.; Victoria Barbosa, J. L. Continuous monitoring seed testing equipaments using internet of things. *Computers and Electronics in Agriculture*, v. 158, p. 122–132, mar 2019. ISSN 01681699. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0168169917309158>. Citado 2 vezes nas páginas 39 e 45.

STEEN, M. v.; TANENBAUM, A. S. *Distributed systems*. Third edition, version 3.01. Erscheinungsort nicht ermittelbar: Maarten van Steen, 2017. ISBN 9781543057386. Citado na página 62.

SZTAJNBERG, A.; STUTZEL, M.; MACEDO, R. Protocolos de aplicação para a internet das coisas: conceitos e aspectos práticos. In: CSBC. *Jornadas de Atualização em Informática*. Natal, RN: SBC, 2018. v. 37, p. 99–148. Citado na página 16.

TADIM, K. *O impacto das leis de incentivo na Indústria 4.0*. 2020. Acessado: 26-02-2020. Disponível em: <http://anpei.org.br/industria-4-0-o-que-e/>. Citado na página 15.

TEEL, J. *Comparison of Wireless Technologies*. 2018. Acessado: 08/03/2020. Disponível em: <https://predictabledesigns.com/wireless_technologies_bluetooth_wifi_zigbee_gsm_lte_lora_nb-iot_lte-m/>. Citado na página 16.

THINGSPEAK. *Moisture Sensor using HTTP POST Requests to Channel - MATLAB & Simulink*. 2021. <https://www.mathworks.com/help/thingspeak/MoistureMonitor.html>. (Acessado on 07/09/2021). Citado na página 96.

TRILLES, S.; GONZáLEZ-PéREZ, A.; HUERTA, J. An iot platform based on microservices and serverless paradigms for smart farming purposes. *Sensors*, v. 20, n. 8, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/8/2418>. Citado 2 vezes nas páginas 39 e 43.

VIKASH; MISHRA, L.; VARMA, S. Middleware technologies for smart wireless sensor networks towards internet of things: A comparative review. *Wireless Personal Communications*, Springer Science and Business Media LLC, v. 116, n. 3, p. 1539–1574, ago. 2020. Disponível em: <https://doi.org/10.1007/s11277-020-07748-7>. Citado 2 vezes nas páginas 53 e 54.

WEISER, M. The computer for the 21st century. *Scientific American*, p. 94–104, September 1991. Disponível em: <https://www.lri.fr/~mbl/Stanford/CS477/papers/Weiser-SciAm.pdf>. Citado na página 16.

WIERINGA, R. J. *Design Science Methodology for Information Systems and Software Engineering*. Berlin: Springer, 2014. ISBN 978-3-662-43838-1. Citado na página 19.

XIE, C.; YU, B.; ZENG, Z.; YANG, Y.; LIU, Q. Multilayer internet-of-things middleware based on knowledge graph. *IEEE Internet of Things Journal*, v. 8, n. 4, p. 2635–2648, 2021. Citado 2 vezes nas páginas 39 e 44.

ZHAO, R.; WANG, L.; ZHANG, X.; ZHANG, Y.; WANG, L.; PENG, H. A OneM2M-Compliant Stacked Middleware Promoting IoT Research and Development. *IEEE Access*, v. 6, p. 63546–63559, 2018. ISSN 2169-3536. Disponível em: <https://ieeexplore.ieee.org/document/8492453/>. Citado 2 vezes nas páginas 39 e 43.