UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
INSTITUTO DE GEOCIÊNCIAS E CIÊNCIAS EXATAS – RIO CLARO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESTABELECIMENTO DE UMA ARQUITETURA DE REFERÊNCIA PARA SISTEMAS CIBER-FÍSICOS AUTOADAPTATIVOS

MARCOS PAULO DE OLIVEIRA CAMARGO

RIO CLARO-SP
2023

UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

MARCOS PAULO DE OLIVEIRA CAMARGO

ESTABELECIMENTO DE UMA ARQUITETURA DE REFERÊNCIA PARA SISTEMAS CIBER-FÍSICOS AUTOADAPTATIVOS

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Frank José Affonso

Rio Claro – SP
2023

C172e

Camargo, Marcos Paulo de Oliveira
Estabelecimento de uma Arquitetura de Referência para Sistemas Ciber-Físicos Autoadaptativos / Marcos Paulo de Oliveira Camargo. -- Rio Claro, 2023
120 p.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Geociências e Ciências Exatas, Rio Claro
Orientador: Frank José Affonso

1. Arquitetura de referência. 2. Sistemas ciber-físicos. 3. Sistemas autoadaptativos. 4. Self-cps. 5. E-health. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Geociências e Ciências Exatas, Rio Claro. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

MARCOS PAULO DE OLIVEIRA CAMARGO

ESTABELECIMENTO DE UMA ARQUITETURA DE REFERÊNCIA PARA SISTEMAS CIBER-FÍSICOS AUTOADAPTATIVOS

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Comissão Examinadora

Prof. Dr. FRANK JOSÉ AFFONSO
Departamento de Estatística, Matemática Aplicada e Computação (DEMAC)
Universidade Estadual Paulista "Júlio De Mesquita Filho" (UNESP) – Câmpus de Rio Claro

Prof. Dr. MÁRCIO ANDREY TEIXEIRA
Instituto Federal de São Paulo (IFSP)
Unidade de Catanduva

Profa. Dra. RENATA SPOLON LOBATO
Departamento de Ciências de Computação e Estatística (DCCE)
Universidade Estadual Paulista "Júlio De Mesquita Filho" (UNESP) – Câmpus de São José do Rio Preto

Conceito: Aprovado

Rio Claro (SP), 30 de agosto de 2023

Este trabalho é dedicado a Deus. Deu-me forças
nos momentos difíceis e não me deixou
desistir diante dos problemas.

Agradecimentos

Agradeço a Deus. Abençoou-me com saúde e disposição para concluir essa longa jornada acadêmica.

Agradeço a minha família, que me apoiaram em todos os sentidos e não me deixaram me desanimar dessa oportunidade.

Agradeço ao meu orientador Prof. Dr. Frank José Afonso por proporcionar essa oportunidade, pelos ensinamentos, paciência, companheirismo e por ajudar a trilhar essa fase acadêmica.

Agradeço a Alpha7 Software por permitir a flexibilização do horário de trabalho, o qual facilitou o desenvolvimento deste trabalho. Além disso, agradecimentos ao Fábio Gomes Suzarth Júnior e ao João Eduardo Rocha pelos ensinamentos técnicos fornecidos, que por consequência, ajudaram no desenvolvimento do mesmo.

Agradecimentos especiais aos integrantes do nosso grupo de pesquisa (Gabriel dos Santos Pereira, Daniel de Almeida, Leandro Apolinário Bento e William Fernandes Dorante) pelo auxílio no desenvolvimento do estudo de caso.

Por fim, agradeço a todos amigos e familiares que me incentivaram em busca dessa conquista acadêmica.

Para nós os grandes homens não são aqueles que resolveram os problemas, mas aqueles que os descobriram.

Albert Schweitzer

RESUMO

Nos últimos anos, sistemas de computação têm emergido em diversos setores da sociedade. Do ponto de vista tecnológico, diversos sistemas têm sido beneficiados com o apoio de tecnologias como IoT (do inglês, Internet of Things), Cloud Computing, entre outras. Dentre as classes de sistemas existentes na literatura, os Sistemas Ciber-Físicos (do inglês, *Cyber Physical Systems* – CPS) possui uma posição de destaque na comunidade científica e iniciativa privada por caracterizar a interação de entidades computacionais com processos físicos, dispondo de serviços de processamento e armazenamento localizados na Internet. Contudo, por lidarem com problemas de incertezas de requisitos na fase de *design* e mudanças em tempo de execução, um novo conjunto de dificuldades é emergido. Diante desse contexto, origina-se uma nova classe de sistemas denominada Sistemas Ciber-Físicos Autoadaptativos (do inglês, *Self-adaptive Cyber Physical Systems – Self-CPS*). Em linhas gerais, um Self-CPS combina as características dos CPSs com mecanismos autoadaptativos capazes de reconhecer as mudanças do sistema e de contexto e propor soluções de natureza estrutural, comportamental e/ou relacionado ao ambiente de execução com a finalidade de manter seu estado de execução sem (ou com o mínimo de) intervenção humana. Nesse sentido, tem-se observado que os Self-CPS têm sido desenvolvidos sem a sistematização de conhecimento e boas práticas de engenharia. Visando obter uma visão geral acerca dos Self-CPS, foi conduzida uma investigação da literatura adotando a técnica de mapeamento sistemático (do inglês, *Systematic Mapping Study* – SMS) em relação aos estudos secundários, expondo as principais contribuições e os objetos de investigação de cada estudo. Em um segundo momento, foi realizada uma investigação acerca dos estudos primários que propuseram modelos/arquiteturas de referência para Self-CPS. A partir dos resultados oriundos do mapeamento sistemático realizado, ficou perceptível a inexistência de uma abordagem sistemática que apoie o desenvolvimento de tais modelos/arquiteturas de referência, além de não lidar com os interesses de autoproteção e observabilidade. Diante do exposto, o principal objetivo desta dissertação é sistematizar o conhecimento obtido em uma arquitetura de referência que integre abordagens de autoadaptação, autoproteção, observabilidade e as melhores práticas de Engenharia de Software. Além disso, pode-se destacar também que apresentar um panorama detalhado a respeito do estágio atual sobre modelos e arquiteturas de referência também é um objetivo deste trabalho. Por fim, como forma de avaliar a arquitetura de referência proposta, um estudo de caso voltado ao domínio da saúde foi conduzido, uma vez que foi evidenciado uma carência na literatura sobre trabalhos que apresentam soluções baseadas em (Self-)CPS aplicadas a este domínio. Para averiguar a qualidade da arquitetura proposta, uma inspeção baseada em *checklist* foi conduzida, analisando se os requisitos arquiteturais propostos foram alcançados.

Palavras-chave: Arquitetura de referência, Sistemas ciber-físicos; Sistemas autoadaptativos; Self-cps; E-health

ABSTRACT

In recent years, computing systems have emerged in many segments of our society. From a technological perspective, several systems have benefited from the support of technologies such as IoT (Internet of Things), Cloud Computing, among others. Among the system classes found in the literature, Cyber-Physical Systems (CPS) have played a prominent position in the scientific community and private enterprise. In short, CPS can be characterized by the interaction of computational entities with physical processes, with processing and storage services on the Internet. However, because of dealing with problems of uncertain requirements in the design phase and changes at runtime, a new set of difficulties emerged. Thus, a new class of systems, referred to as Self-adaptive Cyber-Physical Systems (Self-CPS), has been created. In summary, a Self-CPS combines features of CPS with self-adaptive mechanisms so that it can recognize changes in context and propose solutions to a structural, behavioral, and/or execution environment in order to maintain their state of execution without (or with minimal) human intervention. In this sense, it has been observed that these systems have been developed without the systematization of knowledge and good engineering practices. In order to get an overview of Self-CPS, an investigation of the literature was conducted using the systematic mapping technique in relation to secondary studies, exposing the main contributions and the main objects of investigation of each study. Next, an investigation was conducted regarding the primary studies that proposed reference models/architectures for Self-CPS. Based on the results of our investigation, it became clear that there is no systematic approach to support the development of these models/architectures for Self-CPS, besides not addressing the interests of self-protecting and observability. Thus, the main purpose of this dissertation is to systematize the knowledge obtained, condensing it into a reference architecture that integrates approaches to self-adaptation, self-protecting, observability, and the best practices of Software Engineering. Moreover, it can be also highlighted that presenting a detailed overview of the current state of reference models and architectures is an objective of this dissertation. In order to evaluate the proposed architecture, a case study applied to the healthcare domain was conducted, since there is a lack of studies in the literature presenting solutions based on (Self-)CPS applied to this domain. To check the quality of our architecture, a checklist-based inspection was conducted, analyzing whether the proposed architectural requirements were met.

Keywords: Reference architecture, Cyber-physical systems; Self-adaptive systems; Self-cps; E-health.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AR	Arquitetura de Referência
BAA	*Broad Agency Announcement*
CPPS	*Cyber-Physical Production Systems*
CPS	*Cyber-Physical Systems*
EC	*Exclusion Criteria*
HTTP	*Hyper Text Transfer Protocol*
IC	*Inclusion Criteria*
ICT	*Information and Communication Technology*
IoT	*Internet of Things*
JSON	JavaScript Object Notation
JWT	*JSON Web Token*
MAPE	*Monitor, Analyze, Plan, Execute*
MAPEK	*Monitor, Analyze, Plan, Execute*, Knowledge
MCPS	*Medical Cyber-Physical Systems*
NSF	*National Science Foundation*
OAUTH	*Open Authorization*
PROSA-RA	*Process based on Software Architecture* - Reference Architecture
QoS	*Quality of Service*
QP	Questões de Pesquisa
RA	Requisitos de Adaptação
RASCPS	*Reference Architecture for Self CPS*
RCF	Requisitos Ciber Físico
REST	*Representational State Transfer*
SaS	*Self-adaptive Systems*
SCPS	*Self Management Cyber-Physical Systems*
SEAMS	*International Symposium on Software Engineering for Adaptive and Self-Managing Systems*
Self-CPS	*Self-adaptive Cyber-Physical Systems*
SHHC	*Smart Home Health Care*
SIC	Sistema de Inteligência Coletiva
SMS	*Systematic Mapping Study*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
V&V	Validação e Verificação
V2V	*Vehicle-to-Vehicle*
V2X	*Vehicle-to-Everything*
WSDL	*Web Services Description Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Atualmente, nota-se a presença de sistemas de software em todos os setores de nossa sociedade. Sobre a perspectiva dos usuários comuns, observa-se, por exemplo, que a maneira como os conteúdos audiovisuais estão sendo consumidos atualmente foi alterada drasticamente em relação a um passado recente. Essa mudança é respaldada pela popularização das redes sociais e dos serviços de *streaming* sob demanda (ERICSSON, 2020; ERICSSON, 2018). Nessa direção, estima-se que até em 2024, 75 bilhões de dispositivos inteligentes ao redor do mundo estejam conectados à Internet (NORD; KOOHANG; PALISZKIEWICZ, 2019). Essa mudança de cultura pode ser explicada pela massiva adoção dos dispositivos móveis (por exemplo, *Smartphones*, *Notebooks*, entre outros), dispositivos inteligentes (por exemplo, *Smartwatches*, *Smart TVs*, etc) e outros tipos de dispositivos (por exemplo, entidades físicas que controlam um ambiente). Além disso, verifica-se também que, no cenário exposto, certas tecnologias e ferramentas como *Cloud Computing*, *Big Data*, *IoT* (do inglês, *Internet of Things*), Inteligência Artificial, Realidade Aumentada, entre outras, têm criado um ambiente próspero para o desenvolvimento de sistemas que operam em diferentes domínios.

Dentre as classes de sistemas existentes na literatura, os Sistemas Ciber-Físicos (do inglês, *Cyber-Physical Systems* – CPS) vêm ganhando destaque na comunidade científica e na iniciativa privada (LI *et al.*, 2017). Ao analisar a literatura, diferentes definições sobre CPS podem ser encontradas. Dentre elas, optou-se por destacar neste trabalho as definições de Kang *et al.* (2016) e Zeadally, Sanislav e Mois (2019) por ambas enfatizarem a comunicação entre os componentes desse tipo de sistema via Internet e pela segunda complementar a primeira quanto ao uso do espaço, tempo e interação entre o sistema e o ambiente físico da aplicação.

> "(...) sistemas com capacidade computacional que estão conectados com entidades do mundo físicos e os processos que os cercam, ao mesmo tempo que fornecem e usam serviços de acesso e processamento de dados disponíveis na Internet" (KANG *et al.*, 2016) (tradução nossa).

> "(...) uma nova geração de sistemas de engenharia onde componentes cibernéticos e físicos estão fortemente interconectados, onde cada um opera em escala espacial e temporal distinta, exibindo comportamentos e interações diversas que podem ser alterados conforme mudanças no ambiente de aplicação" (ZEADALLY; SANISLAV; MOIS, 2019) (tradução nossa).

No que diz respeito à empregabilidade, Muccini, Sharaf e Weyns (2016), Musil *et al.* (2017) comentaram que os CPS têm sido utilizados predominantemente no desenvolvimento de sistemas em seis domínios de software, a saber: automação industrial, robótica, transporte, infraestrutura em grande escala, militar e saúde. Especificamente sobre este último, os referidos autores reportaram um importante destaque para os sistemas *e-Health*, que são geralmente projetados para auxiliar os usuários finais por meio de um monitoramento contínuo, visando melhorar a sua qualidade de vida desses usuários (GRUA; SANCTIS; LAGO, 2020). Ainda nessa direção, Grua, Sanctis e Lago (2020) elencou as principais características desse tipo de sistema, a saber: (i) emprego de sensores dos *smartphones*, (ii) facilidade de acesso do usuário ao sistema devido aos baixos custos de infraestrutura, e (iii) fornecimento de serviços por meio de dispositivos móveis.

Assim como ocorre nas demais classes de sistemas, os CPS também lidam com problemas de incertezas de requisitos na fase de *design* e mudanças em tempo de execução (CHENG *et al.*, 2009; GEROSTATHOPOULOS *et al.*, 2016; RATASICH *et al.*, 2017; KLUGE, 2020). Nesse sentido, pode-se citar a dificuldade em se estabelecer processos e/ou metodologias capazes de prever/trabalhar com todos os requisitos dos sistemas. Além disso, por lidar tanto com projeto do software quanto o sensoriamento do ambiente físico, prever as mudanças e/ou comportamentos desse ambiente também é uma dificuldade inerente ao desenvolvimento de CPS. Diante do exposto, uma nova de classe de sistemas denominada Sistemas Ciber-Físicos Autoadaptativos (do inglês, *Self-adaptive Cyber Physical Systems* – Self-CPS) tem emergido na literatura (MUCCINI; SHARAF; WEYNS, 2016; ZHOU *et al.*, 2019). Essa nova classe de sistema combina a base dos CPS com características de adaptação existentes nos Sistemas Autoadaptativos (do inglês, *Self-adaptive Systems* – SaS). De acordo com (SALEHIE; TAHVILDARI, 2009; CHENG *et al.*, 2009), SaS é um sistema onde *"suas funcionalidades são avaliadas e modificadas quando não está cumprindo seus objetivos, ou quando os atributos de desempenho não são satisfeitos"*. Assim, pode-se dizer que os Self-CPS combinam a natureza dos CPS com as melhores práticas de engenharia de sistemas autoadaptativos para que tais sistemas possam reconhecer as mudanças de contexto (ou seja, ambiente de execução) e propor soluções (ou seja, mudanças estruturais, comportamentais e/ou de contexto) para manter seu estado de execução sem (ou com o mínimo de) intervenção humana.

1.2 Motivação

Motivado pelo contexto apresentado e pela necessidade de se obter uma visão geral sobre o tema de pesquisa entorno das áreas envolvidas (SaS e CPS), um mapeamento sistemático (do inglês, *Systematic Mapping Study* - SMS) foi conduzido durante o desenvolvimento deste trabalho. Vale destacar que essa técnica (SMS) permite que seja possível conduzir uma avalia-

ção completa e confiável sobre um tema de interesse por meio de uma metodologia confiável e isenta de influências pessoais (KITCHENHAM; DYBA; JORGENSEN, 2004; KITCHENHAM; CHARTERS, 2007). Primeiramente, foi conduzida uma análise da literatura em relação aos estudos secundários sobre Self-CPS, destacando as principais contribuições e objetos de investigação de cada estudo. Por fim, uma investigação acerca dos estudos primários que reportaram modelos e/ou arquiteturas de referência para Self-CPS foi realizada. Além disso, vale ressaltar que o foco de investigação desse SMS é o domínio de saúde (*e-Health*).

Ao analisar o resultado do mapeamento sistemático ficou evidente a ausência da aplicação de uma abordagem sistemática que apoie o desenvolvimento dessas arquiteturas/modelos. Em paralelo, considerando a relevância das arquiteturas de referência em relação ao desenvolvimentos desses sistemas, pode-se dizer que nenhum estudo teve como foco o domínio da saúde. Por outro lado, também vale destacar que nenhum estudo desse mapeamento lidou com interesses de autoproteção e observabilidade nas arquiteturas/modelos de referência propostos, características importantes para o funcionamento adequado de um (Self-)CPS.

1.3 Objetivos

Diante da motivação reportada neste capítulo, o objetivo deste projeto de mestrado acadêmico é sintetizar o conhecimento obtido através da execução de uma revisão literária sobre modelos e/ou arquiteturas de referência para Self-CPS, que possam ser instanciados(as) para o domínio de saúde. Considerando a importância de arquitetura de referência no desenvolvimento de sistemas (BASS; CLEMENTS; KAZMAN, 2012; NAKAGAWA; OQUENDO; BECKER, 2012), optou-se por representar o conhecimento adquirido nessa estrutura (ou seja, arquitetura de referência) de tal modo a integrar abordagens de autoadaptação, autoproteção, observabilidade e as melhoras práticas e técnicas da Engenharia de Software. O conceito de observabilidade pode ser sintetizado como a capacidade de estimar, mensurar e entender o estado atual do sistema por meio dos dados de saída recolhidos. Já a autoproteção pode ser definida como a capacidade que uma aplicação tem de se proteger de ataques/ameaças externas. Por fim, a integração de técnicas adaptativas tem por objetivo minimizar significativamente os principais desafios encontrados no desenvolvimento de CPS, os quais são mencionados na Seção 2.4. Para que esse objetivo geral seja alcançado, os seguintes objetivos específicos são propostos:

- Projeto de uma arquitetura de referência para Self-CPS. Para isso, o mesmo método de construção de arquiteturas de referência, utilizado no projeto de outras arquiteturas de nosso grupo grupo de pesquisa (AFFONSO; NAKAGAWA, 2013; AFFONSO; PASSINI; NAKAGAWA, 2019), deve ser adotado;
- Especificação e desenvolvimento de um sistema que possa lidar com a observabilidade e

autoproteção. Para isso, pretende-se experiências anteriores de nosso grupo de pesquisa (PASSINI, 2020; MARTINS, 2022) e *frameworks*/bibliotecas disponíveis na literatura; e

- Avaliação da arquitetura sob o ponto de vista do impacto de utilização. Para isso, pretende-se utilizar técnicas similares de avaliação às reportadas no mapeamento apresentado no Capítulo 3.

Conforme mencionado na Seção 1.2, um SMS foi conduzido durante o desenvolvimento deste trabalho. Portanto, outro objetivo geral deste trabalho é apresentar um panorama detalhado referente ao estágio atual sobre modelos e/ou arquiteturas de referência para Self-CPS. Essa investigação pode nortear pesquisadores e/ou profissionais da indústria no desenvolvimento desse tipo de aplicação e/ou no projeto de novas arquiteturas para as áreas relacionadas a este trabalho (por exemplo, CPS e arquitetura de software).

1.4 Organização da monografia

Esta monografia é organizada como segue. No Capítulo 2 é apresentada a fundamentação teórica necessária para compreender o conteúdo dos capítulos subsequentes. Dessa forma, são abordados os principais termos, conceitos e teorias relacionados aos temas de pesquisa investigados neste trabalho. No Capítulo 3 é apresentada a condução do mapeamento sistemático da literatura, visando compreender a relação entre SaS e CPS. Esse capítulo é iniciado com uma investigação preliminar sobre estudos secundários relacionados ao tema deste trabalho de mestrado, e na sequência é apresentado o mapeamento da literatura. A partir do conhecimento adquirido, foi possível capturar o estado da arte sobre Self-CPS na literatura. No Capítulo 4, a arquitetura de referência para Self-CPS desenvolvida neste trabalho é apresentada, assim como os detalhes de suas camadas e módulos. No Capítulo 5 é apresentada uma visão detalhada do estudo de caso conduzido para avaliar a aplicabilidade da arquitetura de referência proposta neste trabalho. Por fim, no Capítulo 6 são reportadas as contribuições oriundas desta dissertação, bem como as perspectivas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

O objetivo deste capítulo é apresentar os principais conceitos e definições dos temas relacionados a este projeto de mestrado acadêmico. Inicialmente, na Seção 2.1 são apresentados os principais conceitos sobre arquitetura de software, modelo de referência, arquitetura de referência e arquitetura concreta. Na Seção 2.2 são descritos os conceitos e definições sobre sistemas *e-Health*, sendo evidenciado um panorama geral desse tipo de sistema, assim como as principais lacunas existentes. Na Seção 2.3 são reportadas as principais definições sobre SaS, assim como um breve panorama sobre esse tema de pesquisa, sendo explorados os principais modelos de implementação, abordagens para adaptação e padrões utilizados em tais abordagens. Na Seção 2.4 são apresentadas definições sobre CPS, assim como os principais modelos de arquiteturas e componentes que fazem parte de tal sistema. Por fim, as considerações finais sobre o conteúdo apresentado neste capítulo são reportadas na Seção 2.5.

2.1 Arquitetura de software

Atualmente, nota-se a presença de sistemas de software em diferentes segmentos de nossa sociedade. Contudo, as dificuldades e complexidades encontradas no desenvolvimento desses sistemas aumentaram significativamente nos últimos anos. Em alguns segmentos, a necessidade de monitoramento constante do ambiente de execução pode ser considerada como um requisito de primeira classe, pois uma falha ou uma reação tardia em tais sistemas pode acarretar danos irreparáveis ao ambiente ou até mesmo perda de vidas humanas. Em outros sistemas, a necessidade de operar de modo ininterrupto, aliada a diversas condições adversas, pode ser caracterizada como uma situação ordinária. Diante desse contexto, é imprescindível que determinados atributos de qualidade sejam integrados ao sistema, como escalabilidade, confiabilidade, robustez, entre outros, para garantir a longevidade e sucesso do sistema. Para atingir tal objetivo, a elaboração de uma arquitetura de software é necessária, pois estabelece a estrutura e relação entre componentes, além de atuar como um guia para a equipe de desenvolvimento.

De acordo com o padrão IEEE 1471, arquitetura de software é definida como "uma organização fundamental de um sistema incorporado em seus componentes, suas relações entre si e com o ambiente de aplicação, e os princípios que orientam seu projeto e evolução" (EELES; CRIPPS, 2010). Outra visão é dada por Mesli-Kesraoui *et al.* (2016), estabelecendo a arquitetura de software como "uma configuração de um conjunto de componentes e conectores". Os componentes são caracterizados como unidades computacionais ou de armazenamento, enquanto que os conectores representam a interação entre os componentes, como protocolos de comuni-

cação, retorno de uma execução. Além disso, a arquitetura pode ser apresentada de diferentes maneiras, a saber: (i) estrutural, (ii) comportamental e (iii) físico. Do ponto de vista estrutural, a arquitetura tem como foco apresentar o arranjo estrutural dos componentes e conectores. Do ponto de vista comportamental, o foco é deixar evidente as ações realizadas, o modo e o grau de interação entre os componentes e conectores. Por último, o ponto de vista físico evidencia os componentes físicos e as interações com os componentes físicos.

De acordo com Tummers *et al.* (2021), uma arquitetura de referência (AR) pode ser caracterizada como uma 'arquitetura genérica e que ajuda a desenvolver arquiteturas de softwares específicas para múltiplos sistemas de softwares'. Nesse sentido, uma AR atua como um guia para o desenvolvimento de novas versões do sistema (MULLER, 2008). Além disso, uma AR representa um *design* genérico utilizado para desenvolver uma arquitetura concreta mais rapidamente e com maior qualidade, com base nas preocupações das partes interessadas identificadas.

Em paralelo, Bass, Clements e Kazman (2012) definiram AR como um tipo especial de arquitetura de software por facilitar o reuso do conhecimento arquitetônico adquirido. Na visão de Kruchten (2004), uma AR "é um padrão arquitetônico predefinido, ou conjunto de padrões, parcialmente ou totalmente instanciado, projetado e comprovado para uso em contextos técnicos e de negócios específicos, juntamente com elementos de suporte que permite seu uso". Nakagawa, Oquendo e Becker (2012) sintetiza que uma AR fornece conhecimento sobre como projetar arquiteturas concretas de sistemas de um domínio de aplicação em específico, abrangendo as regras de negócios, as melhores práticas de desenvolvimento de software, estilos arquiteturais e os elementos de software que compõem o desenvolvimento de sistemas para o domínio em questão.

De acordo com Nakagawa *et al.* (2014), as arquiteturas são projetadas sem adotar uma abordagem sistemática, o que pode acarretar a um esforço desproporcional à medida que a complexidade do software aumenta. Para contornar esse cenário, Nakagawa *et al.* (2014) elaboraram o PROSA-RA (do inglês, *Process based on Software Architecture – Reference Architecture*), um processo para auxiliar o projeto de arquiteturas de referência. Para atingir seus objetivos, esse processo possui quatro fases, a saber: (i) investigação das fontes de informação, (ii) estabelecimento dos requisitos arquiteturais, (iii) projeto arquitetural, e (iv) avaliação da arquitetura de referência. Por razões de objetivo e espaço, detalhes dessas fases não serão reportados nesta seção.

2.2 *e-Health*

De acordo com Zhang *et al.* (2017), sistemas de software estão sendo amplamente adotados no domínio da saúde nas últimas duas décadas. Nesse sentido, vale destacar que a

recente evolução tecnológica de diversos recursos computacionais e/ou dispositivos de hardware têm colaborado com o cenário supracitado (ZHANG *et al.*, 2017). Em geral, esses recursos têm sido utilizados no desenvolvimento de sistemas voltados para o monitoramento remoto de pacientes, onde é possível avaliar o estado de saúde dos mesmos por meio de sensores e, caso necessário, emitir alarmes informativos para esses pacientes e/ou profissionais das áreas de saúde (ou seja, enfermeiros, médicos, serviços de emergência).

Grua, Sanctis e Lago (2020) comentaram que os sistemas *e-Health* são geralmente projetados para auxiliar os usuários finais por meio de um monitoramento contínuo, visando melhorar a qualidade de vida desses usuários. Ainda nessa direção, os autores elencaram as principais características desse tipo de sistema, a saber: (i) emprego de sensores dos *smartphones*, (ii) facilidade de acesso do usuário ao sistema devido aos baixos custos de infraestrutura, e (iii) fornecimento de serviços por meio de dispositivos móveis.

De acordo com Hossain, Bari e Khan (2022), os sistemas *e-Health* têm ganhado destaque e popularização nos últimos anos. Além disso, os autores ressaltaram o forte vínculo com a tecnologia IoT, qualificando esses sistemas como os que mais capturam dados em tempo-real com o propósito de fornecer um rápido retorno aos pacientes e/ou especialistas. Segundo Fonseca *et al.* (2021), estes sistemas (*e-Health*) podem ser caracterizados como um 'conjunto de tecnologias aplicadas com o auxílio da internet, em quais serviços de saúde são fornecidos para melhorar a qualidade de vida e facilitar a prestação de cuidados de saúde'. Sahi *et al.* (2018) ressaltaram a questão das tecnologias de informação e comunicação (do inglês, *Information and Communication Technology* – ICT) aplicado ao domínio *e-Health*, permitindo que os usuários tenham ciência de seu estado de saúde independente do local e do horário.

Com a finalidade de estudar as tendências e práticas mais comuns da área, Fonseca *et al.* (2021) conduziram uma revisão da literatura sobre os estudos publicados durante o período de 2014 a 2019, totalizando 446 artigos. Um dos pontos a serem estudados foi a questão das tecnologias integradas e, nesse sentido, os autores identificaram IoT, Segurança, Computação em Nuvem, *Big Data*, criptografia como as mais utilizados. No estudo de Islam *et al.* (2015), Computação em Nuvem, *Big Data*, Redes, *Wearables* e Realidade Aumentada foram as tecnologias em destaque. Em relação aos dispositivos *Wearables*, Singh (2018) propôs uma classificação para os sensores médicos, categorizando em sensores de contato, o qual contempla os *wearables*, e sensores sem contato, conhecidos como periféricos. Ao todo, essas tecnologias supracitadas enriquecem o domínio *e-Health*, ampliando o escopo de aplicação dos sistemas desenvolvidos.

Conforme exposto nesta seção, o domínio *e-Health* vem ganhando destaque na indústria e na comunidade científica nos últimos anos. Por consequência, devido à sua rápida dissemina-ção, algumas lacunas ficam mais evidentes à medida que novas soluções são elaboradas. Diante

desse cenário, Abdulmalek *et al.* (2022) levantaram alguns pontos de destaque a serem explorados nessa área, a saber: (i) segurança, (ii) energia e (iii) inteligência computacional. No que concerne sobre a segurança, os autores reforçam para a necessidade de implementar mecanismos de proteção em cada camada do sistema, minimizando as possibilidades dos dados serem espionados ou manipulados, indevidamente, por terceiros. Em relação à energia, é apontado sobre a limitação e eficiência energética dos diversos dispositivos e, nesse sentido, as arquiteturas que priorizam o baixo consumo são essenciais. Já em relação à inteligência computacional, os autores destacam sua aplicação na questão da coleta e processamento dos dados e apontam que é essencial que os sistemas implementam mecanismos avançados de inteligência computacional, tendo em vista que os recursos computacionais que os dispositivos da aplicação possuem energia e poder computacional limitados.

2.3 Sistemas autoadaptativos

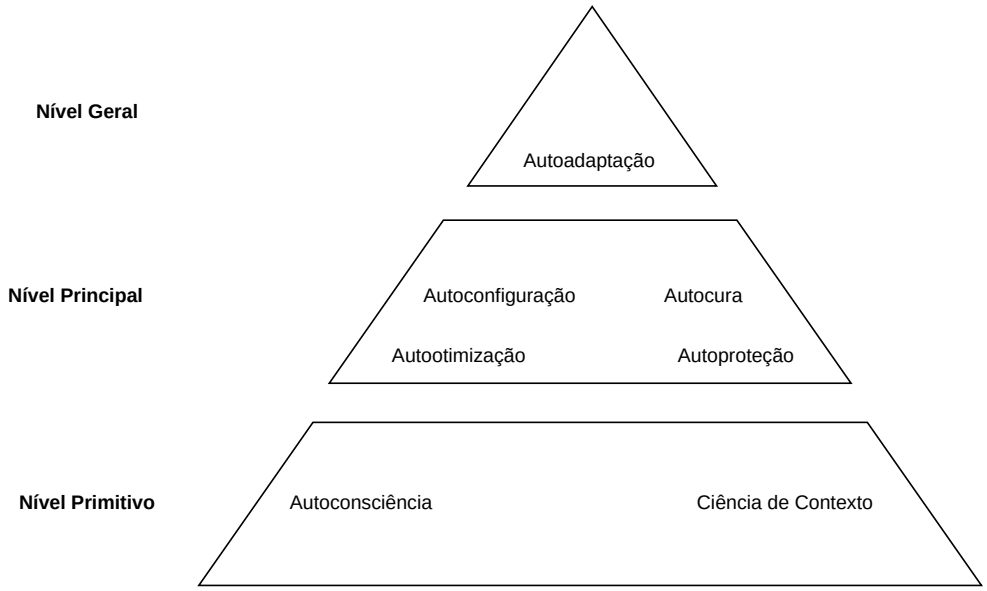
Em 2003, a IBM publicou um manifesto alertando para uma iminente crise de complexidade do software, cuja causa é ocasionada pelo aumento de complexidade em suas diversas fases (por exemplo, instalação, ajuste, configuração e manutenção). Como resultado, esse manifesto reportou que incertezas são geradas depois da fase de implantação do software, pois nem todos os requisitos são previstos na fase de *design*, fazendo com que o software tenha que passar por um processo de modificação para que os requisitos iniciais sejam atendidos. Com base no cenário exposto, uma nova classe de sistemas de software conhecida como autoadaptativos foi criada com base em mecanismos de autogerenciamento, que são capazes de se adaptar sem, ou com o mínimo de, intervenção humana (WEYNS, 2017).

Segundo Oreizy *et al.* (1999), Kephart e Chess (2003), SaS pode ser caracterizado como "um software que modifica seu próprio comportamento em função de mudanças ocorridas em seu ambiente operacional. Por esse ambiente, pode-se entender como qualquer coisa observável pelo sistema de software, como entrada do usuário final, dispositivos e sensores externos de hardware ou instrumentação do programa". Em síntese, a adaptação pode ser alcançada por meio de modificações nos parâmetros ou nos componentes do programa, em respostas aos eventos de mudanças ocorridos no sistema ou em seu ambiente de execução. Uma definição semelhante também é fornecida pela *Broad Agency Announcement* (BAA), que caracteriza SaS como um software que avalia o seu próprio comportamento e realiza modificações em si mesmo quando os objetivos estabelecidos na fase de projeto não são comprido, ou ainda, quando a substituição de uma funcionalidade pode, por exemplo, melhorar sua performance (ROBERTSON; SHROBE; LADDAGA, 2003). Brun *et al.* (2009) define SaS como um sistema que decide sobre ações de maneira autônoma a fim de se adaptar à mudanças ocorridas no contexto e no ambiente de operação do sistema. Na visão de Cheng *et al.* (2009), SaS é um sistema que ajusta seu

comportamento em resposta à sua percepção de si mesmo e/ou do ambiente de execução ao qual está inserido. Segundo Musil *et al.* (2017), SaS é um sistema habilitado para modificar seu comportamento em tempo de execução com o intuito de alcançar os objetivos do sistema.

Diante do contexto atual de desenvolvimento de software, a complexidade de diversos sistemas tem crescido de maneira significativa no decorrer dos últimos anos. Uma forma de gerenciar a complexidade inerente ao software é incorporar mecanismos de adaptação, que podem trazer diversas propriedades (por exemplo, autogerenciamento e autocontrole) quando integrado ao sistema em questão. Como forma de apresentar e organizar essas propriedades, Salehie e Tahvildari (2009) apresenta uma hierarquia organizada em três níveis, como ilustra a Figura 1. A seguir, uma breve descrição de cada nível é apresentada.

Figura 1 – Hierarquia das propriedades de adaptação (self-*)



Fonte: Adaptado de (SALEHIE; TAHVILDARI, 2009)

Nível Geral. Nível topo da hierarquia, composto pelas propriedades globais da autoadaptação, *self-adaptiveness* (autoadaptação) e *self-organizing* (auto-organização). Por sua vez, *self-adaptiveness* pode ser decomposto em várias propriedades, a saber: (i) *self-managing* (autogerenciamento), (ii) *self-governing* (autogovernado), (iii) *self-maintenance* (automanutenção), (iv) *self-control* (autocontrole), e (v) *self-evaluating* (autoavaliação). Sistemas que implementam a propriedade *self-organizing* é caracterizado por serem descentralizados, sendo formados por elementos interativos que possui conhecimento parcial ou nenhum sobre o estado do sistema.

Nível Principal. Nível composto por quatro propriedades, a saber: (i) *self-configuration* (autoconfiguração), (ii) *self-healing* (autocura), (iii) *self-optimization* (auto-otimização), e (iv) *self-protecting* (autoproteção). Essas propriedades são inspiradas em mecanismos

de autoadaptação biológicos (KEPHART; CHESS, 2003), que podem ser exemplificados pela adaptação do corpo humano quando a temperatura ambiente está alta ou baixa.

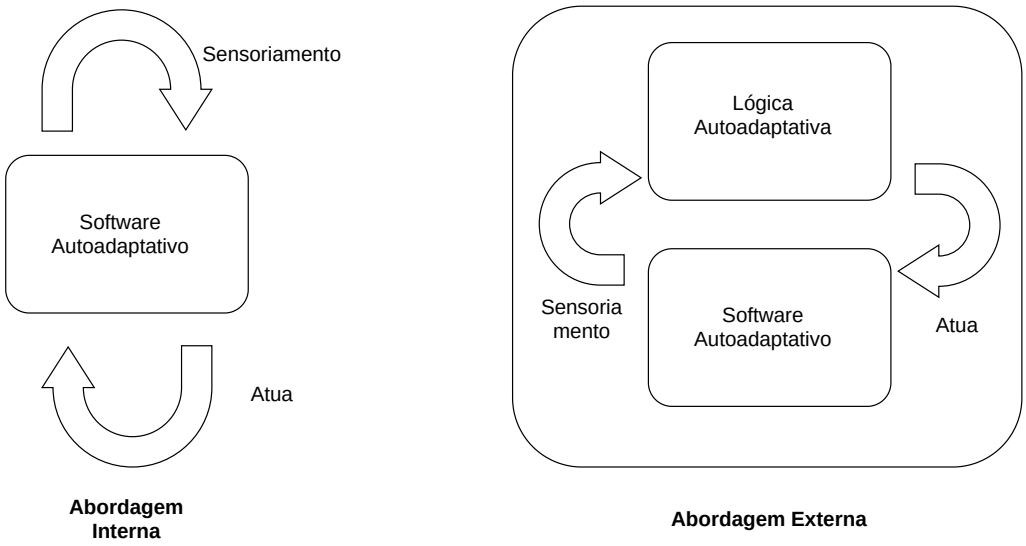
- Autoconfiguração. Caracteriza-se pela capacidade do sistema em se autoconfigurar automaticamente/dinamicamente em respostas às mudanças do sistema em questão.
- Autocura. Caracteriza-se pela capacidade do sistema em encontrar e diagnosticar as exceções do programa. Entende-se por exceção toda operação que não foi prevista para execução. Desse modo, o sistema deve executar ações de antecipação para que erros não sejam gerados. Basicamente, essas ações envolvem tomadas de decisões adequadas que são comportamentos preventivos/reparadores ao implementar essa propriedade. Essa propriedade também pode ser encontrada na literatura como *self-diagnostic* (autodiagnóstico) ou *self-repairing* (autoreparo).
- Auto-otimização. Caracteriza-se pela capacidade do sistema em gerenciar a sua própria performance e recursos computacionais com o propósito de satisfazer seus objetivos. Essa propriedade pode ser também denominada como *self-tuning* ou *self-adjusting* (autoajuste).
- Autoproteção. Caracteriza-se pela capacidade do sistema em detectar e se recuperar de possíveis brechas de segurança. Para isso, é necessário se proteger dos ataques maliciosos e, posteriormente, realizar medidas que extinguem ou mitiguem os efeitos dos ataques.

Nível Primitivo. Este nível possui duas propriedades, a saber: (i) self-awareness (autoconsciência) e (ii) context-awareness (consciência de contexto). A primeira refere-se a capacidade do sistema ter ciência de seus comportamentos e de seus estados (HINCHEY; STERRITT, 2006). Já a segunda, refere-se a capacidade do sistema ter ciência de seu ambiente operacional (contexto) (PARASHAR; HARIRI, 2004).

De acordo com Salehie e Tahvildari (2009), os SaS têm sido desenvolvidos com base em conceitos de duas abordagens, a saber: (i) interna e (ii) externa. Na primeira, o sistema e a lógica de adaptação (ou seja, *Adaptation Engine*) e o software adaptável (ou seja, *Adaptable Software*) estão interligados na mesma camada, sendo caracterizados pelas ferramentas disponíveis na linguagem de programação, como expressões condicionais e exceções. No geral, essa abordagem é carente de recursos, visto que os processos de adaptações estão junto com código do sistema, oferecendo uma fraca escalabilidade e manutenibilidade para os sistemas. No entanto, existem cenários em que essa abordagem pode ser útil quando lidar com adaptações locais. Já na segunda, a lógica de adaptação do sistema está organizada como um mecanismo de adaptação externo em relação ao software adaptável. Essa abordagem é bastante utilizada em sistemas distribuídos e complexos, geralmente compostos por diversos módulos de adaptação. A vantagem de sua

adoção está em fornecer reutilização do código do mecanismo de adaptação, podendo ser customizado para outros sistemas. Na Figura 2 é ilustrada a organização de cada abordagem.

Figura 2 – Abordagens de adaptação



Fonte: Adaptado de (SALEHIE; TAHVILDARI, 2009)

No que diz respeito ao desenvolvimento de SaS, Salehie e Tahvildari (2009) propuseram uma abordagem baseada no modelo 5W1H. Em resumo, essa abordagem visa apoiar a identificação dos requisitos básicos de adaptação com base em seis perguntas, a saber:

- ***Where* (Onde).** Tem por objetivo responder onde será necessário realizar a adaptação. Nesse sentido, é necessário entender como a coleta de dados (ou seja, atributos do sistema) e do ambiente são necessários.

- ***When* (Quando).** Tem por objetivo responder às questões temporais relacionadas a necessidade/possibilidade de realizar adaptações. Nessa perspectiva, questões como *Com que frequência o sistema deve ser alterado?*, *Quando uma alteração deve ser realizada?* e *As adaptações podem ser realizadas sempre que o sistema exige?* são fundamentais para entender quando a adaptação deve ser realizada.

- ***What* (O que).** Tem por objetivo identificar quais atributos ou componentes devem ser alterados através dos mecanismos de adaptação, como também, identificar quais ações devem ser tomadas em cada situação de adaptação. Nessa perspectiva, a adaptação pode ser realizada por meio de alterações nos parâmetros ou nos métodos dos componentes e recursos do sistema.

- ***Why* (Por que).** Tem por objetivo identificar as motivações que implicam na adaptação do sistema. Pode-se dizer que essa questão está relacionada aos fatores de qualidade a

serem implementados no sistema, tais como: (i) eficiência, (ii) robustez, (iii) portabilidade, (iv) confiabilidade, entre outros.

- ***Who* (Quem).** Tem por objetivo identificar o nível de automação e interação humana no sistema. Nesse sentido, é esperado que o sistema tenha o mínimo de intervenção humana, em função de atividades automatizadas que sejam capazes de desempenhar seus propósitos de maneira confiável.

- ***How* (Como).** Questão relacionada ao nível de implementação para adaptação do sistema. Para isso, questões como *Como os componentes adaptáveis devem ser alterados e quais adaptações devem ser realizadas em uma dada condição?* devem ser respondidas.

Bellman *et al.* (2020) comentaram que existem na literatura várias abordagens que podem apoiar o desenvolvimento de SaS, a saber: baseada em agentes, baseada em aprendizado, baseada em arquitetura, baseada em reflexão, e inspirada na natureza. Os autores também comentaram que essas abordagens foram projetadas para atender detalhes específicos que norteiam o desenvolvimento e que, consequentemente, resultam no produto final. A seguir, uma breve descrição de cada abordagem é apresentada.

Baseada em Agentes. Segundo Russell e Norvig (2009), agentes podem ser definidos como componentes de *software* que percebem o ambiente ao seu redor por meio de sensores e realizam mudanças nesse ambiente com base em dados coletados através dos atuadores. Portanto, os agentes podem ser vistos como elementos de software que alcançam seus objetivos de maneira autônoma e cooperativa. Um dos grandes destaques dessa abordagem é a sua descentralização, que facilita o desenvolvimento de sistemas que atuam em ambientes dinâmicos (KRUPITZER *et al.*, 2015).

Baseada em Aprendizado. Este tipo de abordagem é apoiado por modelos matemáticos, que, através de ajustes nas variáveis paramétricas por meio dos dados de entrada, permite que o sistema interprete seu ambiente de operação. Nessa perspectiva, o uso de mecanismos baseados em aprendizado implica em prover ao sistema certas características de auto-organização (*self-organizing*), uma vez que o seu foco é realizar mudanças estruturais no software (KRUPITZER *et al.*, 2015). Abordagens desse tipo tendem a ser mais limitadas e, consequentemente, apresentam problemas de desempenho quando executadas em ambientes complexos e dinâmicos (ALMUTAIRI; ALHARBI, 2017).

Baseada em Arquitetura. Esta é uma das abordagens mais consolidadas na literatura para lidar com incertezas em tempo de execução. Por meio da propriedade *self-awareness*, o software (SaS) pode permanecer ciente de sua organização (ou seja, estrutura e comportamento), conhecendo seus componentes, suas relações e propriedades, e seu estado

corrente de execução. Diante desse contexto, pode-se dizer que um SaS necessita de um gerente autônomo responsável por realizar o monitoramento e adaptação baseado em um conjunto de regras descrito em formato de objetivos de alto nível do sistema (SALEHIE; TAHVILDARI, 2009; WEYNS, 2017). Dessa forma, o principal mecanismo utilizado na comunidade para implementar a propriedade *self-awareness* é o modelo de referência MAPE (do inglês, *Monitor*, *Analyze*, *Plan*, *Execute*) desenvolvido pela IBM (IBM, 2005).

Baseada em Reflexão. Esta abordagem de adaptação permite que o próprio sistema observe seu estado (interno e externo) e, em caso de alguma alteração, modifique seus comportamentos e/ou estruturas em tempo de execução (KRUPITZER *et al.*, 2015). Outros autores caracterizam como a habilidade de um programa raciocinar sobre o próprio comportamento (TOZZI, 2023). A partir desse contexto, as ações principais de reflexão pode ser dividida em (i) introspectivas, quando o sistema observa o seu próprio comportamento, e (ii) intercessão, quando o sistema realiza mudanças a partir do raciocínio ocorrido no próprio sistema por meio dos dados obtidos na atividade anterior (KRUPITZER *et al.*, 2015).

Inspirada na Natureza. Esta abordagem de adaptação é formada por algoritmos, cujo modo de operação é inspirado por características observáveis na natureza. Em geral, esses algoritmos são organizados em quatro categorias, a saber: biológicos, físicos, químicos e sociais (KRUPITZER *et al.*, 2015). Abordagens biológicas traduzem conceitos, fenômenos e comportamentos biológicos em modelos computacionais. Abordagens físicas são inspiradas por processos físicos como, por exemplo, fenômenos originados da termodinâmica, física quântica, entre outros. Abordagens químicas mapeiam reações em modelos que coordenam os componentes do software. Por fim, abordagens sociais traduzem os aspectos sociais das espécies em modelos arquiteturais. De modo geral, as abordagens inspiradas pela natureza implementam a propriedade *self-organizing*.

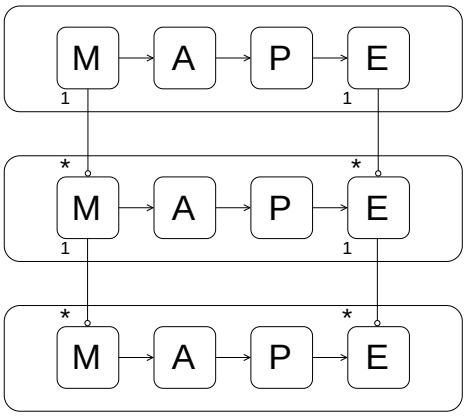
Além das abordagens de desenvolvimento, o projeto de SaS está associado a um conjunto de padrões e boas práticas de engenharia, que provê soluções rápidas baseadas em conhecimentos adquiridos por meio de experiências anteriores. Portanto, quando devidamente empregada, essa prática promove a melhora dos atributos de qualidade do software, como manutenibilidade, reusabilidade, entre outros fatores associados ao software (SaS) (GROWIN; WEYNS; HOLVOET, 2014). A seguir, uma descrição de cada abordagem e de seus respectivos padrões é apresentada.

O *loop* MAPE representa um método consolidado na comunidade científica para a implementação de SaS. Contudo, quando os sistemas se tornam complexos e heterogêneos, a abordagem baseada em um único *loop* torna-se ineficaz para gerenciar as adaptações necessárias. Nesse sentido, múltiplos *loops* podem ser utilizados para gerenciar diferentes partes do

sistema, resultando em um abordagem de adaptação descentralizada (WEYNS *et al.*, 2010). Diante desse contexto, são destacados dois padrões de projeto para mostrar como ocorre a organização arquitetural desse tipo de sistema, a saber: Controle Hierárquico e Compartilhamento de Informações. A seguir, uma descrição de cada padrão é apresentada.

Controle Hierárquico. Padrão de projeto voltado para sistemas distribuídos complexos. Para isso, é necessário considerar a implementação de vários *loops* de controle MAPE para controlar cada funcionalidade em separado. Nesse sentido, vale destacar que a coordenação entre os *loops* é essencial, pois cada *loop* opera em tempos assíncronos e gerencia dispositivos e recursos distintos. Na Figura 3 é ilustrada uma visão geral desse padrão.

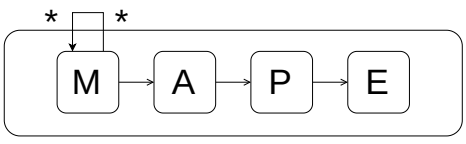
Figura 3 – Controle Hierárquico



Fonte: Adaptado de (WEYNS *et al.*, 2010)

Compartilhamento de Informações. Em certas ocasiões, pode-se encarregar as funcionalidades (nós) de um sistema a um *loop* MAPE. Contudo, em certos ambientes de aplicações, é cabível que cada nó se adapte localmente sem a participação ativa de outro, não sendo necessária a coordenação das atividades de adaptação. Nesse sentido, através do uso dos dados coletados de outros *loops* MAPE com a finalidade de ajustar a melhor adaptação possível, cada instância MAPE possui ciência do estado do sistema. Conforme é ilustrado na Figura 4, cada instância pode comunicar com múltiplas instâncias, através da ligação existente entre elas na fase de monitoramento.

Figura 4 – Compartilhamento de Informações



Fonte: Adaptado de (WEYNS *et al.*, 2010)

Na literatura existe um conjunto de outros padrões que podem ser utilizados no desenvolvimento de SaS em conjunto com uma abordagem de desenvolvimento ou de maneira independente (ou seja, a depender da natureza do sistema alvo). Assim, por motivos de objetividade e espaço, detalhes desses padrões não serão apresentados nesta seção. No entanto, a abordagem baseada em agentes será considerada para exemplificar esse cenário. Para esse tipo de abordagem são recomendados dois modelos arquiteturais que visam otimizar a interação entre agentes em diversos contextos de aplicabilidade, a saber: *Agent Service Patterns* e *Virtual Environment Pattern*. Detalhes sobre esses padrões podem ser obtidos em Mueller, Braun e Kowalczyk (2005).

2.4 Sistemas ciber-físicos

Nos últimos anos, um novo paradigma caracterizado pela computação ubíqua surgiu na literatura, cujo foco é a interação humano-máquina de maneira imperceptível (CASARIN, 2016). Nessa perspectiva, diversos elementos do ambiente tornam-se uma fonte valiosa de informação para que se possa interpretar reações mais precisas dos sistemas (SILVA, 2015). Dentre os sistemas que permitem tal interação, destacam-se os CPS, cujos elementos computacionais também interagem com entidades físicas. Sistemas Ciber-Físicos é termo originado em 2006 pela NSF (*National Science Foundation*), cujo propósito é a integração entre sistemas computacionais e redes de comunicação, representando um novo desafio para, no mínimo, três grandes áreas: engenharia, computação e física. Na literatura, é possível encontrar várias definições sobre CPS, as quais podem estar relacionadas com sistemas híbridos[1], tempo de resposta, componentes integrados, controles, entre outros. Em síntese, CPS podem ser definidos pela integração de entidades computacionais e físicas que se comunicam por meio de uma rede de comunicação (SÁNCHEZ *et al.*, 2018). A seguir, os principais conceitos e definições relacionados ao contexto de CPS são apresentados.

Segundo Rajkumar *et al.* (2010), CPS são descritos como "sistemas físicos, cujas operações são monitoradas, coordenadas, controladas e integradas por um núcleo de computação e comunicação". Já Lee (2006) caracteriza CPS como a integração de computação com processos físicos. Em paralelo, Marwedel (2010) define CPS como sistemas embarcados integrados a seus ambientes físicos. Para Li *et al.* (2012), os CPS têm como base o suporte à tomada de decisão em tempo real, a qual depende diretamente de fatores como (i) segurança, que se refere a garantia que os impactos das operações realizadas estejam dentro do limite desejado, e (ii) previsibilidade, que é caracterizada pela capacidade do sistema agir diante das condições não previstas por meio de mecanismos de autoadaptação. O estudo de Martins e Martins (2015)

[1] Sistemas híbridos são caracterizados por integrar a dinâmica contínua do mundo físico e a lógica discreta do mundo cibernético (SILVA, 2015)

reporta uma visão mais abrangente sobre CPS, caracterizando-os pela presença dos seguintes itens: (i) subsistemas que possibilitam interação com o mundo físico; (ii) sensores que permitem obter informações do mundo físico para processamento e tomada de decisões; (iii) mecanismos de comunicação entre o meio físico e os subsistemas a fim de gerenciar determinadas condições do ambiente/sistema; e (iv) mecanismos de computação das informações, cujos dados coletadas do ambiente são encaminhados aos subsistemas de tomada de decisão para que comandos sejam enviados para os atuadores.

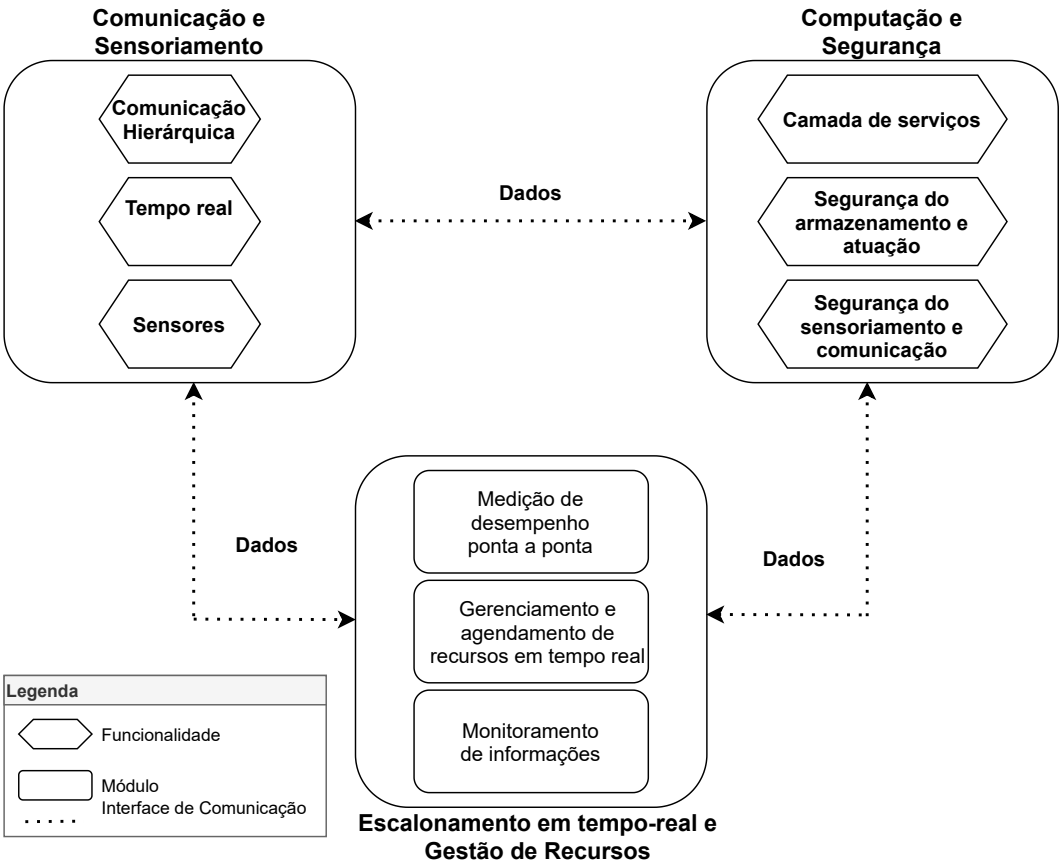
Muccini, Sharaf e Weyns (2016), Musil *et al.* (2017) apresentaram diversos domínios em que CPS podem ser aplicados, a saber: Automação Industrial, Robótica, Transporte, Infraestrutura em grande escala, Militar, e Saúde. Dependendo do domínio de aplicação, a denominação de tais sistemas pode ser alterada e/ou estendida para que as novas necessidades desse sistema/-domínio sejam atendidas. Por exemplo, Kagermann *et al.* (2013), PISCHING (2008) reportaram que CPS aplicados ao domínio de manufatura podem ser definidos como CPPS (do inglês, *Cyber-Physical Production Systems*), pois esses sistemas são compostos por maquinários e vários outros sistemas (ou seja, estoque, distribuição, produtivos, entre outros). No que diz respeito ao funcionamento, os CPPS devem ter a capacidade de executar operações autônomas como trocar informações, disparar eventos e controlar processos de maneira independente. Os Sistemas Ciber-Físicos Médicos (do inglês, *Medical Cyber Physical Systems* – MCPS), voltados para o domínio de saúde, podem ser classificados como um outro exemplo nessa direção. Esses sistemas apresentam interação com meio físico (por exemplo, mecanismos de monitoramento de pacientes) para que decisões possam ser tomadas. Pela natureza crítica, vale ressaltar que os MCPS estão associados a vários atributos de qualidade (por exemplo, tempo de resposta, segurança, confiabilidade).

Na comunidade científica podem ser encontradas também definições que relacionam CPS, IoT e Sistemas Embarcados (SE), sendo destacados, em geral, os pontos de convergentes e divergentes entre as áreas de pesquisa. Segundo Leitão, Colombo e Karnouskos (2016), CPS podem ser considerados como uma evolução de SE, cujo precursor tem como foco os elementos computacionais e o sucessor a interação entre elementos computacionais e dispositivos físicos. Em contrapartida, a relação existente entre CPS e IoT é tratada como opcional, pois esta tem sido caracterizada como uma rede de interação e integração entre elementos computacionais (ou seja, rede de sensores, atuadores e dispositivos), formando novos sistemas e serviços.

De acordo com Bellman *et al.* (2020), Rawat, Rodrigues e Stojmenovic (2015), o desenvolvimento de CPS apresenta diversos desafios em diferentes etapas do ciclo de desenvolvimento desse tipo de sistema, que podem ser sintetizados pela interação e complexidade dos ambientes de operação, os diversos dispositivos que os compõem, entre outros. Dessa forma, nota-se que a comunidade acadêmica tem se mobilizado para contornar esses desafios, propondo soluções que possam aliviar a pressão em diferentes etapas. Nesse sentido, pode-se citar o desenvolvimento

de modelos arquiteturais que possam servir como um guia para o desenvolvimento de aplicações para domínios específicos ou similares. Dessa forma, nota-se que alguns modelos arquiteturais têm sido propostos nos últimos anos, os quais se diferenciam pelos mecanismos que implementam e, principalmente, pelas propostas de domínio de aplicação. Esta seção destaca um modelo de referência baseado em serviços com suporte a segurança e gerenciamento de recursos. Na Figura 5 é ilustrado um modelo de arquitetura para CPS voltado para sistemas de monitoramento de saúde denominado $CPeSC^3$ (WANG *et al.*, 2011).

Figura 5 – Modelo arquitetural para área de saúde



Fonte: Adaptado de (WANG *et al.*, 2011)

Como pode-se observar (Figura 5), esse modelo foi organizado em três módulos, a saber: Comunicação e Sensoriamento, Computação e Segurança, e Escalonamento em tempo-real e Gestão de recursos. Em resumo, os sensores podem estar acoplados a uma pessoa ou presentes no ambiente físico. Os dados coletados podem revelar o estado de saúde do paciente, além de atividades humanas para os serviços de saúde. Por exemplo, as imagens capturadas pelas câmeras são utilizadas para identificar os movimentos de uma pessoa, sendo possível identificar diversas atividades, a saber: andar, sentar, caminhar, entre outras. Maiores detalhes sobre esse modelo podem ser obtidos em Wang *et al.* (2011).

Nos últimos anos, tem-se notado alguns desafios frente aos modelos arquiteturais tradici-

onais adotados para o desenvolvimento de CPS. Como exemplo, pode-se destacar a incorporação de novos interesses como segurança, poder de processamento computacional, e capacidade armazenamento para grande volume de dados na atividade de monitoramento (GUO; ZENG, 2019; RAWAT; RODRIGUES; STOJMENOVIC, 2015). Diante do exposto, visando melhorar as capacidades computacionais e contornar as adversidades encontradas, a comunidade científica, juntamente com iniciativas privadas, têm buscado integrar diversos mecanismos e abordagens para apoiar o desenvolvimento de CPS para diferentes domínios de aplicação. Uma alternativa é combinar recursos de outras áreas (por exemplo, computação em nuvem, aprendizado de máquina e IoT) aos CPS para ampliar sua capacidade de processamento, tomada de decisão e comunicação com o ambiente físico. Outra alternativa é combinar os mecanismos de autoadaptação aos CPS.

De acordo com Muccini, Sharaf e Weyns (2016), Musil *et al.* (2017), CPS podem ser caracterizados como sistemas complexos pela forte interação com o ambiente operacional. Contudo, o monitoramento constante gera um novo conjunto de desafios a serem superados por esses sistemas. Dentre eles, destacam-se a incerteza de requisitos em tempo de execução, que pode ser caracterizada pela dificuldade em resolver situações não previstas na fase de projeto. Ainda no cenário de incertezas, pode-se destacar também as mudanças ocorridas no ambiente de execução, as quais não fazem parte do projeto inicial do sistema. Diante do exposto, visando contornar as situações de incertezas e/ou imprevisibilidade, recursos de autoadaptação têm sido incorporados ao desenvolvimento de CPS, dando origem a uma nova classe de sistemas denominada Sistemas Ciber-Físicos Autoadaptativos.

Muccini, Sharaf e Weyns (2016) comentaram que a comunidade científica tem integrado diversos mecanismos autoadaptativos aos CPS. Dentre eles, pode-se destacar o *loop* MAPE, a propriedade *self-organization*, agentes e reflexão. Contudo, a integração de recursos de adaptação não está limitada ao uso de um único recurso. Em geral, SaS são sistemas complexos quanto aos componentes de projeto e necessitam de boas práticas de engenharia e modelos arquiteturais. Por fim, os autores supracitados mencionaram que os principais interesses na implementação de recursos de autoadaptação localizam na camada de aplicação, *middleware*, comunicação e serviço (MUCCINI; SHARAF; WEYNS, 2016). Maiores detalhes sobre os Self-CPS podem ser visualizados no Seção 3.2.

2.5 Considerações finais

Neste capítulo foram apresentados um panorama geral dos principais conceitos que permeiam o desenvolvimento deste projeto de mestrado acadêmico. Inicialmente, foram apresentados conceitos sobre arquitetura de software, modelo de referência, arquitetura de referência e arquitetura concreta. Em seguida, uma visão geral sobre sistemas *e-Health* foi reportada, vi-

sando apresentar um breve panorama desse tipo de sistema, assim como as principais lacunas existentes. No que concerne os CPS, destacam-se um modelo de arquitetura para CPS voltado para sistemas de monitoramento de saúde denominado $CPeSC^3$ e as diversas áreas de pesquisa, como Aprendizado de Máquina, Computação em Nuvem e Internet das Coisas. Em relação aos SaS, destacam-se as abordagens baseadas em arquitetura e as propriedades self-*, como *self-organizing* e *self-configuring*. Essas propriedades permitem que os CPS sejam capazes de antecipar um erro e corrigi-los sem (ou com o mínimo de) intervenção humana, que depende do nível de adaptação implementado no sistema. Em linhas gerais, essas tecnologias em conjunto nos revelam que os CPS podem ser aplicados em diversos ambientes e em diferentes setores da sociedade.

3 MODELOS E ARQUITETURAS DE REFERÊNCIA PARA SELF-CPS

O objetivo do capítulo é apresentar o estado da arte sobre Self-CPS, uma classe de sistemas que envolve conceitos de diversas áreas de pesquisa, como Engenharia e Arquitetura de Software, Inteligência Artificial, Teoria de Controle, Redes de Computadores, entre outras. Com o propósito de apresentar um panorama geral sobre modelos de referência e arquiteturas de referência para Self-CPS que têm sido projetados pela comunidade científica, uma pesquisa sistemática foi conduzida utilizando as principais bases de publicação (KITCHENHAM; DYBA; JORGENSEN, 2004; KITCHENHAM; CHARTERS, 2007; KITCHENHAM *et al.*, 2010; PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Inicialmente, na Seção 3.1 é reportado o levantamento dos estudos secundários sobre Self-CPS, sendo destacadas principais contribuições e objetos de investigação de cada estudo. Na Seção 3.2 são apresentados e discutidos os trabalhos do mapeamento da literatura sobre o tema de pesquisa deste trabalho. Por fim, na Seção 3.3 são reportadas as considerações finais deste capítulo.

3.1 Estudos secundários sobre Self-CPS

O objetivo desta seção é apresentar como a busca por estudos secundários sobre Self-CPS foi conduzida. Em seguida, uma discussão de cada estudo selecionado é reportada.

3.1.1 Planejamento de pesquisa

Considerando o contexto de pesquisa, uma estratégia de busca foi definida, a qual pode ser sintetizada nos seguintes itens:

Critérios de seleção de fontes. Foram utilizados os seguintes critérios para encontrar os estudos secundários, a saber: (i) base de publicações que são regularmente atualizadas; (ii) disponibilidade de conteúdo; (iii) qualidade dos resultados retornados pela busca; e (iv) versatilidade para exportar os resultados, ou seja, arquivos de referência (`.bib`) e de conteúdo (`.pdf`). Estes critérios foram definidos por Dieste e Padua (2007);

Lista de fontes. As bases selecionadas do mapeamento dos estudos secundários sobre Self-CPS são apresentadas na Tabela 1. De acordo com Dyba, Dingsoyr e Hanssen (2007), Kitchenham e Charters (2007), Kitchenham, Dyba e Jorgensen (2004), Kitchenham e Charters (2007), Kitchenham *et al.* (2010), Petersen, Vakkalanka e Kuzniarz (2015), essas

bases são eficientes para conduzir esse tipo de estudo na área de Engenharia de Software. Além disso, a base Scopus foi adicionada, pois é considerada umas das maiores bases de citações (KITCHENHAM; DYBA; JORGENSEN, 2004; KITCHENHAM; CHARTERS, 2007);

Tabela 1 – Bases de publicações selecionadas

Base	**Endereço**
ACM Digital Library	<https://dl.acm.org>
IEEE Xplore	<https://ieeexplore.ieee.org>
ScienceDirect	<https://www.sciencedirect.com>
Scopus	<https://www.scopus.com>
Springer	<https://link.springer.com>
Web of Science	<http://webofknowledge.com>

Idioma dos estudos. Somente estudos secundários escritos em inglês serão considerados por ser o idioma mais comum em artigos científicos;

Palavras-chaves. *Cyber-Physical Systems*, *Systematic Review* e *Self-adaptive Systems* foram os termos selecionados para obter o levantamento dos estudos secundários com seguintes sinônimos:

- Cyber-Physical Systems: CPS, *cyber physical*, *cyber-physical*, *cyberphysical*, *embedded*.
- Systematic Review: SLR, SMS, *survey*, *systematic literature review*, *systematic mapping*, *systematic review*.
- Self-adaptive Systems: *self-adaptable*, *self-adaptation*, *self-adapting*, *self-adaptive*.

String de pesquisa. Foi utilizado operador booleano `OR` para vincular os sinônimos relacionados. Esses termos foram combinados usando o operador booleano `AND` para compor a *string* de busca do mapeamento, como apresentado na Tabela 2.

Tabela 2 – *String* de busca

```
(SLR OR SMS OR survey OR systematic literature review OR systematic
mapping OR systematic review)
```

AND

```
(CPS OR cyber physical OR cyber-physical OR cyberphysical OR embedded)
```

AND

```
(self-adaptable OR self-adaptation OR self-adapting OR self-adaptive)
```

3.1.2 Condução da pesquisa

Definir os critérios de inclusão (do inglês, *Inclusion Criteria* – IC) e exclusão (do inglês, *Exclusion Criteria* – EC) é um importante componente para trabalhos de revisão formal da literatura. Esses critérios permitem incluir estudos relevantes para responder às questões de pesquisa elaboradas para esse tipo de trabalho, além de excluir estudos que não as respondem. Um estudo é incluído se atender a um ou mais ICs e excluído se atender a um ou mais ECs. Assim, os critérios deste mapeamento são listados abaixo:

- **IC1**: O estudo reporta algum tipo de revisão (ou seja, revisão sistemática, mapeamento sistemático, ou *survey*) da literatura sobre Self-CPS;
- **EC1**: O estudo não aborda uma revisão sobre Self-CPS;
- **EC2**: O estudo secundário é escrito em um idioma diferente do inglês;
- **EC3**: O estudo secundário apresenta somente um resumo ou não está disponível em sua totalidade;
- **EC4**: O estudo secundário é um trabalho anterior desenvolvido pelo mesmo autor. Nesse caso, apenas o estudo mais recente ou mais completo é considerado.

As buscas nas bases de publicação, conforme fontes apresentadas na Tabela 1, foram efetuadas em 12/01/2021, resultando em 1.270 estudos. Em seguida, para selecionar o conjunto final de estudos foi necessário realizar uma filtragem composta pelas seguintes etapas: (i) remoção das entradas sem autores; (ii) inserção da *tag "source"* para identificação de cada base de publicação; (iii) remoção dos trabalhos que não são estudos secundários; (iv) remoção de estudos duplicados; e (v) aplicação dos critérios de inclusão e exclusão. Após condução das etapas supracitadas, sete estudos secundários que abordam propriedades autoadaptativas em CPS foram selecionados. Na Tabela 3 são apresentadas as referências simplificadas para cada estudo, sendo que na coluna ID são listados os identificadores, na coluna Referência são listados os autores e os títulos do estudo, e na coluna Ano são apresentados o ano de publicação de cada estudo. Por fim, vale destacar que os estudos foram ordenados pelo título.

3.1.3 Discussão

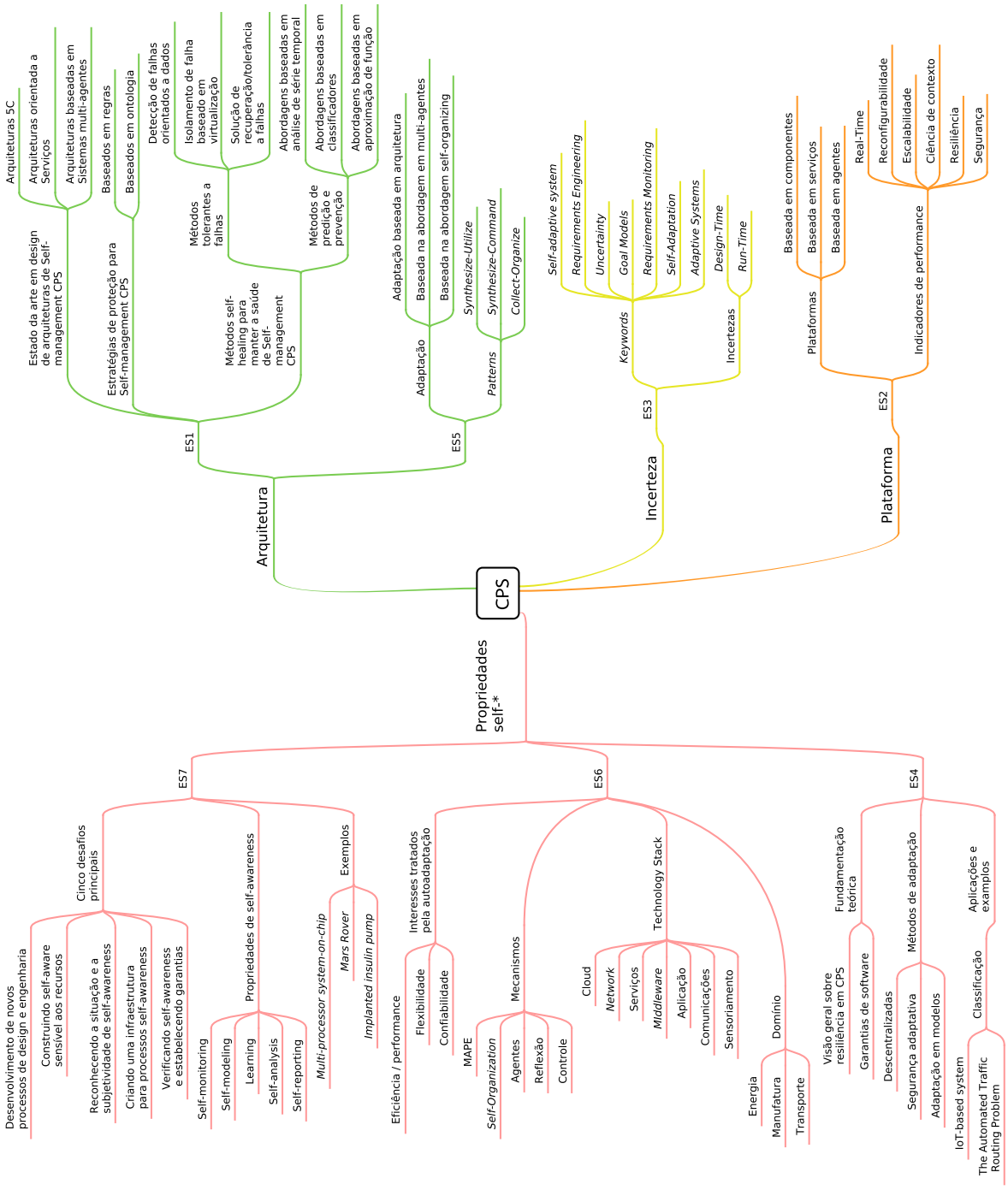
O objetivo desta seção é reportar uma síntese de cada estudo apresentado na Tabela 3, contextualizando seu foco de pesquisa e destacando suas principais contribuições. Na Figura 6 é ilustrado um mapa mental que visa categorizar tais estudos em relação a quatro categorias, a saber: arquitetura, incertezas, plataformas de adaptação, e propriedades *self-**.

Tabela 3 – Listagem dos estudos coletados ordenados por título.

ID	Referência	Ano
ES1	Zhou, P. and Zuo, D. and Hou, K.M. and Zhang, Z. and Dong, J. and Li, J. and Zhou, H., ***A comprehensive technological survey on the dependable self-management CPS: From self-adaptive architecture to self-management strategies.***	2019
ES2	Sun, Yuan and Yang, Gang and Zhou, Xing-she, ***A survey on run-time supporting platforms for cyber physical systems.***	2017
ES3	Brings, J. and Salmon, A. and Saritas, S., ***Context uncertainty in requirements engineering: Definition of a search strategy for a systematic review and preliminary results.***	2015
ES4	Bennaceur, Amel and Ghezzi, Carlo and Tei, Kenji and Kehrer, Timo and Weyns, Danny and Calinescu, Radu and Dustdar, Schahram and Hu, Zhenjiang and Honiden, Shinichi and Ishikawa, Fuyuki and Jin, Zhi and Kramer, Jeffrey and Litoiu, Marin and Loreti, Michele and Moreno, Gabriel A. and Müller, Hausi A. and Nenzi, Laura and Nuseibeh, Bashar and Pasquale, Liliana and Reisig, Wolfgang and Schmidt, Heinz and Tsigkanos, Christos and Zhao, Haiyan, ***Modelling and Analysing Resilient Cyber-Physical Systems.***	2019
ES5	Musil, A. and Musil, J. and Weyns, D. and Bures, T. and Muccini, H. and Sharaf, M., ***Patterns for self-adaptation in Cyber-Physical Systems.***	2017
ES6	H. Muccini and M. Sharaf and D. Weyns, ***Self-Adaptation for Cyber-Physical Systems: A Systematic Literature Review.***	2016
ES7	Bellman, K. and Landauer, C. and Dutt, N. and Esterle, L. and Herkersdorf, A. and Jantsch, A. and TaheriNejad, N. and Lewis, P. R. and Platzner, M. and Tammemäe, K., ***Self-Aware Cyber-Physical Systems.***	2020

Zhou *et al.* (2019) (ES1) conduziram um estudo sobre CPS autogerenciáveis confiáveis. Para isso, os autores conduziram uma investigação na literatura, que permitiu estabelecer um panorama sobre arquiteturas autoadaptativas, estratégias de adaptação, soluções *self-healing* para *Self-management* CPS (SCPS), e abordagens para garantia de confiabilidade desse tipo de sistema baseadas no método V&V (Validação e Verificação). Diante desse contexto, a principal contribuição deste estudo foi apresentar um arcabouço completo da área com a finalidade de orientar futuras pesquisas e melhorar a questão do autogerenciamento em CPS. Em síntese, as arquiteturas mais utilizadas podem ser categorizadas em 5C, orientadas a serviços e baseadas em Sistemas Multi-Agentes. No que diz respeito ao uso de adaptação, estratégias baseadas em regras e/ou ontologias podem ser destacadas. Contudo, como os SCPS lidam com diversos cenários de adaptação do sistema, a base de conhecimento poderá aumentar de maneira significativa, exigindo um grande poder computacional por parte desses sistemas para classificação de algum tipo de anomalia e recomendação de solução para contornar a adversidade identificada. Visando minimizar os efeitos e causas dessa limitação, os autores recomendam abordagens

Figura 6 – Categorização dos estudos



Fonte: Elaborada pelo autor

de aprendizado por reforço ou aprendizado de máquina. Por fim, para garantir a "saúde" do SCPS métodos derivados de soluções *self-healing* são sugeridos para atuar em dois cenários: (i) tolerância a falhas, e (ii) predição e prevenção de anomalias. Para o primeiro, métodos como detecção de falhas orientados a dados, isolamento de falha baseado em virtualização e solução de recuperação/tolerância a falhas podem ser utilizados. Já para o segundo, abordagens baseadas em análise de série temporal, em classificadores, ou em aproximação de função são sugeridas.

Em conclusão, o estudo destaca e explora o uso de técnicas V&V em conjunto com o *framework Model@run.time* como tendência para a área de SCPS.

No trabalho de Sun, Yang e Zhou (2017) (ES2) foram investigadas as plataformas de desenvolvimento para CPS com suporte em tempo de execução. Dessa forma, a contribuição deste trabalho é estudar as plataformas existentes com a finalidade de avaliar os esforços realizados pela comunidade diante dessas ferramentas. Diante desse contexto, os autores categorizam as plataformas em três categorias, a saber: baseadas em componentes, baseadas em agentes, e baseadas em serviços. Para cada categoria são abordados uma visão geral, explorando os conceitos e mecanismos/técnicas de adaptação que foram implementados nesses paradigmas de programação pela comunidade científica. Em seguida, comparações das plataformas em relação a dois fatores foram realizadas, a saber: (i) abordagens de *design*, explorando as interações e composições entre as unidades estruturais (por exemplo, componentes, agentes, e serviços); e, (ii) suporte às propriedades não funcionais, expondo os pontos positivos e negativos das plataformas em relação aos conceitos relacionados a tempo real, reconfigurabilidade, escalabilidade, ciência de contexto, resiliência e segurança. O estudo desenvolvido pelos autores mostrou que a comunidade apresenta um desinteresse significativo em abordar as propriedades "tempo real", "segurança" em suas plataformas de desenvolvimento, enquanto que as propriedades restantes citadas apresentam-se sólidas na área, pois há um grande volume de estudos que abordam as mesmas. Em síntese, os autores recomendam as abordagens multi-agentes para lidar com autonomia, suporte em tempo real e ambientes dinâmicos; abordagens baseadas em serviços para lidar com interoperabilidade e ambientes dinâmicos; e abordagens baseadas em componentes para lidar com suporte em tempo real. Como trabalhos futuros, os autores destacaram que as seguintes lacunas de pesquisa acerca do desenvolvimento de plataformas, a saber: (i) implementação de técnicas da Engenharia de Software para averiguar o funcionamento correto do CPS; (ii) suporte para aplicações *human-in-the-loop*, pois, em certos contextos, os humanos são vistos como peça fundamental na questão dos *feedback loops* e, dessa forma, em algum momento, será necessário o auxílio humano para realizar certa ação; e, por fim, (iii) adição de mecanismos de monitoramento *runtime* das tarefas em um CPS, visando aprimorar a questão da resiliência e segurança nas plataformas.

Em Brings, Salmon e Saritas (2015) (ES3), os autores realizaram uma revisão sistemática da literatura com o objetivo de investigar as abordagens que lidam com as incertezas no contexto de CPS e SaS. Para isso, os autores analisaram manualmente estudos publicados entre 2010 e 2014 em duas conferências e um periódico, a saber: *International Requirements Engineering Conference*, *International Working Conference on Requirements Engineering: Foundation for Software Quality International Requirements Engineering Conference*, e *Requirements Engineering Journal*. Por se tratar de um estudo em estágio inicial, os resultados preliminares reportam poucas descobertas sobre o tema de pesquisa investigado. Deste modo, os autores mapearam

como as questões de incertezas foram lidadas durante as fases do ciclo de desenvolvimento de software, assim como os principais termos e sinônimos mais utilizados neste contexto de pesquisa. Os autores também identificaram que a maioria das abordagens de adaptação lidam com as incertezas na fase de *design* do software e poucas na fase de *runtime*.

No trabalho de Bennaceur *et al.* (2019) (ES4) foi conduzido um estudo sobre resiliência em CPS. Dessa forma, a contribuição deste trabalho foi apresentar um panorama geral da área, elencando os principais desafios e futuras direções. Primeiramente, os autores abordaram as principais questões relacionadas aos CPS, tais como: sua natureza multidisciplinar, as garantias que devem ser asseguradas para promover o funcionamento correto do software, e a escala/escopo dos ambientes de aplicação. Em seguida, visando elencar os principais conceitos do desenvolvimento de CPS resilientes, os autores recomendaram a utilização de mecanismos de adaptação descentralizadas, de preferência por meio de abordagens baseadas em modelos e mecanismos de *self-protecting* visando proteger as vulnerabilidades dos dispositivos que podem ser exploradas e os dados coletados pelo sistema. Por fim, os autores elaboraram um esquema de classificação para os principais tipos de aplicações CPS. Tal esquema tem por objetivo identificar as principais características do sistema, a saber: (i) domínio, como *healthcare*, transporte, e alimentos; (ii) atributos de qualidades, como autoadaptação e confiabilidade (segurança e privacidade); e (iii) velocidade de mudanças no ambiente (lentas, moderadas, e rápidas). Os autores exemplificaram o uso do esquema de classificação por meio de dois exemplos (*The Automated Traffic Routing Problem (ATRP)* e *IoT-based ecosystem (FmFm)*), ambos advindos do catálogo SEAMS (do inglês, *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*).

No estudo de Musil *et al.* (2017) (ES5), os autores elaboraram uma revisão de estudos sobre CPS que abordam a aplicação de diferentes mecanismos de autoadaptação sobre diversas camadas do software. Portanto, a contribuição deste trabalho foi apresentar um mapeamento sobre Self-CPS que elencou e comparou os principais mecanismos de adaptação. A partir desse mapeamento, os autores sumarizaram e exploraram os principais métodos de adaptação contemplados no contexto da pesquisa por eles conduzida, a saber: adaptação baseada em arquitetura, abordagem baseada em multi-agentes, e abordagem baseada na propriedade *self-organizing*. Com base nos resultados, uma síntese dos padrões de *design* dos mecanismos de adaptação adotados foi elaborada, a saber: *Synthesize-Utilize*, *Synthesize-Command* e *Collect-Organize*. Em resumo, esses padrões criam uma arquitetura voltada para a adaptação, através do uso de mecanismos como *self-organizing*, sistemas multi-agentes e MAPE. Para tal, o sistema é organizado em cinco camadas, a saber: *Application Layer*, *Service Middleware Layer*, *Communication Layer*, *Proxy Layer* e *Physical Layer*. Por fim, os autores abordaram o conceito de Sistema de Inteligência Coletiva (do inglês, *Collective Intelligence Systems* – SIC) aplicado a CPS e CPPS. Em síntese, segundo Musil, Musil e Biffl (2014), SIC é um sistema multi-agente sociotécnico

caracterizado por utilizar a inteligência coletiva dos atores humanos através de um ambiente *Web* voltado para o compartilhamento, recuperação e distribuições das informações de uma maneira eficiente. Como exemplo, pode-se citar *Youtube*, *Wikipedia*, *Facebook*, entre outros. Por meio do conhecimento adquirido pelo mapeamento, os autores destacam três vantagens relacionadas a integração dos SIC a CPS, a saber: (i) melhora na capacidade raciocínio/decisão, devido ao fato da integração dos humanos no processo de *feedback loop*; (ii) melhora na capacidade de troca de informações entre humanos, pois a inteligência coletiva desses sistema facilita a integração e comunicação do conhecimento de várias áreas, por exemplo, a comunicação entre equipes da Engenharia de Software, Elétrica e Mecânica no desenvolvimento de um CPS; e (iii) melhora nas interações máquina-máquina, que podem ser alcançadas por meio da integração de mecanismos de organização inspirados pela natureza ou por *self-organizing*.

Em Muccini, Sharaf e Weyns (2016) (ES6), os autores conduziram uma revisão sistemática da literatura, utilizando quatro bases científicas, a saber: IEEE, *Xplore Digital Library*, ACM *Digital Library*, *SpringerLink* e *ScienceDirect*. O foco dessa revisão foi estabelecer um panorama, norteado por quatro questões de pesquisa, sobre as abordagens *self-adaptation* existentes para CPS. Dessa forma, a contribuição desse estudo foi analisar como a adaptação tem sido conduzida em CPS, quais problemas foram solucionados, quais métodos foram utilizados, e como as soluções foram avaliadas. Diante da investigação realizada, verifica-se que os principais mecanismos de *feedback loop* utilizados, em ordem decrescente, são: MAPE, *self-organisation*, agentes, reflexão e controle. 36% dos estudos analisados reportaram ter utilizado a combinação de mecanismos de adaptação, sendo agentes e MAPE a combinação mais utilizada, seguida da reflexão e *self-organizing*. Em relação aos domínios de aplicação, energia (domínio relacionado ao contexto de produção, distribuição e otimização de energia), manufatura e transporte são os principais domínios beneficiados pela presença dos CPS. É verificado, também, que eficiência/performance, flexibilidade e confiabilidade são os principais atributos de interesse na implementação dos mecanismos de *feedback loop*. Por fim, os autores concluem que a adaptação em CPS não está relacionada a uma única camada de software. Dessa forma, a comunidade científica tem reportado seu interesse em soluções que envolvem múltiplas camadas, que podem combinar diversos mecanismos de adaptação. Como desafios e/ou trabalhos futuros são apontados alguns pontos chaves, a saber: (i) coordenar os mecanismos de adaptação dentro e entre as camadas; (ii) garantir a consistência de adaptação em todo o sistema; e (iii) mapear as adaptações necessárias nos mecanismos existentes e nas camadas do software.

Por fim, em Bellman *et al.* (2020) (ES7) foi conduzido um estudo com o foco na propriedade de autoconsciência (do inglês, *self-aware*) aplicado aos CPS. Assim, a contribuição deste trabalho foi elencar as dificuldades enfrentadas na integração da autoconsciência e discutir soluções e/ou alternativas para minimizá-las. Primeiramente, foram expostos os desafios enfrentados no desenvolvimento de CPS a partir de três exemplos, a saber: *Heterogeneous Mul-*

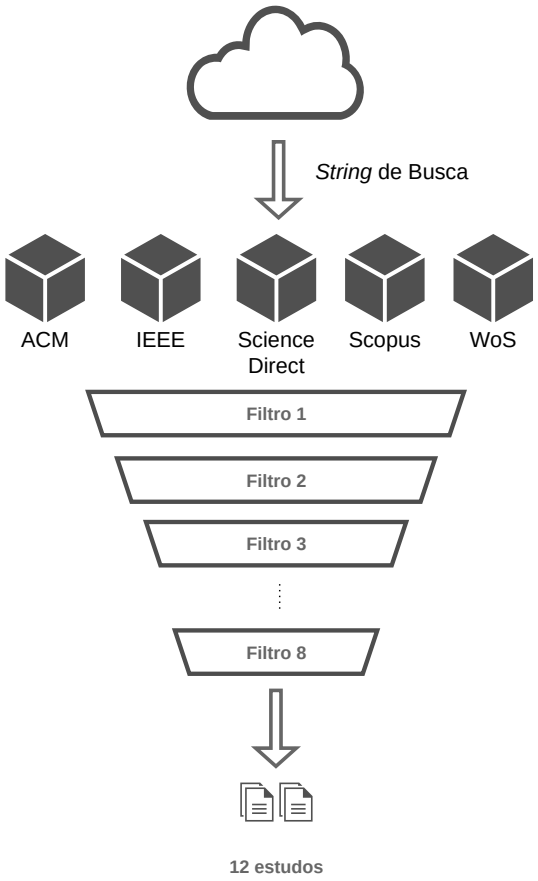
tiprocessor System-on-Chip, *NASA Mars Rover Curiosity* e *Wearable Medical Devices*. Diante desse contexto, os principais desafios podem ser destacados em três tópicos: ambientes dinâmicos, dificultando o sistema ter ciência do ambiente como um todo, diminuindo as chances de sucesso nas ações do sistema; situação e subjetividade, pois a partir do contexto do ambiente e os dados coletados, a interpretação da ação a ser realizada pode ser diferente da real ação a ser feita; e limitações computacionais e confiabilidade, devido aos recursos serem limitados ao poder computacional, tamanho, custo de energia e serem passíveis de erros. Em seguida, embasados nos resultados dessa revisão, os autores destacam cinco propriedades essenciais para o desenvolvimento de um sistema autoconscientes: *self-monitoring*, *self-modeling*, *learning*, *self-analysis*, e *self-reporting*. Por último, os autores sumarizaram e discutiram os principais desafios encontrados pela comunidade ao desenvolver *self-aware* CPS, a saber: (i) processos de Engenharia de *Software*, pois as técnicas tradicionais não levam em consideração a integração de elementos de software autoconscientes; (ii) implementação de componentes *self-aware*, devido ao limitado poder computacional dos recursos do sistema; (iii) reconhecimento do ambiente, pois são necessárias técnicas e ferramentas para os CPS raciocinarem acerca da decisão correta com base no conhecimento adquirido pelo sistema; (iv) infraestruturas para os processos *self-aware*, as quais podem ser minimizadas pela criação de novas plataformas e componentes para facilitar o desenvolvimento; e, por fim, (v) estabelecer garantias para certificar o funcionamento correto do software, que pode ser solucionada através do uso de técnicas V&V desenvolvidas especialmente para o contexto da autoconsciência.

Portanto, diante do exposto, uma visão geral sobre os principais estudos secundários no contexto de Self-CPS foi apresentada nesta seção. Em geral, os autores dos estudos propuseram realizar revisões da literatura envolvendo a integração de diversas arquiteturas, *frameworks* e métodos adaptativos. Dessa forma, acredita-se que os interesses que a comunidade tem explorado, além dos tópicos que demandam estudos mais profundos, foram aqui reportados. Em resumo, embasados nesses estudos pode-se concluir que a comunidade necessita de um período de tempo significativo para que a integração de propriedades autoadaptativas, como métodos baseados na área de Inteligência Artificial, em CPS estejam consolidadas (ZHOU *et al.*, 2019). Embora alguns mecanismos e arquiteturas encontram-se em estágio de maturidade mais consolidadas na comunidade, tais como o *loop* MAPE, arquiteturas multi-agentes, e arquiteturas orientadas a serviços, a área de Self-CPS ainda carece de modelos arquiteturais, modelos de referência, arquiteturas de referência, *frameworks* e processos de engenharia para que seja possível garantir em fase de desenvolvimento o funcionamento correto de cada tipo de sistema (BELLMAN *et al.*, 2020; MUCCINI; SHARAF; WEYNS, 2016).

3.2 Modelos e arquiteturas de referência para Self-CPS

O objetivo desta seção é apresentar um panorama geral da literatura sobre modelos/arquiteturas de referência para Self-CPS. Para isso, foi conduzido um mapeamento sistemático da literatura em setembro de 2021. Em síntese, essa técnica de revisão da literatura permite estabelecer uma visão justa e completa sobre um determinado tema de pesquisa, minimizando possíveis influências que um pesquisador pode cometer de maneira involuntária. Para conduzir esse mapeamento foi estabelecido um protocolo de pesquisa (veja Apêndice A), o qual é composto pelos seguintes elementos: *string* de busca utilizada para obtenção dos estudos, conjunto de bases de publicações selecionada, a definição das Questões de Pesquisa (QP) e os critérios de seleção dos estudos. Na Figura 7 são apresentadas as principais fases do mapeamento, a saber: (i) identificação da pesquisa, (ii) obtenção dos estudos, e (iii) aplicação de filtros. A seguir, uma breve descrição de cada fase é apresentada:

Figura 7 – Fases do mapeamento sistemático



Fonte: Elaborado pelo autor.

Identificação da Pesquisa. Fase caracterizada pelo estabelecimento do protocolo de pesquisa do mapeamento, sendo definido o tema de pesquisa, os termos comuns sobre tal tema

e a *string* de busca. Como objetivo deste trabalho é o estabelecimento de um modelo e/ou arquitetura de referência para apoiar o desenvolvimento de Self-CPS voltado para o domínio de saúde, a *string* de busca elaborada neste mapeamento é formada por três termos chaves, a saber: *reference architecture*, *cyber-physical systems*, *self-adaptive systems*. Os sinônimos de cada termo foram concatenados pelo operador `OR` e os termos resultantes pelo operador `AND`. Na Tabela 4 é apresentada a *string* de busca utilizada neste mapeamento.

Tabela 4 – *String* de pesquisa

`(reference architecture OR reference architectures OR reference model OR reference models)`
`AND`
`(CPS OR cyber physical OR cyber-physical OR cyberphysical OR embedded )`
`AND`
`(autonomic OR autonomous OR dynamic OR self-adaptable OR self-adaptation OR self-adapting OR self-adaptive)`

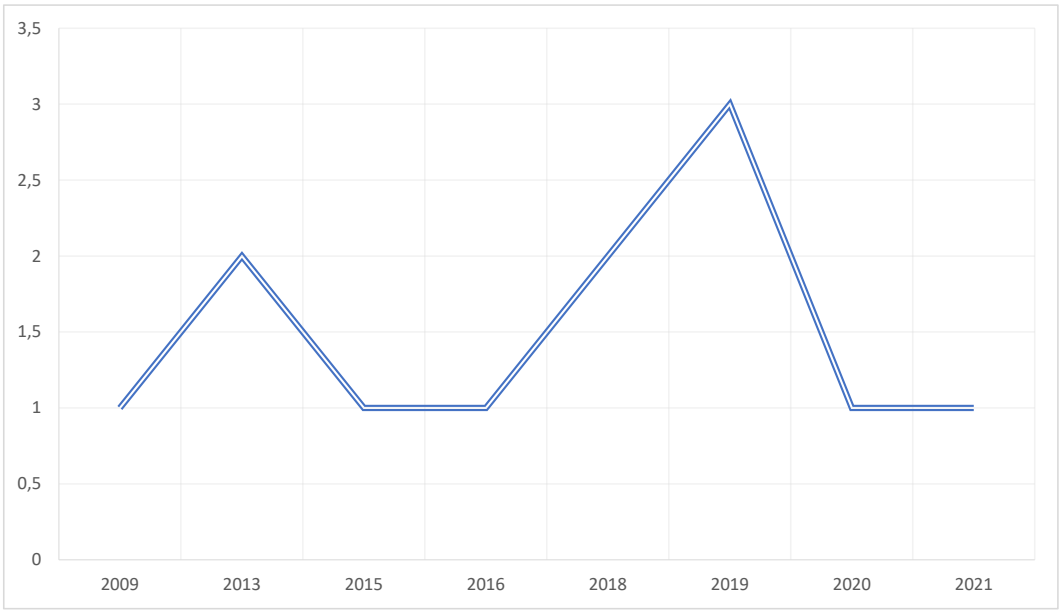
Obtenção dos Estudos. Fase caracterizada pela seleção de fontes e busca de estudos. De acordo com Dyba, Dingsoyr e Hanssen (2007), Kitchenham e Charters (2007), Petersen, Vakkalanka e Kuzniarz (2015), ACM, IEEE Xplore, Science Direct, Scopus, SpringerLink e Web of Science(vi) WoS, são as principais bases de publicação utilizadas na área de Engenharia de Software. A *string* de busca (veja Tabela 4) foi adaptada para atender as restrições de cada base, tais como: limitação quanto ao número de operadores lógicos e combinação de *strings* para que a busca seja realizada pelo título, *abstract* e palavra-chave.

Aplicação de Filtros. Após a obtenção dos estudos, oito filtros, seguindo os critérios definidos no protocolo de pesquisa (veja Apêndice A), foram aplicados. Para obter o resultado de 12 estudos, os seguintes filtros foram realizados: (i) Filtro 1, caracterizado pela inserção da *tag* `source` nos arquivos `.bibs` (251 estudos); (ii) Filtro 2, aplicada a remoção dos estudos que não são primários (231 estudos); (iii) Filtro 3, unificação dos estudos em um único arquivo `.bib`; (iv) Filtro 4, caracterizado pela remoção dos estudos repetidos (125 estudos); (v) Filtro 5, aplicação dos critérios de seleção (critérios de inclusão e exclusão) a partir do título, *abstract* e palavras chaves dos estudos (32 estudos); (vi) Filtro 6, leitura da introdução e conclusão dos estudos (32 estudos); (vii) Filtro 7, caracterizado pela exclusão de dois estudos indisponíveis para *download* e revisão de critérios de alguns artigos em conflito de critério (27 estudos); (viii) Filtro 8, leitura completa dos artigos que resultou em 12 estudos.

Antes de apresentar os dados resultantes em cada Questão de Pesquisa (QP), as seguintes distribuições são reportadas: ano de publicação, base de publicação e tipo de publicação. Em

relação aos anos de publicação, os estudos foram selecionados para este mapeamento estão entre **01 de janeiro de 2009 a 17 de setembro de 2021**, como ilustra a Figura 8. Nesse sentido, pode-se observar que o ano de 2019 apresentou a maior quantidade de estudos (três). Em contrapartida, os anos de 2009 e de 2020 apresentam a menor quantidade de estudos (um para cada ano). Uma vez que não foi selecionado um período cheio para o ano de 2021 (aproximadamente 9 nove meses), acredita-se que possa atingir uma maior quantidade de estudos.

Figura 8 – Distribuição dos estudos por ano de publicação

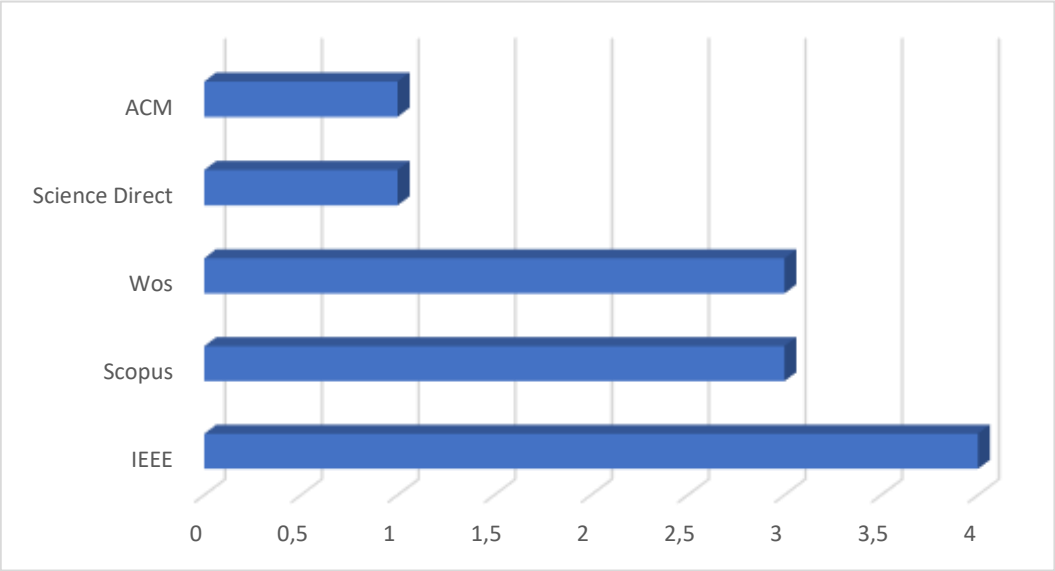


Fonte: Elaborado pelo autor.

Na Figura 9 é ilustrada a distribuição dos estudos selecionados para cada base de pesquisa. A partir dessa distribuição, observa-se que as bases IEEE, *Scopus* e *Web of Science* (WoS) possuem a maior quantidade de estudos, sendo quatro estudos indexados na IEEE (33.33%) e três estudos nas *Scopus* e WoS (25%), respectivamente. Uma justificativa para a quantidade retornada da base *Scopus* é o fato de que esta base indexa várias outras bases de publicação.

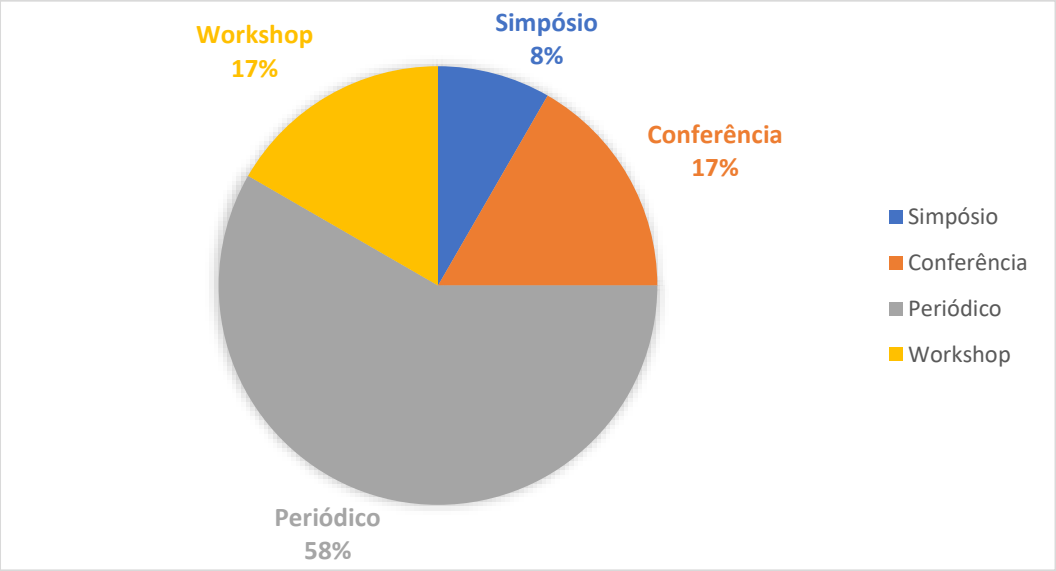
Por fim, outra perspectiva interessante a ser detalhada sobre os estudos é apresentar onde os mesmos foram publicados, conforme ilustra a Figura 10. Diante disso, pode-se verificar que a maioria dos estudos são publicados em periódicos (58%). Nesse sentido, pode-se dizer que o interesse da comunidade científica pelo tema em veículos de publicação considerados mais sólidos. Por outro lado, 17% dos estudos são publicados em conferência e *Workshops*, enquanto que somente 8% são publicados em simpósios. Em relação ao último veículo de publicação, pode ser inferido que há carências de trabalhos mais sólidos na área. A seguir, detalhes de cada questão de pesquisa são apresentados.

Figura 9 – Distribuição dos estudos por base de pesquisa



Fonte: Elaborado pelo autor.

Figura 10 – Distribuição dos estudos por tipo de publicação



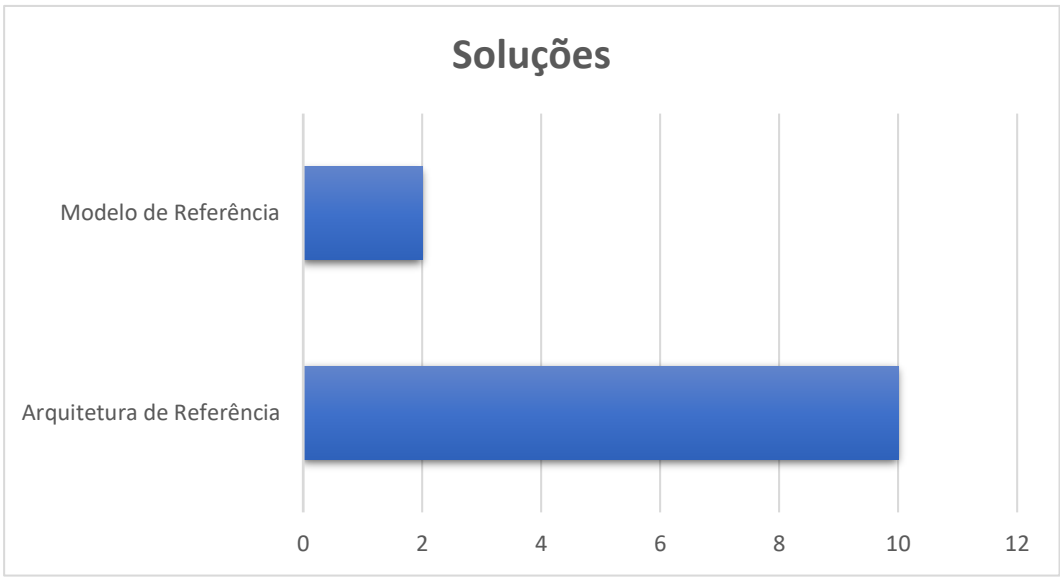
Fonte: Elaborado pelo autor.

QP 1: Quais são os modelos de referência e/ou arquiteturas de referência voltados ao desenvolvimento de Self-CPS?

O objetivo desta questão é identificar quais modelos/arquiteturas foram elaborados(as) para apoiar o desenvolvimento de Self-CPS. Nesse sentido, os estudos de nosso mapeamento foram categorizados em dois tipos, conforme reportado no Apêndice A. Conforme ilustra a Figura 11, os estudos que propuseram modelos de referência configuram 16,67%, enquanto que os estudos baseados em arquiteturas de referência representam 83,33% dos estudos selecionados.

A seguir, detalhes de cada categoria são apresentados.

Figura 11 – Distribuição dos estudos por tipo de solução



Fonte: Elaborado pelo autor.

Modelo de Referência. Categoria que engloba estudos que propuseram soluções que são caracterizadas como um modelo de referência. Em outras palavras, os estudos dessa categoria possuem escopos menores com soluções específicas para determinados cenários/problemas.

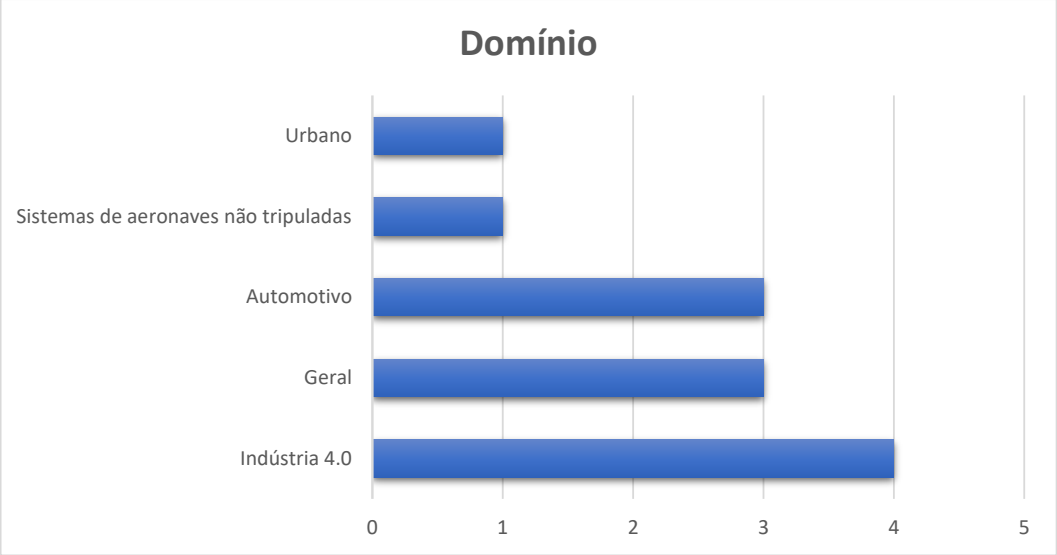
Arquitetura de Referência. Categoria que engloba estudos que propuseram soluções abrangentes. Em resumo, os estudos dessa categoria englobam o desenvolvimento de algoritmos e, principalmente, a decomposição/organização do sistema em camadas e/ou módulos evidenciando seus modos de funcionamento, que facilita o desenvolvimento desse tipo de sistema perante a comunidade acadêmica e profissionais do mercado.

A partir da Figura 11, percebe-se a diferença significativa entre os estudos (ou seja, modelos e arquiteturas de referência). A partir dessa distribuição, nota-se o interesse da comunidade científica em soluções que abordam escopos maiores, evidenciando que CPS são sistemas complexos e soluções dessa natureza (ou seja, arquiteturas de referência) são alternativas viáveis para apoiar o desenvolvimento. Além disso, pode-se destacar também o conhecimento multidisciplinar desse tipo de sistema (Self-CPS), uma vez que as soluções baseadas em arquiteturas tendem a cobrir um escopo maior de um sistema (Self-)CPS.

QP 2: Quais domínios de aplicação foram beneficiados por tais modelos/arquiteturas para Self-CPS?

Esta questão de pesquisa teve como finalidade identificar os principais domínios beneficiados pela utilização de Self-CPS. Vale destacar que os estudos deste mapeamento não reportaram de maneira clara e objetiva os nomes dos domínios, sendo necessário realizar uma inferência e agrupamento pelos tipos de aplicação que foram desenvolvidas em cada estudo. Dessa forma, foram identificados cinco domínios, a saber: (i) Urbano; (ii) Sistemas de aeronaves não tripuladas; (iii) Automotivo; (iv) Geral; e (v) Indústria 4.0. Conforme ilustrado na Figura 12, os principais domínios beneficiados são, respectivamente, Indústria 4.0 (representando 33% dos estudos selecionados), Geral e Automotivo (configurando 25% dos estudos cada). Em contrapartida, os domínios Sistemas de aeronaves não tripuladas e Urbano representam os domínios menos adotados (com 8,33% cada). A seguir, são detalhados os domínios identificados.

Figura 12 – Distribuição dos estudos por domínio



Fonte: Elaborado pelo autor.

Urbano. Caracterizado pela presença de diversos Self-CPS em variados níveis de infraestrutura das cidades. Em síntese, o domínio urbano é beneficiado por meio da inteligência coletiva destes sistemas, empregando técnicas de análise e busca de padrões nos dados advindos de diversas origens, aperfeiçoando os sistemas já presentes na cidade, como semáforos, controle de luzes, energia e água. Vale ressaltar que o papel do *smartphone* configura-se como primordial para esses sistemas, uma vez que diversos dados podem ser obtidos por meio desse dispositivo, caracterizando como a principal via de contato do sistema com seus usuários.

Sistemas de aeronaves não tripuladas. Domínio majoritariamente voltado para aplicações militares. Sua aplicação é focada em defesa e/ou ataque de um determinado território. Desse modo, os Self-CPS são integrados a este domínio com a finalidade de aperfeiçoar o sistema de navegação e raciocínio com foco na coordenação de voos autônomos. Nesse sentido, técnicas e algoritmos de inteligência artificial são empregues intensivamente nas diversas atividades destes sistemas.

Automotivo. Caracterizado pela inserção de diversos equipamentos eletrônicos nos veículos com o objetivo central de tornar a direção dos mesmos de forma automática. Para tal finalidade, mecanismos de comunicação com outros veículos (V2V, do inglês *vehicle-to-vehicle*) e com as infraestruturas da cidade e outros equipamentos (V2X, do inglês *vehicle-to-everything*) são essenciais, uma vez que configuram como o principal meio de percepção. Outras finalidades não centrais para este domínio incluem minimização de acidentes, alcançado por meio do acionamento automático dos pedais, e de congestionamento, realizado através de módulos que capturam dados de trânsito e calculam uma rota ótima para o destino desejado.

Geral. Foram incluídos estudos nesta categoria de domínio aqueles que implementam arquiteturas/modelos de referência que não são voltados para um domínio específico, podendo ser instanciado para qualquer domínio.

Indústria 4.0. Configura-se como o principal domínio dos Self-CPS, uma vez que a origem dos CPS é motivada em aplicações voltadas para a indústria. As aplicações deste domínio tem como finalidade otimizar o processo de fabricação das indústrias por meio da automatização, comunicação e inteligência das máquinas integradas, minimizando a participação humana no processo. Para isso, são utilizados conceitos de Internet das Coisas, Computação em Nuvem, Inteligência Artificial, *Digital Twin*, entre outros. Uma boa maneira de caracterizar esse domínio é apresentar os seis princípios básicos, a saber: capacidade de controle em tempo real; *Digital Twin*; descentralização; orientação a serviços; modularidade; e interoperabilidade.

Como ilustrado na Figura 12, observa-se que Indústria 4.0 e Automotivo representam os principais domínios para Self-CPS. O primeiro tem-se mostrado notório por impulsionar pela adoção dos CPS e pelo destaque na comunidade científica e nos países desenvolvidos que fazem uso desse tipo de sistema. O segundo domínio indica uma tendência da indústria automotiva de modernização dos veículos, representados pelo empenho na substituição dos combustíveis derivados do petróleo em energia elétrica e na automatização da atividade de direção por meio de mecanismos advindos da Inteligência Artificial. Por outro lado, a partir dos domínios menos adotados, é possível destacar os seguintes pontos, a saber: (i) o domínio de sistemas

de aeronaves não tripuladas representam um assunto crítico de interesse predominantemente de um país, não sendo interessante o compartilhamento de conceitos desse tipo de sistema; e (ii) apesar do domínio urbano indicar uma tendência a longo prazo, sua baixa adoção representa que o mesmo ainda não está em um estágio de maturidade desejado. Embora as evidências dessa questão revelarem a ausência do domínio de saúde (interesse de investigação de pesquisa - veja Apêndice A), a natureza crítica desses domínios podem colaborar com o projeto da arquitetura de referência para Self-CPS, pois são considerados domínios vizinhos ao que se pretende estabelecer neste projeto.

QP 3: Quais são os atributos de qualidade adotados no projeto de tais modelos e/ou arquiteturas

O objetivo desta questão de pesquisa é elencar os atributos empregues no desenvolvimentos dos modelos/arquiteturas. Ao final da etapa de extração, doze atributos foram selecionados, são eles: autonomia, confiabilidade, configurabilidade, consistência, integridade, interoperabilidade, performance, pontualidade, precisão, resiliência, robustez e segurança. Na Figura 13 são ilustrados os atributos de qualidade identificados nos estudos deste mapeamento.

Figura 13 – Nuvem de palavras sobre atributos de qualidade

Fonte: Elaborado pelo autor.

Os atributos ilustrados na Figura 13 podem atuar em diferentes contextos de um (Self-

)CPS. Por exemplo, na otimização dos algoritmos de adaptação (performance, precisão, pontualidade) com o mínimo de intervenção humana (autonomia). Esses sistemas necessitam de estabilidade (segurança, consistência, integridade) e devem ser resistentes à falhas (resiliência, robustez, configurabilidade) para que possam ser utilizados em diversos processos do software (interoperabilidade). Em questão de frequência, o atributo autonomia foi o mais utilizado representando, aproximadamente, 14,29%. Em seguida, os atributos de confiabilidade, interoperabilidade, performance, resiliência, robustez e segurança apresentam, aproximadamente, 9% cada. Por último, os atributos de configurabilidade, consistência, eficiência, integridade, pontualidade e precisão apresentam, aproximadamente, 4% cada.

QP 4: Como esses modelos e/ou arquiteturas têm sido avaliados(as)?

O objetivo desta questão de pesquisa foi identificar as técnicas de avaliação utilizadas nos estudos deste mapeamento. Consequência, essas técnicas podem revelar evidência sobre o nível de maturidade dos modelos/arquiteturas propostos nos estudos selecionados. Ao final da etapa de extração foram identificadas cinco abordagens de avaliação, a saber: protótipo, cenário de testes, exemplo instanciado, estudo de caso, e utilização de métricas. De acordo com a Figura 14, constata-se que a técnica que apresenta maior aplicação é a de estudo de caso com 17% dos estudos selecionados. Por outro lado, é possível observar que a maioria dos autores não propuseram nenhuma técnica avaliativa, representando 50% dos objetos de estudo. Dessa forma, constata-se que há uma dificuldade em estabelecer meios de avaliação, a qual pode ser justificada pelo fato que para realizar uma avaliação completa neste sistema é necessário ter um amplo conhecimento de várias áreas de pesquisa, pois os (Self-)CPSs interagem com diversos sistemas. A seguir, uma descrição de cada abordagem de avaliação é apresentada.

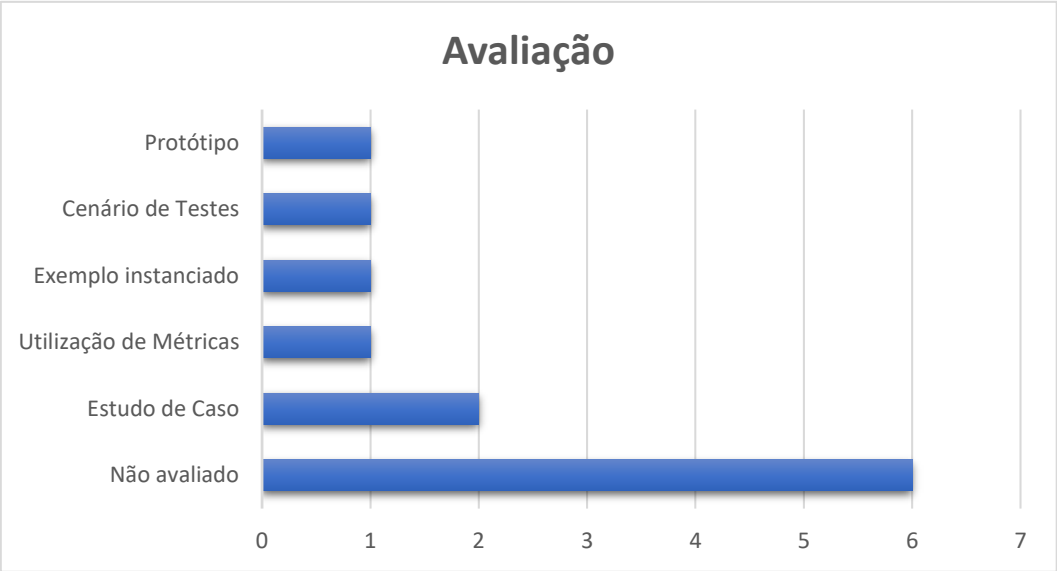
Protótipo. Pode ser caracterizado por uma abordagem menos madura acerca da implementação dos conceitos, tendo como foco a avaliação do sistema perante às novas ideias apresentadas.

Cenário de Testes. Abordagem que aborda um conjunto de testes a fim de verificar a aplicabilidade dos conceitos apresentados.

Exemplo instanciado. Pode ser caracterizado, no estudo selecionado, por colocar em práticas os conceitos apresentados em forma de um sistema de software, abordando as suas dificuldades geradas, mas sem a preocupação de realizar testes importantes em um dado cenário.

Estudo de caso. Este tipo de abordagem permite realizar uma investigação empírica que tem como base várias fontes de evidência para investigar um problema, sendo necessária uma

Figura 14 – Distribuição dos estudos por técnicas de avaliação



Fonte: Elaborado pelo autor.

descrição mais rigorosa sobre o comportamento do sistema tendo em vista as situações de um cenário proposto.

Utilização de métricas. Abordagem que apresenta como característica principal a utilização de métricas específicas para a solução desenvolvida. A avaliação é realizada comparando os valores das métricas aplicadas em relação aos valores obtidos das soluções tradicionais empregues no domínio de estudo.

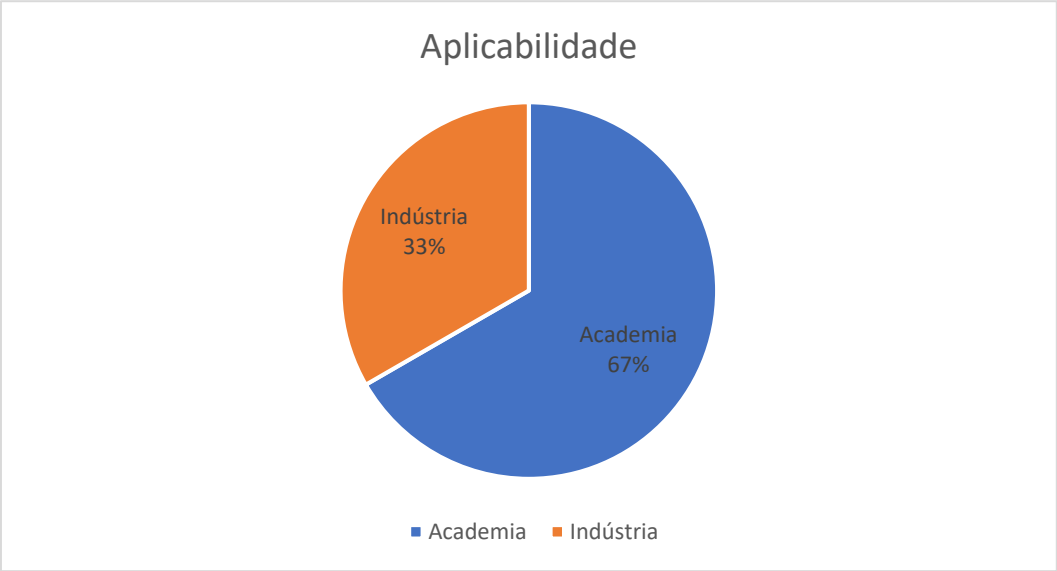
Não avaliado. Categoria que agrupa os estudos que não conduziram qualquer tipo de avaliação das soluções apresentadas.

QP 5: Quais são as evidências que motivam a adoção desses modelos e/ou arquiteturas pelos interessados?

O objetivo desta questão de pesquisa foi reunir evidências sobre o nível de adoção dos modelos/arquiteturas. Para esse fim foi analisada a maturidade de tais modelos/arquiteturas com base em alguns critérios, tais como: cooperação com empresas, completude do modelo/arquitetura, escopo do projeto, métodos de avaliação adotados, entre outros. Visando estabelecer um panorama das soluções adotadas, os estudos são classificados em duas categorias, a saber: Academia e Indústria. Assim, estudos classificados como “Academia” apresentam em sua natureza escopo menores ou ausência de avaliação do modelo/arquitetura proposto ou soluções teóricas. Por outro lado, estudos classificados como “Indústria” apresentam escopo maior e métodos de avaliação mais completos/sólidos. Conforme ilustra a Figura 15, as soluções classificadas como

"Academia" representam 67% dos estudos selecionados, sendo o restante (33%) para a "Indústria". Esses dados revelam que os estudos acadêmicos falham em apresentar uma solução com um escopo completo e com aplicabilidade dos métodos de avaliação, aumentando a resistência de cooperação entre a academia e a indústria.

Figura 15 – Distribuição dos estudos por evidências de adoção



Fonte: Elaborado pelo autor.

QP 6: Como esses modelos e/ou arquiteturas foram projetados(as)

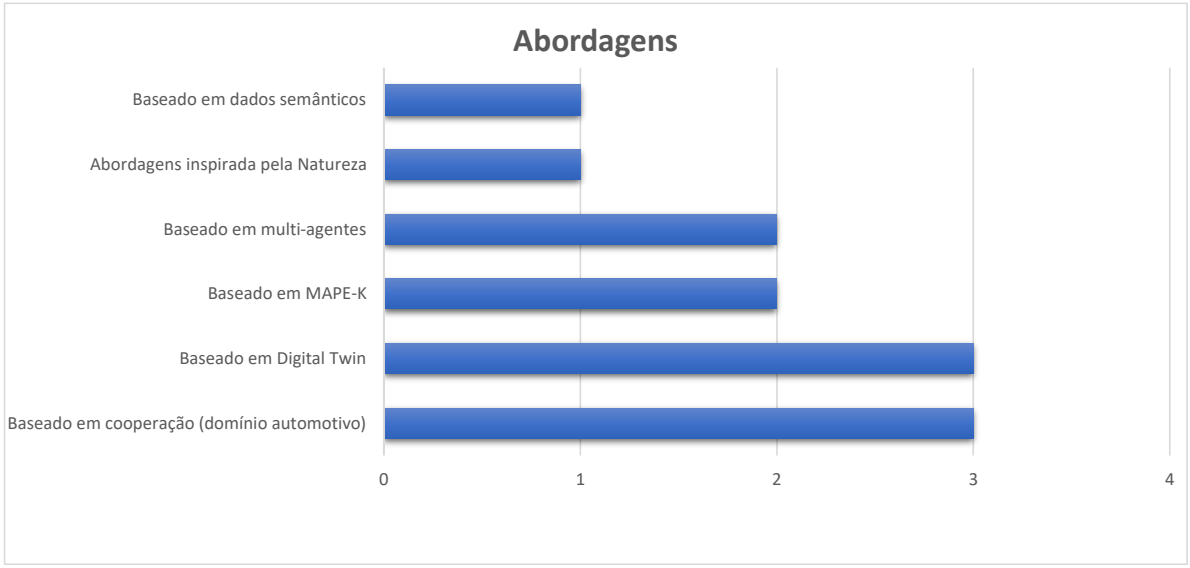
Esta questão de pesquisa tem como finalidade entender como tais modelos e/ou arquitetura têm sido projetados. Como esta questão está associada a vários outros aspectos, a mesma foi organizada em quatro subquestões. A seguir, uma descrição de cada subquestão é apresentada.

QP 6.1: Como esses modelos e/ou arquiteturas têm sido organizados?

O objetivo desta questão de pesquisa foi reunir evidências sobre a organização dos modelos/arquiteturas para Self-CPS. Em outras palavras, a finalidade desta questão foi identificar e entender como as principais abordagens de adaptação têm sido implementadas nos Self-CPS. Ao decorrer do processo de extração, verificou-se que os modelos/arquiteturas foram organizados em decorrência da abordagem de adaptação selecionada. Diante disso, ao final da etapa de extração, foram identificados seis tipos de abordagens de adaptação que podem fornecer evidências sobre a organização de tais modelos/arquiteturas, a saber: (i) baseada em dados semânticos; (ii) inspiradas pela natureza; (iii) baseada em multiagentes; (iv) baseada

no *loop* MAPE-K; (v) baseada em *Digital Twin*; e (vi) baseada em cooperação. A partir da Figura 16, percebe-se que as principais abordagens são baseadas em *Digital Twin* e baseadas em cooperação (representando 25% dos estudos). Por outro lado, as abordagens menos adotadas configuram 8,33% dos estudos (ou seja, dados semânticos e inspiradas pela natureza). A seguir, detalhes de cada abordagem são apresentados.

Figura 16 – Distribuição dos estudos por abordagens de adaptação



Fonte: Elaborado pelo autor.

Baseada em dados semânticos. Abordagem de adaptação apoiada por modelos semânticos que visam representar os principais elementos do software e do ambiente de aplicação, modo de operação dos equipamentos e seus relacionamentos. Desse modo, quando uma alteração é necessária, o modelo é utilizado para averiguar se a nova alteração respeita as regras de adaptação.

Abordagens inspiradas pela natureza. Ressaltando os conceitos apresentados na Seção 2.3, essa abordagem apresenta soluções cuja implementação é inspirada por características observáveis da natureza. Como exemplo pode-se citar o *Ant Colony Optimization*, algoritmo genético, *Artificial Bee Colony*, *Grey Wolf Optimizer*.

Baseado em multi-agentes. Conforme explorado na Seção 2.3, um sistema multi-agentes é composto por agentes, que são elementos computacionais responsáveis por perceber o ambiente e coletar informações por meio de sensores e decidir sobre o próximo estado do sistema.

Baseado em MAPE-K. Conforme descrito na Seção 2.3, o MAPE-K é um modelo voltado para o desenvolvimento de sistemas autônomos proposto pela IBM, sendo composto por 4 qua-

tro módulos, são eles: *Monitor* (Monitoramento), *Analyze* (Análise), *Plan* (Planejamento) e *Execute* (Execução).

Baseado em *Digital Twin*. Abordagem que possui forte vínculo com o domínio Indústria 4.0. Em resumo, *Digital Twin* é um modelo computacional desenvolvido com a finalidade de obter uma cópia digital do objeto/ambiente de interesse. Dessa forma, toda interação com o objeto/ambiente pode ser revista e estudada por este modelo antes de ser colocada em prática.

Baseado em cooperação. Abordagem que possui forte vínculo com o domínio Automotivo. Em síntese, a adaptação é alcançada por meio da comunicação e cooperação dos veículos, que ocorre através do compartilhamento de dados para que as ações de percepção e o raciocínio sobre os ambientes de operação sejam alavancados.

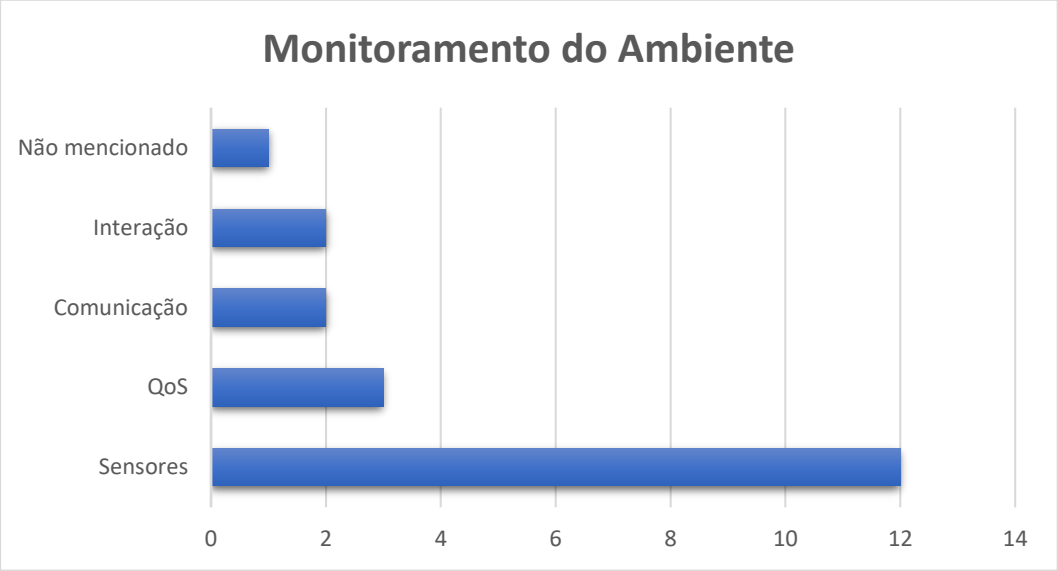
Portanto, a partir da Figura 16, é possível inferir que as abordagens baseadas em MAPE-K e multi-agentes, apesar de serem soluções tradicionais, ainda é vista pela comunidade científica como soluções interessantes para resolverem problemas de propósitos gerais. Em contrapartida, as abordagens baseadas em *Digital Twin* e em cooperação indicam uma tendência para resolver problemas relacionados aos domínios novos (por exemplo, Indústria 4.0 e Automotivo). Por fim, as abordagens restantes apresentam-se como soluções interessantes para resolverem problemas específicos, uma vez que soluções inspiradas pela natureza são ótimas para resolverem questões de otimização e soluções baseados em dados semânticos são interessantes para resolverem questões relacionadas à representação de relacionamentos de componentes do sistema e sua semântica.

QP 6.2: Como os modelos e/ou arquiteturas lidam como o monitoramento do ambiente de execução?

O objetivo desta questão de pesquisa é identificar as soluções de monitoramento do ambiente de execução reportadas nos estudos selecionados neste mapeamento. Em outras palavras, esta questão tem por finalidade expor os mecanismos que atuam como gatilho para o processo de adaptação. Ao final da extração, quatro categorias foram identificadas, a saber: (i) interação, (ii) comunicação, (iii) QoS e (iv) sensores. Vale destacar que as soluções não são exclusivas, ou seja, o uso de uma solução não invalida a aplicação de outra. Na Figura 17 é ilustrada a distribuição das categorias identificadas nesta questão de pesquisa. Em seguida, uma descrição de cada categoria é apresentada.

Não mencionado. Categoria que agrupa os estudos que não reportaram estratégias, métodos e/ou soluções de monitoramento para o ambiente de execução de uma aplicação Self-CPS.

Figura 17 – Distribuição categorias de monitoramento do ambiente



Fonte: Elaborado pelo autor.

Interação. Categoria que engloba um conjunto de estudos que exploram o domínio Urbano e Geral. Em síntese, essa categoria é caracterizada por uma constante interação com humanos para determinar a situação atual do sistema e, a partir dos coletados por esses sistemas, determinar se uma adaptação é necessária.

Comunicação. Essa categoria é voltada para os estudos que abordam o domínio Automotivo. Basicamente, os estudos exploram a comunicação entre veículos com a finalidade de obter uma visão ampla do ambiente de execução e, consequentemente, para obter dados mais precisos de um contexto específico.

QoS. Categoria caracterizada pelo uso de métricas, cuja finalidade é estimar a eficiência e qualidade de um componente do sistema (Self-CPS), serviço ou parte dele. Caso as métricas do objeto em análise apresente um limite abaixo do estipulado na fase de projeto, um processo de adaptação pode ser desencadeado para correção do problema e preservação da qualidade de comunicação.

Sensores. Essa categoria engloba todos os estudos selecionados neste mapeamento. A principal característica dessa categoria é a aplicação de hardwares especializados para capturar e enviar dados de diversas origens do ambiente da aplicação para componentes específicos que realizam análises e processamento dos mesmos para tomadas de decisão quanto às possibilidades de funcionamento e modificação do software (Self-CPS).

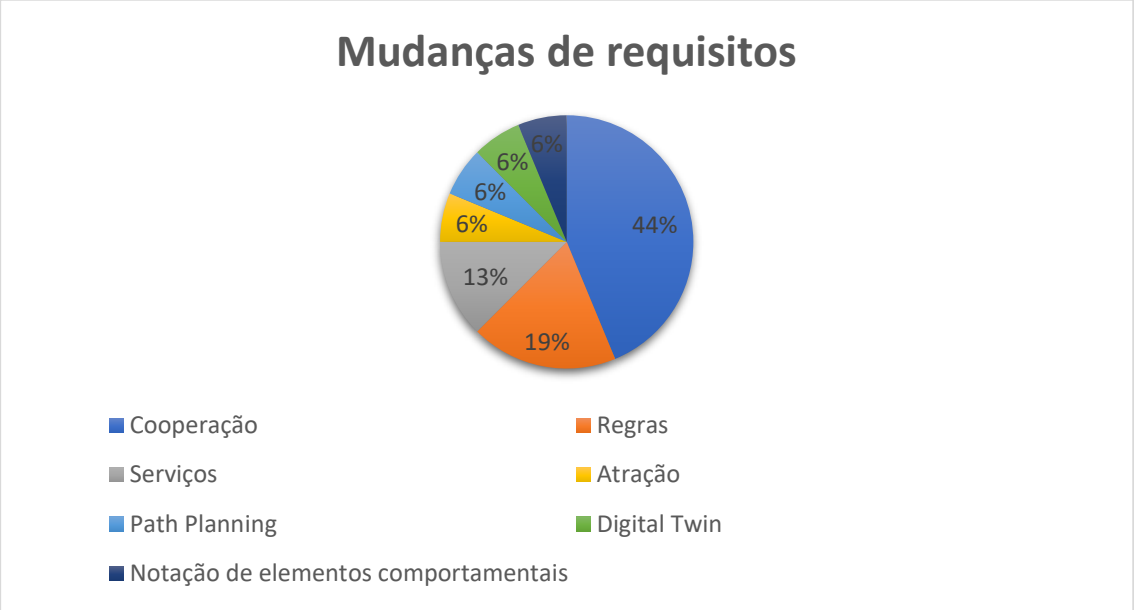
A partir da distribuição ilustrada na Figura 17, pode-se observar que devido a natureza intrínseca dos CPS, todos os estudos reportaram ter utilizado hardwares especializados como

forma de monitoramento do ambiente. Em outras palavras, os sensores são utilizados para coletar os dados do ambiente e transferidos para um sistema interno de um Self-CPS. Em outra perspectiva, nota-se uma carência de meios alternativos para realizar o monitoramento do ambiente, limitando-se principalmente a análise de medidas como QoS, Comunicação e Interação. Tal cenário pode indicar que a comunidade científica analisa a implementação direta (sensores) como uma alternativa para ciência do ambiente como satisfatória, sendo as outras categorias identificadas neste mapeamento soluções adicionais para resolverem problemas específicos.

QP 6.3: Como a mudança de requisitos e/ou de contexto em tempo de execução tem sido tratada em tais modelos/arquiteturas?

O objetivo desta questão de pesquisa é elencar as abordagens responsáveis por modificar o comportamento do software em tempo de execução. Ao final da etapa de extração, oito abordagens foram identificadas, são elas: (i) Notação de elementos comportamentais, (ii) *Path Planning*, (iii) Atração, (iv) Regras, (v) Orientado a Objetivos, (vi) Serviços e (viii) Cooperação. Na Figura 18 é ilustrada a distribuição das abordagem identificadas nos estudos deste mapeamento.

Figura 18 – Distribuição dos estudos em respostas a mudanças de requisitos



Fonte: Elaborado pelo autor.

Notação de elementos comportamentais. Categoria que agrupa estudos que apoiam em modelos descritivos para analisar as regras e a dinâmica dos componentes de software e ambiente de operação, propondo uma solução para o problema satisfazendo as determinações que o domínio de aplicação exige.

***Path Planning*.** Abordagem específica para o domínio Automotivo. Em suma, é caracterizada pela aplicação de algoritmos que fornecem a melhor ou uma ótima rota do ponto A para o ponto B, analisando diversos fatores, como trânsito, tempo de viagem, semáforos, dentre outros.

Atração. Implementação que emprega o conceito de atração em componentes de *software*, que por sua vez, ditam a maneira que os mesmos irão se interagir, modificando assim seu comportamento no ambiente de aplicação. Dessa forma, ao alterar certos parâmetros relacionados à atração dos componentes, os comportamentos dos componentes são modificados.

Regras. Categoria que possui relação com o paradigma de sistemas multi-agentes, uma vez que o mesmo é orientado a regras. Nesse sentido, caso o conjunto de regras atuais não atendam o domínio em questão, o sistema reavalia suas regras atuais, de tal forma a modificá-lo para suprir as necessidades do ambiente de aplicação.

***Digital Twin*.** Abordagem específica para o domínio Industrial. Nesse sentido, ao detectar uma adaptação necessária, o sistema avalia a efetividade da modificação realizada no *Digital Twin* para se decidir qual alteração será implementada no ambiente de execução.

Serviços. Abordagem caracterizada por modificar o comportamento do software através da composição dos serviços e/ou alteração de parâmetros.

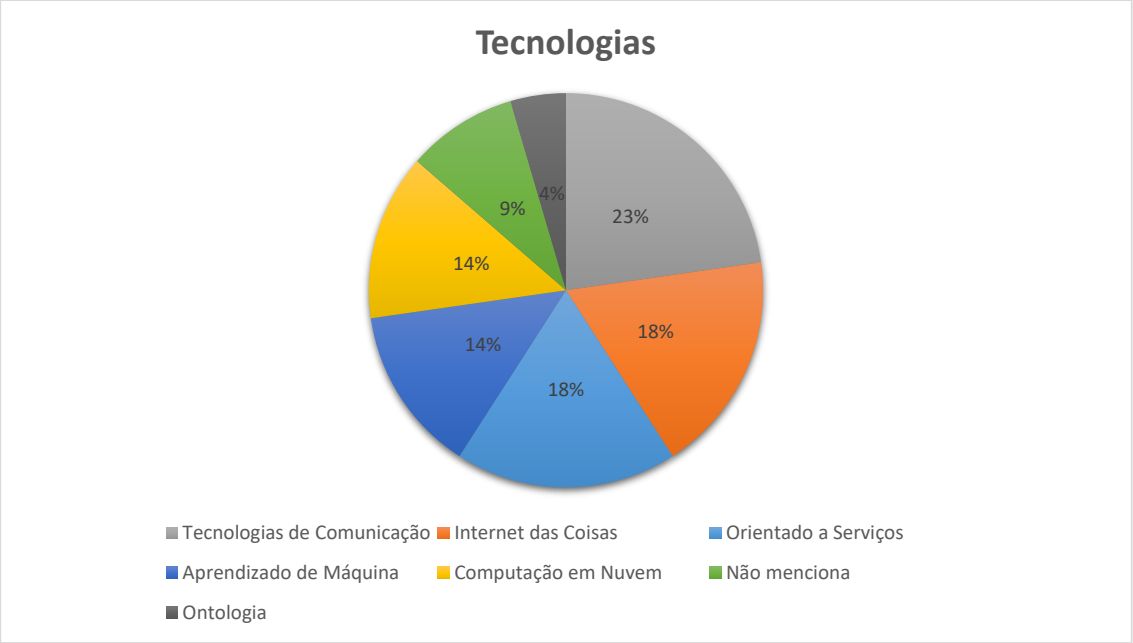
Cooperação. Principal abordagem aplicada em diferentes domínios (Automotivo, Geral, Indústria 4.0 e Sistemas de aeronaves não tripuladas). De forma sucinta, nos estudos selecionados, a cooperação é entendida por qualquer interação realizada entre componentes de software, de modo a trocar informações para estar solucionando um dado problema em conjunto.

A partir da Figura 18, verifica-se uma diversidade nas abordagens de mudanças de requisitos implementadas. Como consequência, a distribuição ilustrada nessa figura reforça a questão multidisciplinar dos CPS, uma vez que uma solução pode apresentar resultados satisfatórios em um conjunto de domínios e insatisfatórios em outros. Além disso, constata-se que a abordagem baseada em cooperação é a solução que possui maior adoção nos estudos selecionados. Dessa forma, essa situação acentua a questão de complexidade das incertezas encontradas nestes sistemas, dado que a adaptação é estendida a diversos componentes de software.

QP 6.4: Quais recursos têm sido empregados em tais modelos e/ou arquiteturas?

Esta questão de pesquisa tem por finalidade reportar os principais recursos que foram aplicados nos modelos/arquiteturas selecionados neste mapeamento. Em síntese, esta questão agrupa as tecnologias utilizadas em áreas do conhecimento, modelos de dados e abordagens de comunicação. Ao final da etapa de extração, seis categorias foram identificadas, a saber: (i) Aprendizado de Máquina, (ii) Computação em Nuvem, (iii) Internet das Coisas, (iv) Orientado a Serviços, (v) Tecnologias de Comunicação, e (vi) Ontologia. Na Figura 19 é ilustrada a distribuição dos estudos em relação às tecnologias utilizadas. A seguir, uma breve descrição de cada categoria é apresentada.

Figura 19 – Distribuição dos estudos por tecnologias utilizadas



Fonte: Elaborado pelo autor.

Aprendizado de Máquina. Representa um conjunto de estudos que implementam algoritmos de aprendizado de máquina em seus modelos/arquiteturas. Conforme explorado na Seção 2.4, aprendizado de máquina é um ramo da Inteligência Artificial que estuda técnicas voltadas para fazer com que computadores e softwares possam aprender e se autoprogramarem por meio de dados coletados.

Computação em Nuvem. Agrupa estudos que possuem a característica de permitir o acesso a diversos recursos computacionais, como aplicações, servidores, armazenamento, entre outros, por meio de tecnologias de comunicação de rede.

Internet das Coisas. Representa um conjunto de estudos que implementam conceitos relacionados à área de Internet das Coisas. Conforme mencionado na Seção 2.4, pode ser caracterizado como um meio de comunicação entre sensores, atuadores e elementos inteligentes com o objetivo de prover a troca de dados entre dispositivos.

Orientado a Serviços. Categoria que agrupa estudos que apresentam arquitetura/modelos caracterizados por implementarem conceitos relacionados a SOA (do inglês, *Service-Oriented Architecture*).

Tecnologias de Comunicação. Engloba um conjunto de estudos que pertencem ao domínio Automotivo. Nesse contexto, é imprescindível tecnologias de comunicação para que esses sistemas possam obter uma completa ciência do ambiente de execução. Dentro desse domínio, essas tecnologias são essenciais por viabilizarem a comunicação entre veículos e a infraestrutura das cidades.

Não menciona. Reúne estudos que não propuseram e/ou fizeram uso de alguma tecnologia, sendo classificados como estudos teóricos que não instanciaram os modelos e/ou arquiteturas propostos.

Ontologia. Agrupa estudos que aplicaram um modelo de dados cuja sua principal finalidade é capturar uma representação semântica do ambiente de aplicação e/ou dos recursos que compõem o sistema.

Segundo os dados apresentados na Figura 19, verifica-se que os Self-CPS empregam diversos recursos tecnológicos originados de várias áreas do conhecimento, com destaque nas categorias Tecnologias de Comunicação, Internet das Coisas e Orientado a Serviços. Sumariamente, esses dados realçam o potencial de aplicação dos Self-CPS em diferentes domínios, uma vez que é identificado uma diversidade e equilíbrio em relação ao emprego dos recursos tecnológicos. Vale destacar ainda que os estudos que não aplicaram nenhum recurso tecnológico apresentam cunho teórico, uma vez que a instância de tais arquiteturas/modelos em soluções concretas não foi o objetivo desses estudos.

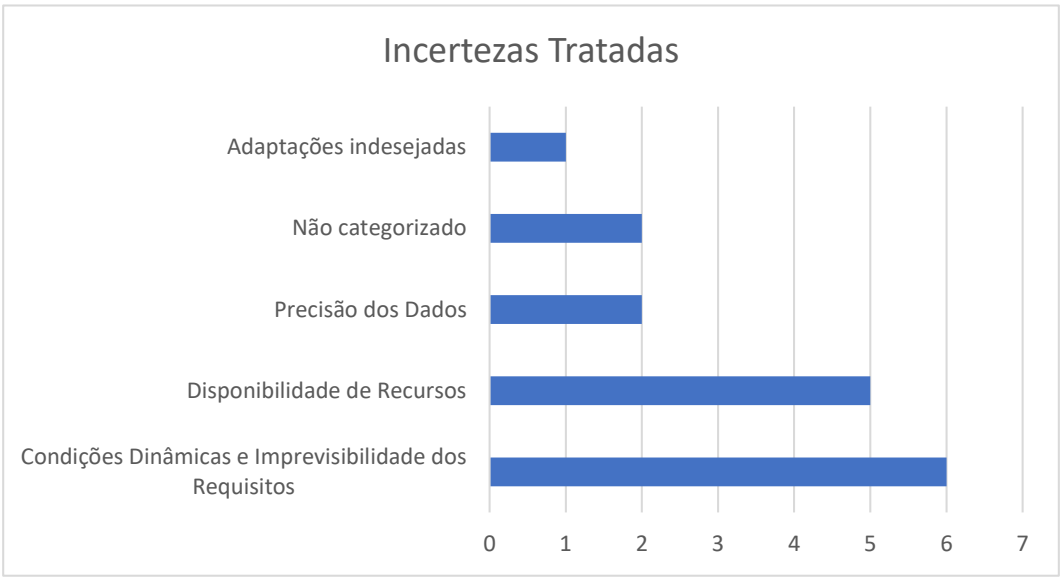
QP 7: Como os interesses de adaptação em tempo de execução têm sido abordados em tais modelos e/ou arquiteturas?

Esta questão tem como finalidade compreender como tais modelos e/ou arquitetura foram projetados em relação aos interesses de adaptação, ou seja, como a lógica de adaptação foi organizada nesse tipo de aplicação (Self-CPS). Como esta questão está associada a vários outros aspectos, a mesma foi organizada em duas subquestões.

QP 7.1: Quais tipos de incertezas têm sido tratadas nesses modelos e/ou arquiteturas?

O objetivo desta questão de pesquisa foi identificar os tipos de incertezas que têm sido tratadas nos estudos selecionados neste mapeamento. De modo geral, esta questão teve por finalidade identificar as motivações para implementar técnicas de adaptação ao sistema (Self-CPS). Ao final da extração, foram identificados quatro categorias, a saber: (i) Condições Dinâmicas e Imprevisibilidade dos Requisitos, (ii) Adaptações Indesejadas, (iii) Precisão dos Dados, e (iv) Disponibilidade dos Recursos. A distribuição dos estudos em relação às tecnologias utilizadas é ilustrada na Figura 20. A seguir, uma breve descrição de cada categoria é apresentada.

Figura 20 – Distribuição dos estudos por tipo de incerteza



Fonte: Elaborado pelo autor.

Adaptações Indesejadas. Estudos classificados nesta categoria indicam que a adaptação do software em tempo de execução ocorre de maneira controlada. Em outras palavras, deve-se delimitar os escopo de adaptação para que adaptações indesejadas não ocorram, fazendo com que o software não seja inviabilizado.

Não categorizado. Categoria que descreve os estudos que não explicitaram as incertezas tratadas em seus modelos/arquiteturas.

Precisão dos Dados. Engloba um conjunto de estudos que necessitam de alta precisão da informação. Por se tratar de um domínio crítico onde o conhecimento do ambiente é alcançado por meio de dados obtidos por outras fontes, mecanismos de análise de dados são imprescindíveis para posterior análise.

Disponibilidade dos Recursos. Categoria que é caracterizada por tratar incertezas pertinentes a disponibilidade de recursos referente a natureza de software e hardware. Em relação ao software, podem ser destacados módulos/partes do sistema que tornam indisponíveis por questões de latência na comunicação e/ou fila de processamento cheia. No que diz respeito ao hardware, questões relacionadas à falha dos componentes de hardware podem ser destacadas, pois causam funcionamento inadequado no sistema.

Condições Dinâmicas e Imprevisibilidade dos Requisitos. Categoria que engloba condições adversas que não são previstas em fase de *design*. Dessa forma, a abordagem de adaptação é implementada de forma a obter uma ciência geral do ambiente de aplicação para que seja possível realizar adaptações de diversos tipos de natureza.

A partir da Figura 20, nota-se que a categoria "Condições Dinâmicas e Imprevisibilidade dos Requisitos", que trata incertezas de âmbito geral, é questão de maior dificuldade dentro do contexto de desenvolvimento para Self-CPS. Em paralelo, percebe-se uma preocupação quanto ao funcionamento do sistema em nível de hardware e software em meio a diversas situações adversas que esses sistemas operam. Por outro lado, categorias como "Precisão de Dados" e "Adaptações Indesejadas" representam questões relacionadas a propósitos específicos, e por isso, apresentam baixa adesão nos estudos selecionados neste mapeamento.

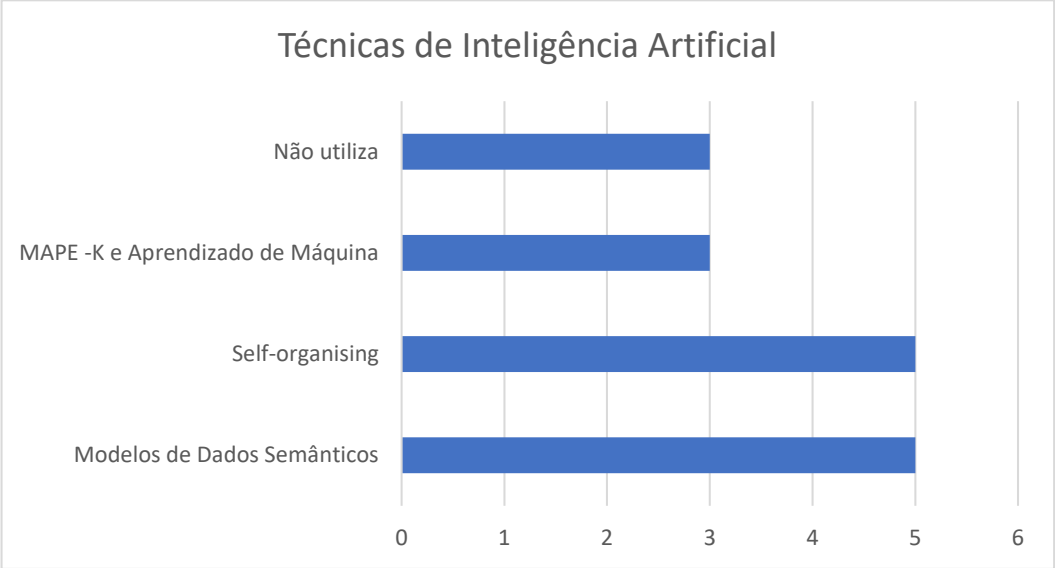
QP 7.2: Quais técnicas de inteligência artificial têm sido empregadas no projeto desses modelos e/ou arquiteturas para lidar com os interesses de auto-adaptação?

Esta questão de pesquisa tem por objetivo capturar as técnicas de inteligência artificial utilizadas nos estudos selecionados neste mapeamento. Em resumo, esta questão analisa abordagens que auxiliam no processo de adaptação do sistema. Ao final da etapa extração, três categorias foram identificadas, a saber: (i) MAPE-K e Aprendizado de Máquina, (ii) *Self-organising*, e (iii) Modelos de Dados Semânticos. Na Figura 21 é ilustrada a distribuição dos estudos em relação às técnicas de inteligência artificial utilizadas. A seguir, uma descrição de cada categoria é apresentada.

MAPE-K e Aprendizado de Máquina Agrupa estudos que implementaram um modelo voltado para o desenvolvimento de sistemas autônomos proposto pela IBM ou que utilizaram algoritmos de aprendizado de máquina para gerenciamento do conhecimento.

Não utiliza. Categoria que agrupa os estudos que não aplicaram nenhuma abordagem relacionada a área da Inteligência Artificial.

Figura 21 – Distribuição dos estudos por técnicas de inteligência artificial



Fonte: Elaborado pelo autor.

***Self-organising*.** Agrupam estudos que implementaram em seus modelos/arquiteturas uma propriedade denominada *self-organising*. Essa propriedade representa um processo adaptativo e dinâmico fazendo com que o software mantenha sua estrutura sem o intermédio de um controle externo. Em outras palavras, é um comportamento que adquire informações sobre si mesmo e mantém sua normalidade.

Modelos de Dados Semânticos. Agrupa estudos que aplicaram um modelo de dados cuja finalidade é capturar uma representação semântica do ambiente de aplicação e/ou dos recursos que fazem parte do sistema (Self-CPS).

Conforme apresentado na Figura 21, nota-se uma tendência em utilizar a propriedade *self-organising* e modelos de dados semânticos. Como os CPS são sistemas complexos e operam em ambientes imprevisíveis, a aplicação dessas técnicas facilitam o desenvolvimento desses sistemas nestes ambientes, uma vez que a adaptação é realizada sem o intermédio de um controle externo. Além disso, percebe-se um número expressivo (em relação ao total) de estudos que aplicaram abordagens baseadas em MAPE-K e/ou Aprendizado de Máquina. Em síntese, essas técnicas permitem que o sistema possa estar ciente das mudanças de requisitos realizadas, fazendo com que a adaptação seja aplicada em tempo hábil. Uma razão para que a abordagem baseada *self-organising* foi mais aplicada pode ter como base o fato de que a abordagem baseada em MAPE-K e em Aprendizado de Máquina necessite de mais conhecimento dos desenvolvedores acerca do ambiente do aplicação, uma vez que uma base de conhecimento é estritamente necessária para que o funcionamento seja satisfatório. Por último, nota-se que três estudos não propuseram nenhuma implementação de técnicas de inteligência artificial.

3.3 Considerações finais

Neste capítulo foi apresentado um panorama sobre modelos/arquiteturas de referência para Self-CPS. Para isso, uma pesquisa sistemática foi conduzida com o intuito de levantar as principais soluções e interesses da comunidade científica desses sistemas ao que concerne aos modelos e arquiteturas de referência. Estudos foram extraídos da comunidade científica e analisados, e ao final deste processo, 12 estudos (ver Apêndice B) foram selecionados para uma análise aprofundada utilizando questões de pesquisa como parâmetros desta análise. Em resumo, verifica-se que os Self-CPS são utilizados em diversos domínios de aplicação, tendo apoio de desenvolvimento por abordagens de adaptação baseadas em cooperação, *digital twin* e MAPE-K com foco para antecipar as condições dinâmicas do ambiente e situações imprevistas e maximizar a disponibilidade. Além disso, os Self-CPS são beneficiados por diversas tecnologias, como Internet das Coisas, Aprendizado de Máquina, Computação em Nuvem, entre outros. A partir dos resultados extraídos, referências e soluções foram capturadas para o desenvolvimento da arquitetura de referência.

4 Arquitetura de referência para Self-CPS

O objetivo deste capítulo é apresentar as etapas que guiaram o desenvolvimento da arquitetura de referência RA4Self-CPS (do inglês, *Reference Architecture for Self-CPS*). O projeto dessa arquitetura reúne as principais contribuições do mapeamento da literatura reportado no Capítulo 3 e experiências anteriores de nosso grupo de pesquisa (AFFONSO; NAKAGAWA, 2013; AFFONSO; PASSINI; NAKAGAWA, 2019; PASSINI, 2020). No que diz respeito ao projeto da RA4Self-CPS, optou-se por utilizar as diretrizes do PROSA-RA combinadas com a abordagem proposta por Tummers *et al.* (2021), que visa apoiar a construção de arquiteturas de referência para sistemas de informação voltados para o domínio de saúde. Em resumo, o projeto da RA4Self-CPS pode ser sintetizado em quatro etapas, a saber: (i) investigação das fontes de informação, (ii) identificação dos requisitos arquiteturais, (iii) projeto arquitetural, e (iv) avaliação da arquitetura de referência. A seguir, Seções 4.1 a 4.4, uma descrição de cada etapa é apresentada.

4.1 Passo AR-1: Investigação das fontes de informação

Nesta etapa foram elencadas as principais fontes de informação para identificação dos requisitos relacionados ao projeto da RA4Self-CPS. Em resumo, essas fontes de informação foram agrupadas em dois conjuntos, a saber: (i) diretrizes para o desenvolvimento de Self-CPS, e (ii) arquiteturas/modelos de referências para Self-CPS. O primeiro conjunto teve por objetivo reunir os principais conceitos e fornecer evidências sobre as boas práticas utilizadas no desenvolvimento de Self-CPS. Já o segundo conjunto teve como meta sintetizar as principais evidências sobre o projeto de arquiteturas e/ou modelos de referência para Self-CPS, onde foi possível identificar as funcionalidades obrigatórias, as semelhanças entre os modelos e/ou arquiteturas. A seguir, detalhes de cada conjunto são apresentados.

Conjunto 1 – Diretrizes para desenvolvimento de Self-CPS. Neste conjunto foram agrupados conceitos e informações que atuam como suporte para o desenvolvimento de sistemas autoadaptativos de âmbito geral e sistemas ciber-físicos, gerando os chamados Self-CPS (veja Seção 2.4). Conforme exposto na Seção 2.3, SaS é um tipo especial de sistema de software dotado de funcionalidades específicas, uma vez que as modificações podem ser realizadas em diferentes âmbitos: estrutural, comportamental, contextual e/ou combinação de ambos. Assim, visando elencar os requisitos para o projeto da RA4Self-CPS quanto ao âmbito da adaptação, optou-se por utilizar o modelo 5W1H. Conforme já mencionado na Seção 2.3, esse modelo é formado por seis questões, que facilita a identificação dos requisitos, as quais devem ser

respondidas durante a fase de desenvolvimento e fase operacional.

No que diz respeito ao estado da arte sobre os Self-CPS, foi conduzido um mapeamento da literatura (veja Capítulo 3). Com base nos estudos selecionados nesse mapeamento, informações importantes foram extraídas, a saber: (i) os principais domínios que envolvem (Self-)CPS, (ii) as principais abordagens adotadas para viabilizar a autoadaptação, (iii) os principais padrões de projeto utilizados em (Self-)CPS, (iv) os principais interesses tratados pela adaptação, e (v) as principais dificuldades encontradas ao implementar propriedades de adaptação.

Conjunto 2 – Arquiteturas de referência para Self-CPS. Neste conjunto foram agrupadas e estudadas as principais arquiteturas de referência para Self-CPS. Para isso, foi conduzido o mapeamento sistemático da literatura (veja Capítulo 3), que teve como resultado 12 estudos primários distribuídos entre 01 de janeiro de 2009 a 17 de setembro de 2021. Os resultados desse mapeamento permitiram estabelecer uma visão geral e um entendimento mais sólido sobre como os modelos e/ou arquiteturas de referência foram projetados nos últimos anos. Dessa forma, é possível evidenciar importantes indicadores, como o estágio atual dos Self-CPS, as lacunas de pesquisa e perspectivas para trabalhos futuros. Portanto, pode-se dizer que esse panorama/entendimento foi importante para estabelecer os requisitos arquiteturais da RA4Self-CPS.

4.2 Passo AR-2: Estabelecimento dos requisitos arquiteturais

Com base nas informações identificadas no passo anterior, foram desenvolvidos requisitos da arquitetura de referência proposta. Nesse passo, os requisitos foram organizados em duas categorias, a saber: (i) Requisitos de Adaptação, e (ii) Requisitos Ciber-Físico, os quais são relacionados aos propósitos gerais do sistema (ou seja, (Self-)CPS). A seguir, é elencado os requisitos juntamente com sua descrição.

- **[RA - 1]** A arquitetura de referência deve permitir a substituição de uma entidade de software em tempo de execução.
- **[RA - 2]** A arquitetura de referência deve permitir que uma situação adversa seja resolvida em tempo de execução.
- **[RA - 3]** A arquitetura de referência deve prover um mecanismo de rastreabilidade capaz de reportar um *log* a respeito das entidades modificadas em tempo de execução.
- **[RA - 4]** A arquitetura de referência deve prover um mecanismo encarregado pela sincronização das entidades modificadas com o esquema de dados utilizado no banco.
- **[RA - 5]** A arquitetura de referência deve permitir realizar a troca de implementação de um serviço em tempo de execução.

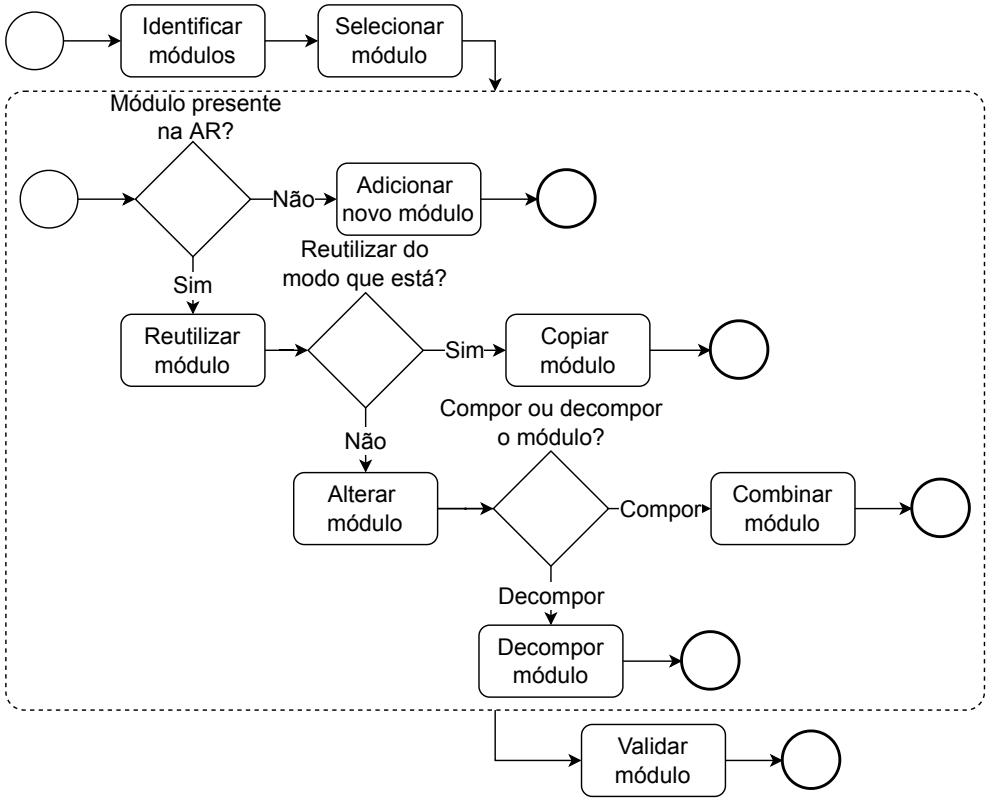
- **[RA - 6]** A arquitetura de referência deve fornecer um mecanismo capaz de identificar e reportar um problema quando uma operação de adaptação é efetuada.
- **[RA - 7]** A arquitetura de referência deve elaborar um plano para adaptar um Self-CPS diante de uma situação não prevista do ponto de vista estrutural, comportamental, contextual e/ou ambos.
- **[RA - 8]** A arquitetura de referência deve comunicar/alertar os interessados sobre uma situação não contornável. A partir desse cenário, entende-se que o sistema pode transitar para um cenário irreversível e necessita de intervenção humana para retornar ao estado normal de funcionamento.
- **[RA - 9]** A arquitetura de referência deve permitir o monitoramento do Self-CPS para avaliar suas condições de funcionamento.
- **[RCF - 1]** A arquitetura de referência deve fornecer um mecanismo capaz de identificar e notificar os interessados do sistema sobre as requisições anômalas realizadas na API (do inglês, *Application Programming Interface*).
- **[RCF - 2]** A arquitetura de referência deve fornecer um mecanismo capaz de oferecer ao usuário a opção de conceber um histórico do estado do ambiente de aplicação e dos componentes de software.
- **[RCF - 3]** A arquitetura de referência deve fornecer aos usuários um *dashboard* para que seja possível visualizar a situação atual do ambiente em uma janela de tempo.
- **[RCF - 4]** A arquitetura de referência deve permitir que um Self-CPS esteja ciente do contexto ao qual está inserido.
- **[RCF - 5]** A arquitetura de referência deve permitir o monitoramento do ambiente de aplicação.

Além de padrões e arquiteturas de referência, outras fontes de informação também foram analisadas para que a lista de requisitos fosse gerada: (i) desenvolvimento do SaS e seu motor de adaptação (SALEHIE; TAHVILDARI, 2009; CHENG *et al.*, 2009); (ii) desenvolvedores SaS com experiência em design e implementando SaS (WEYNS, 2017), e (iii) exemplos de sistemas (IGNATIUS; BAHSOON, 2021; LEITÃO; COLOMBO; KARNOUSKOS, 2016; GREER *et al.*, 2019). Em relação ao último item, vale ressaltar que os autores supracitados desenvolveram tais sistemas para diferentes domínios e forneceram excelentes contribuições (por exemplo, cenários de adaptação, requisitos de arquitetura, necessidades dos usuários finais, entre outros recursos) para o design de arquitetura de referência proposta nesta dissertação.

4.3 Passo AR-3: Projeto arquitetural

Conforme mencionado no início deste capítulo, o projeto da RA4Self-CPS teve como base a combinação entre as diretrizes do processo PROSA-RA e a abordagem proposta por Tummers *et al.* (2021). Essa abordagem fornece um processo para construção e/ou reutilização de módulos em arquiteturas de referência para sistemas de informação voltados para o domínio de saúde. Portanto, as fontes de informação para o projeto da RA4Self-CPS são os conjuntos de informações 1 e 2 (veja Seção 4.1) e os requisitos arquiteturais estabelecidos na Seção 4.2. Na Figura 22 é ilustrado o processo da abordagem supracitada, onde é possível observar o fluxo para reutilização na íntegra de um módulo, reutilização com alteração de módulo e decomposição de um módulo (ou seja, reutilização parcial de um módulo). Uma vez identificado/definido, esse módulo passa por uma etapa de validação para que possa ser integrado à nova arquitetura de referência. Vale ressaltar que essas etapas ocorrem de maneira incremental/evolutiva para que os projetistas e demais interessados possam acompanhar a evolução do projeto da arquitetura de referência.

Figura 22 – Abordagem para construção de arquiteturas de referência

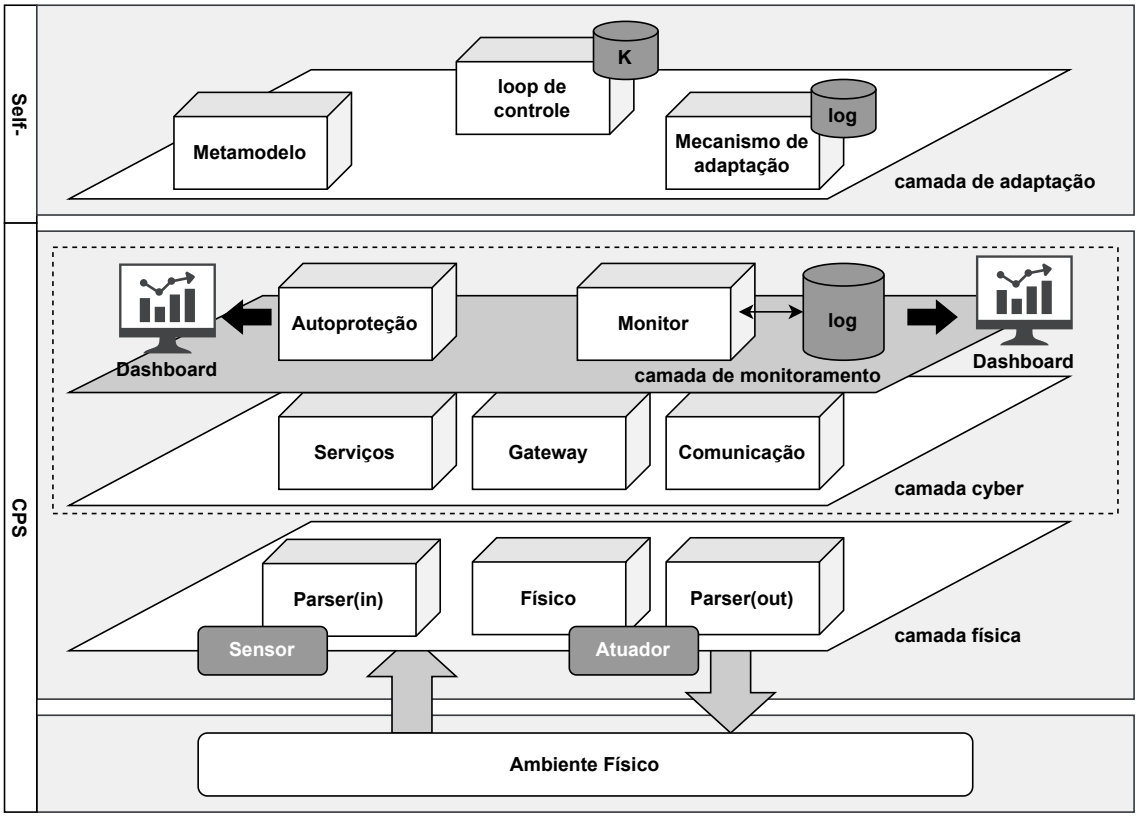


Adaptado de Tummers *et al.* (2021)

Com base no contexto exposto, a RA4Self-CPS foi projetada em uma visão *top-down* em camadas. Na Figura 23 é ilustrada uma visão geral da arquitetura, onde é possível observar as seguintes camadas: adaptação, *cyber* e física. A primeira representa a lógica para lidar com a

adaptação do sistema em tempo de execução. Já a segunda, representada pela linha pontilhada, representa a camada de comunicação do software com a camada física (terceira camada). Para que um software CPS tenha a capacidade de modificar sua estrutura e comportamento em tempo de execução ou até mesmo se ajustar aos cenários de contexto (ou seja, ambiente de execução), a camada de adaptação possui três módulos: metamodelo, *loop* de controle e mecanismos de adaptação. A camada *cyber* contém todo lógica de comunicação entre o CPS e a camada física, fornecendo módulos para elaboração de serviços, comunicação entre os componentes do CPS e direcionamento de rotas. Por se tratar de uma camada que lida com sistemas distribuídos via serviços, esta camada também prevê a elaboração de módulos de monitoramento de serviços e proteção dos mesmos via chamadas HTTP (do inglês, *HyperText Transfer Protocol*). Por fim, a camada física é encarregada de permitir a comunicação entre o ambiente físico e o CPS por meio dos sensores e atuadores. O ambiente físico representa o lado físico de um CPS. A seguir, uma descrição sobre o propósito geral de cada camada é apresentada.

Figura 23 – Visão geral da RA4Self-CPS



Autoria própria

Na **camada física** foi estabelecida a organização para fornecer ao sistema meios de interação com o ambiente físico. Para isso, essa camada possui componentes sensores e atuadores para entrada e saída de dados de um Self-CPS. Em linhas gerais, essa camada deve ser responsável pelas atividades de sensoriamento e de atuação no ambiente de aplicação. Como esse tipo de sistema pode lidar com diferentes tipos de dados, mecanismos para tradução dos dados coletados

(módulo *Parser-in*) do ambiente e dados/ações devolvidas a este ambiente (módulo *Parser-out*) devem ser implementados.

A **camada *cyber*** deve ser responsável pela comunicação entre a **camada física** e o (Self-)CPS. Como essa camada tem uma natureza complexa do ponto de vista de conteúdo, optou-se por organizá-la em três módulos, a saber:

- **Comunicação.** O objetivo principal desse módulo é permitir a comunicação entre os subsistemas desenvolvidos, por meio da implementação das interfaces de comunicação, para que outras camadas possam consumir os dados fornecidos pela *camada física*. Como pode-se observar, esse módulo requer que sejam implementados diferentes interfaces de comunicação (ou seja, *endpoints*) para cada conjunto de dados a ser sensoriado. Assim, o ***módulo de serviços*** pode receber e persistir esses dados em um banco de dados apropriado. Esse módulo também viabiliza a troca de informações com os módulos de monitoramento e proteção de um Self-CPS. Dessa forma, esse módulo requer que sejam implementados, para cada processo desenvolvido, diferentes interfaces de comunicação para que o sistema possa usufruir dos dados persistidos e do poder computacional disponibilizado pelo ***módulo de serviços***.

- **Serviços.** Conforme já mencionado na Seção 2.4, um sistema CPS possui um baixo poder computacional, fazendo com que suas funcionalidades e sua área de aplicação sejam limitadas. Baseado nesse cenário, esse módulo tem como principal motivação enriquecer um CPS, disponibilizando um alto poder computacional a estes sistemas por meio de serviços. Para isso, os projetistas/desenvolvedores devem fazer a extração das regras de negócios do sistema alvo para que se possa elencar as regras que são inviáveis de executar em um CPS convencional e serem inseridas para processamento de terceiros. Exemplos desse tipo de atividade são (i) atividades de aprendizado de máquina aplicadas aos dados recebidos; (ii) atividades de aprendizado de máquina a nível de segurança, tendo como motivação proteger os serviços disponibilizados, (iii) processamentos que envolvem um alto grau de complexidade temporal e espacial devido à quantidade de dados retornados em uma consulta no banco de dados; entre outros.

- **Gateway.** O propósito deste módulo é fornecer um único ponto de controle, acesso e validação de solicitações de serviços da Web para que se possa controlar quais serviços estão disponíveis para diferentes grupos de usuários de um Self-CPS. Esse módulo permite que os serviços sejam disponibilizados pelo módulo **Serviços** para os interessados, os quais podem ser via barramento ou no formato de simples *endpoints*.

Ainda sobre a **camada *cyber***, nota-se que esta possui uma camada supervisora chamada **monitoramento**. Essa camada permite monitorar um Self-CPS sob dois interesses transversais:

qualidade de serviço e segurança. O primeiro pode ser considerado um elemento de primeira classe para sistemas de software que têm como base a comunicação via serviços, pois uma falha de degradação de qualidade pode resultar em um comportamento indesejado para um (Self-) CPS. Portanto, identificar e/ou substituir o serviço problemático para que aplicação não tenha resultados indesejados pode ser uma alternativa viável. Já o segundo pode ser considerado um elemento de extrema relevância também, uma vez que deve auxiliar na preservação da integridade do sistema alvo (Self-CPS). Esse módulo deve ser capaz de identificar requisições consideradas maliciosas e alertar o administrador sobre tais requisições. Por fim, este módulo deve fornecer ao usuário informações sobre o ambiente por meio de gráficos (ou seja, *dashboards*), notificações via mensagens de texto padronizadas sobre tópicos importantes, além de permitir que o usuário possa ajustar o sistema conforme suas necessidades.

A **camada de adaptação** permite que capacidades de adaptação possam ser incorporadas ao CPS, caracterizando esse tipo de sistema como um Self-CPS. Para isso, essa camada possui três módulos: (i) ***metamodelo***, que contém a estrutura do tipo de entidade que um sistema self-* pode manipular; (ii) ***loop de controle***, que é o mecanismo principal para adaptação das entidades de software de um sistema self-*; e (iii) ***mecanismo de adaptação***, que representa o processo para que uma entidade de software possa ser adaptada em tempo de execução. No que diz respeito aos tipos de adaptação suportados por um sistema, pode-se citar automonitoramento, autoconfiguração, autocura, auto-otimização e autoproteção. Em relação ao automonitoramento, abordagens de programação reflexiva, visando obter um contexto geral do componente de software monitorado em questão, e uso de arquivos de *logs* gerados pelas outras camadas, podem ser utilizados para satisfazer a primeira propriedade. A auto-otimização pode ser alcançada na camada **Serviços** por meio do uso de técnicas de aprendizado de máquina, que permitem identificar o cenário real do ambiente de execução, favorecendo o aprimoramento da geração de ações a serem enviadas ao efetuadores. A propriedade autoproteção pode ser alcançada via a implementação de algoritmos de aprendizado de máquina, tendo em vista aplicar uma barreira de proteção nas interfaces de comunicação implementadas na API, barrando as requisições anômalas. Por último, as propriedades de autoconfiguração e autocura podem ser concebidas ao software (Self-CPS) por meio de técnicas de programação reflexiva, sendo a primeira aplicada ao contexto da implementação dos serviços, visando modificar o comportamento do mesmo em tempo de execução, e o segundo possuindo como motivação alterar a estrutura de um componente de software em outras camadas visando sanar uma situação não prevista em tempo de *design*. Por fim, vale destacar que as propriedades self-* podem ser utilizadas separadas ou combinadas, dependendo do cenário de adaptação que se pretende realizar no sistema alvo (Self-CPS). A associação de propriedades pode elevar de maneira significativa a complexidade de desenvolvimento de um sistema. A seguir, detalhes sobre cada uma das camadas e seus respectivos módulos (veja Figura 23) são apresentados.

Ambiente físico

De acordo com Gerostathopoulos *et al.* (2016), CPS são sistemas que integram elementos computacionais e processos físicos que se comunicam por meio de uma rede de elementos distribuídos que respondem às atividades realizadas/ocorridas no mundo físico (ou seja, seu ambiente operacional). Esses sistemas podem combinar componentes mecânicos (por exemplo, máquinas) ou eletrônicos (por exemplo, sensores) com elementos de software via Internet para transferência e monitoramento de dados, formando um sistema unificado (CPS). Conforme exposto pelo autor, os CPS podem ser considerados mais modulares, dinâmicos, conectados em rede e de grande escala, se comparados aos sistemas embarcados tradicionais, os CPS estão se tornando mais modulares, dinâmicos, conectados em rede e de grande escala. Para isso, esses sistemas (CPS) têm se mostrado cada vez mais dependentes de software e de outros sistemas constituintes que fazem parte de um CPS.

Fazendo um paralelo entre os CPS e os sistemas baseados em IoT, pode-se dizer que esses sistemas apresentam algumas similaridades, por serem sistemas baseados na integração de elementos computacionais e objetos físicos (sensores e atuadores). A diferença entre esses sistemas está na não obrigatoriedade de conectividade com a Internet para os CPS. Já os sistemas baseados em IoT necessitam de comunicação via rede para entregar aos interessados os dados coletados do ambiente. Portanto, pode-se dizer que as aplicações baseadas em IoT são um subconjunto de um sistema CPS. Outro ponto a ser destacado entre esses sistemas é o avanço da área de IoT, pois a sinergia existente com a área de CPS tem modificado a cooperação dos mundos virtual e físico, fazendo com que os objetos se tornem mais inteligente (MATHUR; ARORA, 2020).

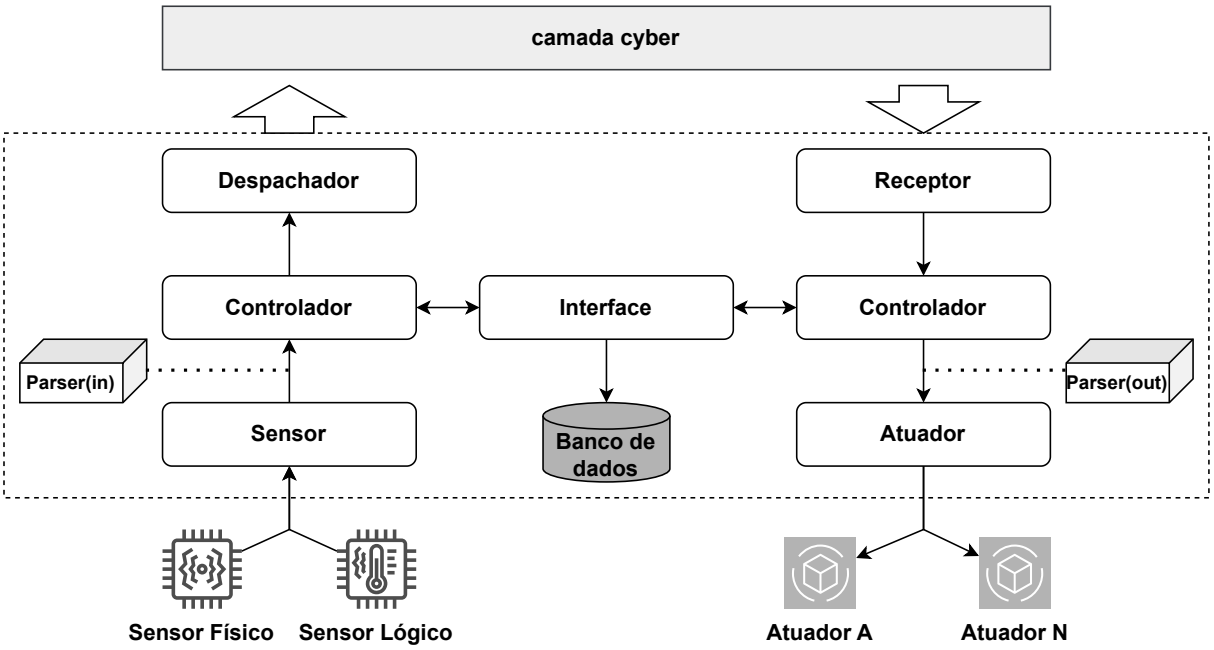
Com base no contexto exposto e nos resultados do mapeamento apresentado na Seção 3.2, o ambiente operacional de um Self-CPS pode atender diferentes domínios de aplicação, inclusive o domínio de saúde. Como cada tipo de aplicação possui natureza específica quanto ao número de equipamentos e sensores (entre outros elementos), a definição do ambiente operacional deve estar em conformidade (e pode variar) de acordo com cada aplicação. Diante da diversidade de elementos que podem estar presentes no ambiente físico de um Self-CPS, a **camada física** da RA4Self-CPS deve prover mecanismos capazes de realizar comunicação entre o ambiente e o software (camadas superiores) para funcionamento adequado do sistema.

Camada física

Conforme reportado na Seção 4.3, esta camada tem por finalidade permitir a interação entre a camada que representa o ambiente físico (camada **ambiente físico**) e a camada **cyber** de um Self-CPS. A Figura 24 ilustra os elementos da camada física para a arquitetura apresentada na Figura 23. Como pode-se observar, a **camada física** está representada pela camada intermediária

(linha pontilhada), estabelecendo interação entre os sensores (camada **ambiente físico**) e a camada ***cyber***. Em relação ao conteúdo da camada **ambiente físico**, os sensores físicos e lógicos representam, respectivamente, os componentes de hardware e de software que fazem parte de um Self-CPS. Já os atuadores representam todos os componentes de hardware que são encarregados de interagir e realizar modificações no ambiente físico. Detalhes sobre a camada **cyber** serão apresentados na próxima seção. A seguir, uma descrição dos componentes da camada **física** é apresentada.

Figura 24 – Componentes da camada física



Autoria própria

Sensor e Atuador. O componente **Sensor** pode ser considerado como a porta de entrada para um Self-CPS, pois é o responsável pela captura de dados da camada que representa o ambiente operacional. De modo oposto ao **Sensor**, o componente **Atuador** pode ser considerado como a porta de saída para um Self-CPS, uma vez que sua função é encaminhar os dados que serão executados no ambiente operacional.

Controlador. Os componentes controladores são responsáveis por receber os dados vindos dos sensores e, após atividades de execução de um Self-CPS, devolver esses dados processados para o ambiente de execução, regendo os fluxos de execução dessa camada. Nota-se que a comunicação entre os componentes **Sensor** e **Atuador** com o **Controlador** pode ser interceptada pelos módulos **Parser(in)** e **Parser(out)**.

Parser(in) e Parser(out). Esses módulos têm por finalidade realizar a tradução de dados entre a camada **ambiente físico** e as demais camadas da arquitetura. Em resumo, essa operação

é realizada pela necessidade de compatibilizar os dados coletados do ambiente aos dados que serão utilizados pelos algoritmos de inteligência artificial para tomada de decisão, identificação de anomalias, classificação de dados, entre outras operações.

Interface. Este componente trabalha em paralelo ao componente **Controlador**, sendo responsável por armazenar os dados coletados do ambiente físico via sensores e atuadores em uma base de dados. Esses dados são utilizados pelos sistemas das camadas superiores para gerenciamento de um Self-CPS em tempo de execução.

Banco de dados. Este componente representa o banco de dados propriamente dito, cuja finalidade é armazenar temporariamente os dados coletados do ambiente de execução (camada **ambiente físico**) em formato apropriado para as camadas superiores da RA4Self-CPS.

Despachador e Receptor. O componente **Despachador** tem por objetivo fornecer os dados coletados do ambiente de execução para a camada **cyber** em formato de serviços Web. Em contrapartida, o componente **Receptor** permite que os dados processados das camadas superiores sejam recebidos e encaminhados para os componentes da camada **física** para serem enviados até o ambiente operacional (camada **ambiente físico**).

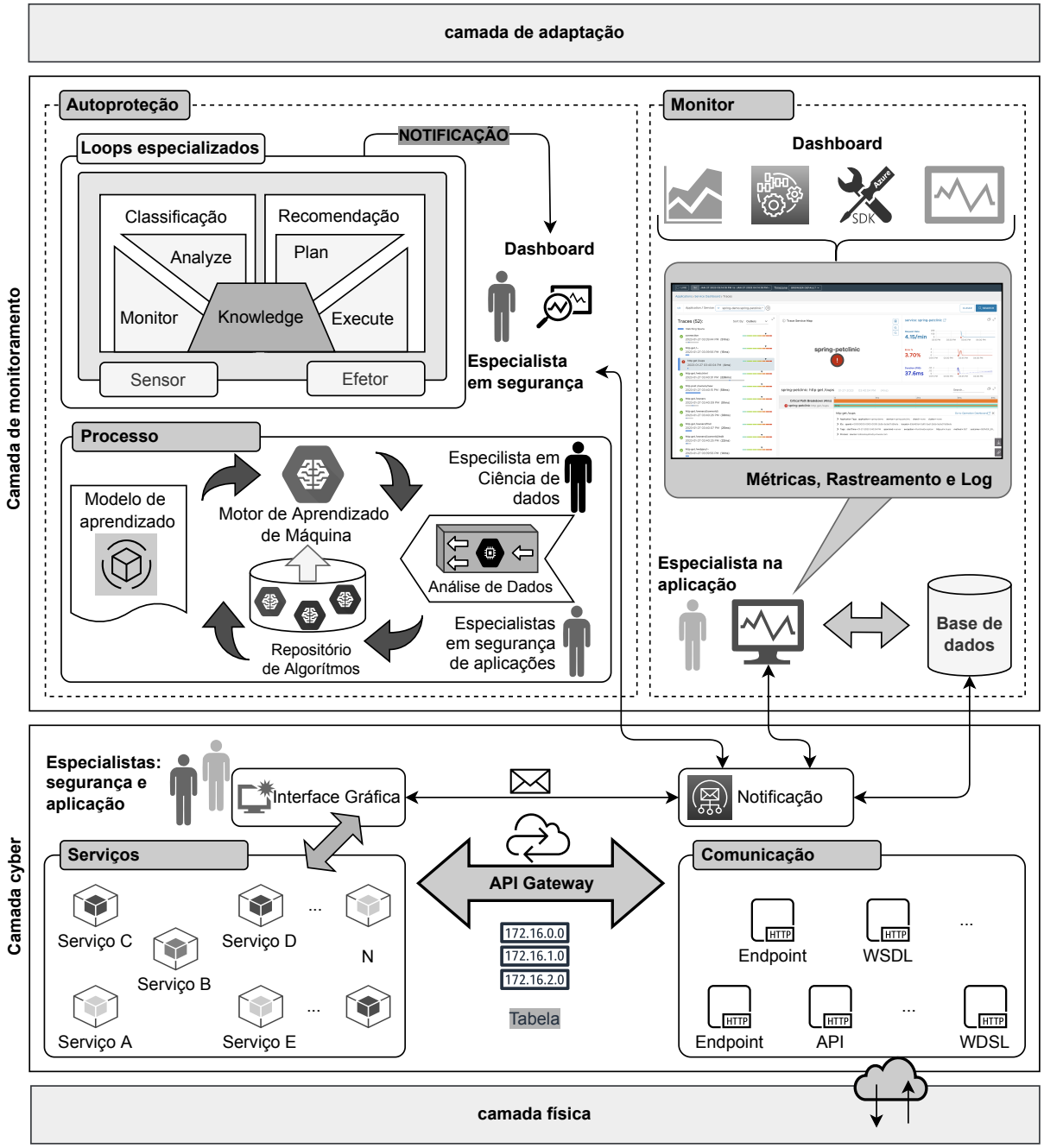
Camada *cyber*

Esta camada tem por objetivo viabilizar a comunicação entre a camada **física** (ambiente físico) e o software de um Self-CPS. Na Figura 25 são ilustrados os elementos da camada **cyber** para a arquitetura apresentada na Figura 23. Em resumo, a camada **cyber** está representada pelas camadas intermediárias (ou seja, **cyber** e de **monitoramento**), estabelecendo interação entre as camadas **física** e de **adaptação**. Conforme exposto na Seção 4.3, a camada **física** tem por finalidade entregar os dados coletados do ambiente de execução, via componente *Despachador* (veja Figura 24), no formato adequado para serem processados pela camada **cyber**. Uma vez processados, esses dados são devolvidos para a camada **física** através do componente **Receptor**. Durante a etapa de processamento, alterações (por exemplo, detecção de anomalias e/ou degradação de qualidade) na aplicação (Self-CPS) podem ser detectadas pela camada de **adaptação**. Tais alterações devem ser analisadas e corrigidas para que danos ou interrupções na aplicação alvo não sejam gerados. A seguir, uma descrição dos componentes da camada **cyber** é apresentada.

Comunicação. Este componente tem por objetivo fornecer para um Self-CPS as interfaces de comunicação para os serviços (ou seja, implementação concreta) que irão realizar o processamento de uma operação. Como pode-se observar na Figura 25, três tipos de comunicação são ilustrados, a saber: (i) *Endpoint* para comunicação via serviços REST[1];

[1] do inglês, *Representation State Transfer*

Figura 25 – Componentes da camada cyber



Autoria própria

(ii) WSDL (do inglês, *Web Services Description Language*) para comunicação via serviços SOAP; e (iii) API, que representa um tipo de comunicação genérica que pode não ser nenhum dos anteriores. Vale destacar que essas interfaces devem ser organizadas em um catálogo de serviços para ser consultado pelos demais componentes da RA4Self-CPS.

Serviços. Este componente representa a implementação concreta dos serviços utilizados pelo Self-CPS. Para viabilizar a autoadaptação em um Self-CPS quanto aos serviços, o con-

ceito[2] de serviço primário e alternativo proposto por (PASSINI; AFFONSO, 2018a; PASSINI; AFFONSO, 2018b; PASSINI, 2020; PASSINI *et al.*, 2021) pode ser utilizado. Esse conceito foi empregado no desenvolvimento de um *framework*, que permite lidar tanto com serviços do tipo REST e/ou SOAP em comunicação por orquestração e/ou coreografia. Para isso, o projetista de um Self-CPS pode definir um **`serviço W`** para executar uma funcionalidade no contexto de um Self-CPS que necessita de outros serviços, caracterizando um serviço composto. Esse **`serviço W`** pode atuar como um orquestrador, realizando chamadas para os serviços A, B e C (serviços primários). Portanto, para que esse **`serviço W`** seja executado corretamente, os demais também devem apresentar funcionamento normal. Com base nesse cenário, se houver algum tipo de anomalia (ou seja, falha de execução ou degradação de qualidade) em apenas um dos serviços, a aplicação cliente poderá ser comprometida. Nesse sentido, o *framework* supracitado permite que sejam definidos para cada serviço primário (A, B, C), um ou mais serviços secundários (por exemplo, A1, A2, ..., AN) na fase de projeto da aplicação. Assim, quando houver algum tipo de anomalia em um serviço primário (por exemplo, A) em tempo de execução, o *framework* efetua a substituição desse serviço por um secundário (por exemplo, A1) e a aplicação não tem a percepção desse tipo de operação.

API Gateway. Conforme exposto nesta seção, o desenvolvimento de um Self-CPS pode exigir a utilização de um conjunto de serviços. Nesse sentido, desenhar uma solução capaz de ser resiliente quanto aos diversos tipos de falhas e/ou degradações de qualidade pode ter um *gateway* como aliado. Conforme descrito na Seção 4.3, esse componente visa facilitar a comunicação/interação entre os serviços que fazem parte de um Self-CPS. Para isso, o serviço de *gateway* deve ser disponibilizado em um endereço diferente para o serviço destino, fazendo com que se possa substituir ou realocar o serviço destino sem alterar os detalhes do serviço de *gateway* associado. Além disso, o componente **API Gateway** deve prover uma interface comum para os serviços, em que os interessados não precisam saber onde cada serviço subjacente está localizado (por exemplo, serviço interno ou externo ao ambiente operacional do Self-CPS). As funcionalidades do componente API Gateway podem ser sintetizadas da seguinte maneira: (i) controle de tráfego de chamadas entre os serviços; (ii) porta de entrada única para integração entre o meio físico e o software; (iii) roteamento de rotas; (iv) mecanismo de segurança pela autenticação de usuário, limite de conexões e registro (logs) de acesso. Dependendo do nível de complexidade de um Self-CPS, outras funcionalidades podem ser atribuídas a esse componente.

Conforme ilustrado na Figura 23, a camada **cyber** foi projetada para lidar com os interesses de software de um Self-CPS. Como os interesses desse tipo de sistema podem ser

[2] Vale ressaltar que outras técnicas/conceitos podem ser utilizados, desde que permitam lidar com a modificação de serviços em tempo de execução.

considerados complexos, do ponto de vista de desenvolvimento, e críticos, pela própria natureza, monitorar o estado de execução e segurança de (Self-)CPS tem sido um aspecto identificado no mapeamento apresentado na Seção 3.2. Nesse sentido, optou-se por criar uma camada supervisora para a camada **cyber** chamada **monitoramento**, a qual possui dois módulos, a saber: Autoproteção e Monitor. A seguir, uma descrição de cada módulo é apresentada.

O módulo **Autoproteção** (veja Figura 25 – linha pontilhada) tem por objetivo analisar todas as requisições HTTP de um Self-CPS e identificar se as mesmas representam algum tipo de ameaça/vulnerabilidade. Para isso, optou-se por utilizar a abordagem proposta por Martins *et al.* (2021), Martins (2022), que viabiliza autoproteção na camada de aplicação em aplicações Web e orientadas a serviços. A opção por essa abordagem está relacionada a dois fatores: (i) flexibilidade e escalabilidade da abordagem para elaboração de classificação de anomalias; e (ii) experiência anterior de nosso grupo de pesquisa. Esses fatores podem ser considerados facilitadores para entendimento da abordagem e utilização da mesma. Diante do exposto, vale destacar que outras alternativas de segurança podem ser utilizadas na arquitetura proposta neste trabalho (RA4Self-CPS), além de outros níveis de segurança que também podem ser inseridos nessa arquitetura.

A solução de autoproteção adotada em nossa arquitetura requer a participação de dois especialistas: segurança e ciência de dados. O primeiro, juntamente com o projetista de um Self-CPS, define os pontos de monitoramento que irão captar informações das requisições. Essas informações serão utilizadas para elaboração do modelo de classificação para cada tipo de ataque. Para isso, é necessário a participação de um especialista na área de segurança e ciências de dados para que os dados sejam tratados de maneira adequada na etapa de criação dos modelos de aprendizado. Como pode-se observar na Figura 25, a elaboração dos modelos é uma atividade cíclica, pois os modelos são gerados e testados quanto sua precisão (e outras medidas) e comportamentos. Portanto, casos insatisfatórios fazem com que o processo seja reiniciado até que um modelo seja elaborado. Uma vez definido, esses modelos são inseridos nos *loops* especializados para serem implantados no ambiente de execução. Nesse ambiente, esses *loops* devem fornecer aos especialistas de segurança informações sobre as requisições para que medidas protetivas sejam por eles tomadas no caso de detecção de algum tipo de anomalia.

O módulo **Monitor** tem por finalidade apresentar aos interessados (ou seja, especialistas de segurança e da aplicação) o estado atual do sistema (Self-CPS) por meio de um conjunto de métricas, informações de rastreamento e *logs*. Em resumo, essas informações podem ser úteis para identificação de algum tipo de anomalia que possa comprometer o funcionamento do sistema. Além disso, esse módulo deve permitir a interação entre os interessados e o Self-CPS, seja no sentido de modificar determinados comportamentos e/ou de estar ciente acerca da situação do ambiente monitorado. Para isso, três elementos devem ser implementados: (i) interface gráfica (camada **cyber**), a qual engloba um conjunto de ferramentas para o desenvolvimento de com-

ponentes de software e gráficos para um Self-CPS. Em resumo, a interface gráfica permite que os interessados possam interagir com o sistema para realizar ajustes em certos comportamentos de sua arquitetura e /ou componentes de software do sistema; (ii) notificação (camada **cyber**), encarregado por analisar os dados e alertar os interessados, por meio de notificações concisas, sobre situações que merecem atenção e/ou interferência humana; e, por fim, (iii) monitor gráfico (*dashboards* – camada de **monitoramento**), cuja função principal é apresentar e informar aos interessados de forma concisa o status de execução do Self-CPS, além da situação corrente do ambiente de aplicação.

No que diz respeito ao conjunto de informações (ou seja, métricas[3], informações de rastreamento[4] e *logs*[5]) que o módulo **Monitor** pode manipular para lidar com a observabilidade de um Self-CPS, este deve permitir que os interessados possam analisar seu comportamento diante de ações maliciosas e/ou normais de execução. O aumento repentino e significativo em requisições para um determinado serviço e o registro de "injeção" de comando capturado pelos analisadores de dados (*logs*) são exemplos de ações que podem ser maliciosas para um Self-CPS. Em contrapartida, existem cenários normais em termos de execução que também devem ser informados aos interessados da aplicação para que danos severos e/ou involuntários não ocorram. Nessa direção, vale ressaltar que existem diversas soluções disponíveis na literatura para realizar a observabilidade de aplicações Web e/ou orientada a serviços. Wavefront[6], Grafana[7] e Zipkin[8] são alguns exemplos de soluções que podem ser utilizadas em um Self-CPS para lidar com a observabilidade, pois essas soluções apresentam relativa facilidade de integração com *frameworks* de desenvolvimento corporativo (por exemplo, Spring Boot[9]).

Camada de adaptação

A camada de **adaptação** deve ser projetada para atender os interesses de adaptação de um Self-CPS em tempo de execução. De acordo com o conteúdo apresentado na Seção 2.3, o software que requer modificação em tempo de execução pode ser projetado para atender uma (ou mais) propriedade self-*. À medida que o número dessas propriedades é elevado em qualquer domínio de sistema, a complexidade de desenvolvimento também aumenta em proporção equivalente ou até mesmo superior. O *loop* de controle proposto pela IBM (2005) tem sido adotado pela

[3] As métricas podem ser definidas como um medidor lógico, ou seja, um contador ou histograma ao longo de um período de tempo (TOZZI, 2023).

[4] As informações de rastreamento lidam com dados com escopo de solicitação, ou seja, qualquer *bit* de dados ou metadados que pode(m) ser vinculado(s) ao ciclo de vida de um único objeto transacional em um sistema (TOZZI, 2023).

[5] Os *logs* lidam com eventos discretos que ocorrem durante a execução de um sistema como mensagens de erro, eventos de trilha de auditoria, metadados específicos de solicitação, entre outros (TOZZI, 2023).

[6] <https://docs.wavefront.com/index.html>

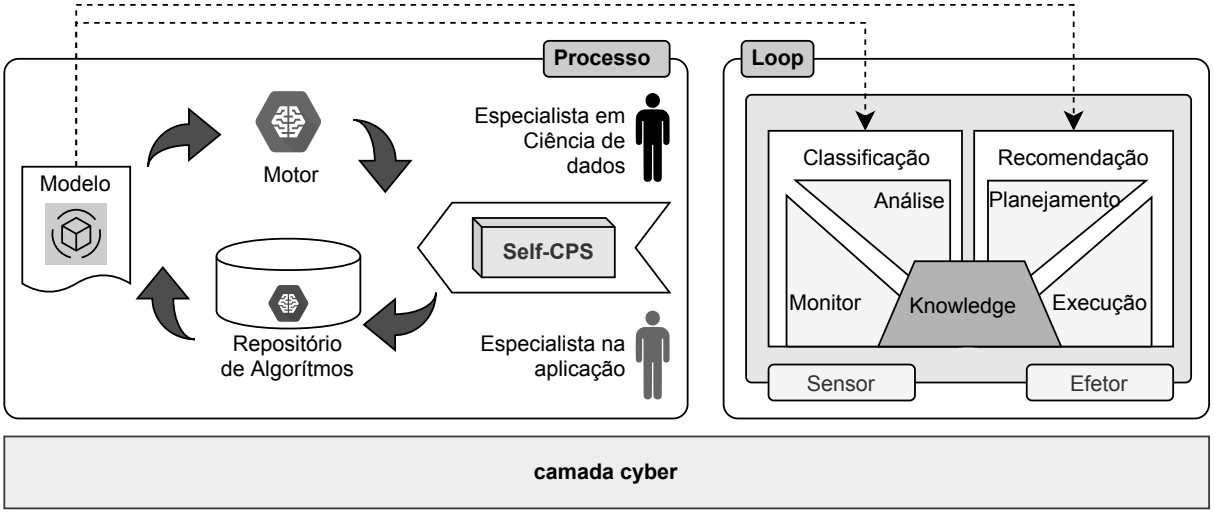
[7] <https://grafana.com/>

[8] <https://zipkin.io/>

[9] <https://spring.io/projects/spring-boot>

comunidade de SaS para lidar com os interesses de adaptação. Em experiências anteriores de nosso grupo de pesquisa (AFFONSO *et al.*, 2015), esse *loop* foi organizado em dois módulos para lidar com os interesses de adaptação de SaS, a saber: classificação e recomendação. A Figura 26 ilustra a organização da camada de **adaptação** para a RA4Self-CPS, com base no *loop* de controle MAPE-K e na organização supracitada (AFFONSO *et al.*, 2015). Observa-se que a camada de adaptação possui duas etapas, sendo a primeira para elaboração dos modelos de classificação e recomendação que serão utilizados pelos *loops* de controle (segunda etapa). A seguir, uma descrição de cada etapa é apresentada.

Figura 26 – Componentes da camada de adaptação



Autoria própria

Na primeira etapa (**Processo**) ocorre a elaboração dos modelos de classificação e recomendação que serão utilizados para lidar com os interesses de adaptação de um Self-CPS. Para isso, como passo preliminar, os especialistas devem definir quais são as modificações que esse sistema deve suportar e quais propriedades self-* que serão utilizadas. Nessa perspectiva, eles podem/devem utilizar o modelo 5W1H (veja Seção 2.3) como mecanismo norteador dos interesses de adaptação em relação ao software alvo. O próximo passo é definir os modelos de classificação e recomendação, com base nos interesses de adaptação (ou seja, propriedade self-*) e conjunto de dados que o sistema (Self-CPS) produz durante seu ciclo de execução. Nesse passo, os especialistas podem lidar com diferentes tipos de algoritmos e estratégias de aprendizado, por isso a representação cíclica para geração dos modelos. Em resumo, essa etapa requer avaliação a cada ciclo de interação para avaliação do modelo gerado, que pode resultar em ajustes nos dados (base de conhecimento) para que os objetivos sejam satisfeitos.

A segunda etapa (**Loop**) representa a instância de um *loop* de controle, com base nos modelos de classificação e recomendação elaborados na etapa de **Processo**. A instância do **módulo de classificação** está associada a três etapas do *loop* proposto pela IBM (IBM, 2005),

a saber: sensor, monitor e análise. Basicamente, para que a classificação seja realizada, esse módulo deve adquirir conhecimento (dados) resultante entre a interação do sistema (Self-CPS) e ambiente externo para que se possa verificar se os objetivos estão sendo atingidos e a maneira pela qual isso ocorre. Em seguida, analisar os dados coletados a partir do histórico do sistema e/ou dados adquiridos no monitoramento. Já a instância do **módulo de recomendação** está associada a três etapas, a saber: planejamento, execução e efetor. Após reconhecer o cenário atual do sistema pelo módulo de classificação, o módulo de recomendação deve determinar ***o que*** deve ser adaptado e ***quais*** serão as ações necessárias para realizar a operação de adaptação. Vale ressaltar que ambos os módulos fazem uso de uma base de conhecimento compartilhada entre todas as etapas com o intuito de melhorar a performance da execução das mesmas. Em relação à natureza dos dados, pode-se dizer que os mesmos podem ser variados quanto ao tipo e tamanho, dependendo do domínio da aplicação que está sendo instanciada.

4.4 Passo AR-4: Avaliação da arquitetura de referência

Como forma de avaliar a qualidade da RA4Self-CPS, uma inspeção baseada em *checklist* foi realizada. Em resumo, essa inspeção teve por objetivo avaliar se os requisitos propostos (veja Seção 4.2) foram representados na arquitetura. Esse tipo de atividade permite identificar problemas relacionados à omissão, ambiguidade de requisitos, incoerência quanto ao tipo de informações que podem estar presentes ou ausentes na arquitetura proposta. Além disso, visando observar a viabilidade da arquitetura proposta e sua capacidade de desenvolver Self-CPS, um estudo de caso foi conduzido, sendo os detalhes de implementação e resultados reportados no Capítulo 5. Esse estudo de caso também permitiu extrair evidências sobre a adaptação/reutilização de módulos de outros projetos de nosso grupo de pesquisa que foram integrados na arquitetura proposta neste projeto de mestrado acadêmico.

4.5 Considerações finais

Diante dos desafios inerentes ao desenvolvimento de (Self-)CPS (por exemplo, monitoramento constante do ambiente de execução, e incertezas de requisitos em tempo de execução), neste capítulo foi apresentada uma arquitetura de referência para Self-CPS para mitigar tais desafios. A principal motivação para integração de adaptação ao contexto de CPS deve-se, principalmente, aos benefícios de flexibilidade e eficiência diante de ambientes dinâmicos, críticos e imprevisíveis. Visando alcançar o objetivo de elaborar uma arquitetura de referência, que está em consonância com as tendências deste domínio (CPS) e com as melhores práticas de engenharia, o processo denominado PROSA-RA foi aplicado em conjunto com a abordagem proposta por Tummers *et al.* (2021). Como resultado, uma arquitetura de referência denomi-

nada RA4Self-CPS foi elaborada, a qual foi organizada em três camadas, como descrito neste capítulo.

5 Estudo de Caso

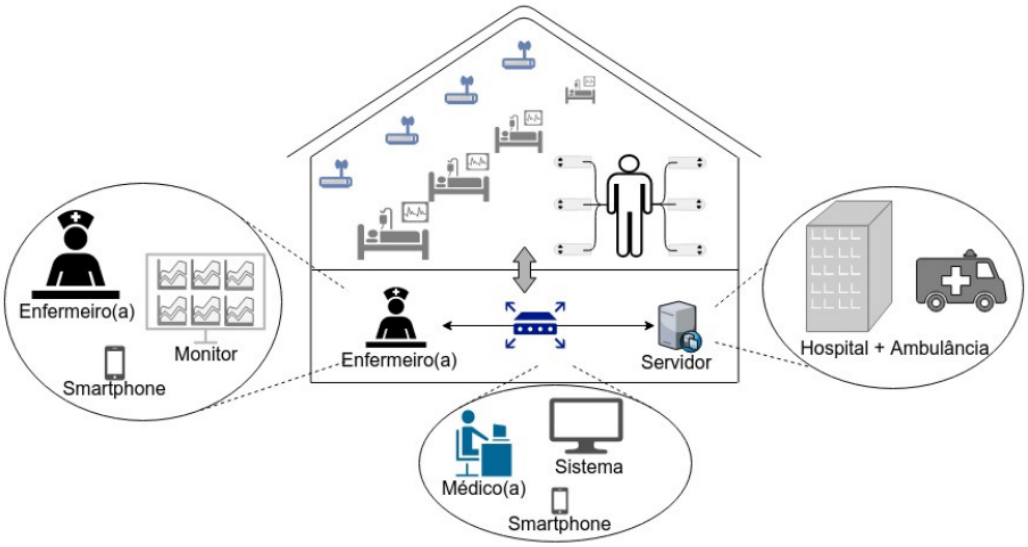
O objetivo deste capítulo é apresentar o estudo de caso que foi elaborado para avaliar a viabilidade, os pontos fortes e fracos da RA4Self-CPS apresentada no Capítulo 4. Para isso, este capítulo foi organizado da seguinte maneira: na Seção 5.1 é apresentada uma visão geral do estudo de caso, onde são detalhados os componentes e as tecnologias utilizadas; na Seção 5.2 é apresentada a instância da RA4Self-CPS como uma arquitetura concreta; e, por fim, na Seção 5.3 são reportadas as considerações finais do capítulo.

5.1 Visão geral

Como parte do processo de avaliação da RA4Self-CPS, um estudo de caso foi conduzido visando instanciar tal arquitetura de referência em uma arquitetura de software concreta, além de analisar o comportamento dos módulos desenvolvidos frente a um domínio de aplicação (Self-CPS). Dessa forma, um sistema voltado para o domínio de saúde (ou seja, *Smart Home Health Care* – SHHC) foi selecionado, visando explorar diferentes cenários de execução que podem cobrir falhas e monitoramento de sensores, vulnerabilidades e segurança da aplicação, comunicação e tempo de resposta, dentre outros fatores. Em síntese, cenários desse natureza maximizam as características de um CPS, pois integram a questão dos sensores e atuadores, dependem da abordagem do monitoramento constante para que o sistema possa estar ciente do contexto (ou seja, ambiente de execução, além de apresentar um forte vínculo com elementos da tecnologias de informação e comunicação. Na Figura 27 é ilustrada uma visão geral para o sistema SHHC, na qual é possível observar quatro elementos principais, a saber: paciente, enfermeiro(a), médico(a) e hospital (ou serviço de emergência). Em resumo, o sistema SHHC tem como motivação contribuir para a qualidade da saúde dos pacientes em um ambiente doméstico, contendo uma infraestrutura básica para monitorar a saúde de uma pessoa com algum tipo de enfermidade. Para alcançar este objetivo, é necessário coletar alguns dados da saúde desse paciente para serem processados para atender diversos propósitos, tais como: estabelecer um histórico detalhado do paciente, analisar tendências para determinados tipos de doenças, averiguar a saúde atual do paciente, entre outros fatores. Além da coleta via sensores, também fazem parte desse tipo de sistema, plataformas fornecidas pelo provedor do serviço (por exemplo, *dashboads* para monitoramento, sistemas Web) e aplicativos para dispositivos portáteis (por exemplo, *smartphones*, *smartwatches*). Por fim, vale destacar que esse sistema foi instanciado em um ambiente simulado de computação, onde os sensores que coletam informações sobre os pacientes foram implementados como APIs do tipo REST. A seguir, uma descrição sobre a organização modular do sistema SHHC é apresentada, assim como as tecnologias utilizadas no

desenvolvimento.

Figura 27 – Visão geral do sistema SHHC



Autoria própria

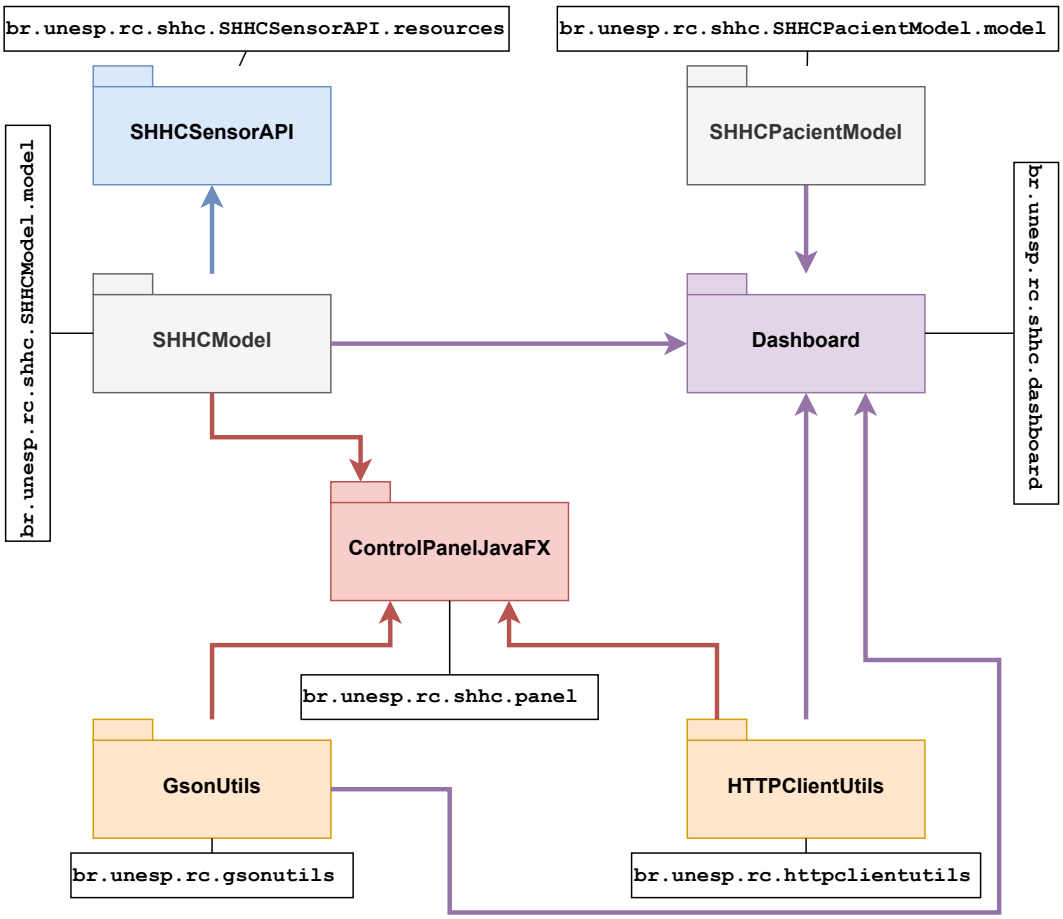
Em relação aos elementos ilustrados na Figura 27, esse sistema foi organizado em sete módulos, como ilustrado na Figura 28. Os módulos `GsonUtils` e `HTTPClientUtils` implementam funcionalidades de bibliotecas de terceiros que permitem lidar, respectivamente, com a serialização de objetos em formato JSON e realizar chamadas HTTP da aplicação cliente para APIs do tipo REST. Os demais módulos contêm a lógica do sistema SHHC, onde destacam-se os módulos `SHHCModel` e `SHHCPatientModel` por fornecerem classes lógicas para outros módulos. Essa estratégia de organização permite compartilhar/reutilizar as classes do sistema, melhorando aspectos de reutilização e manutenção do software. A seguir, uma descrição desses módulos é apresentada:

`SHHCSensorAPI`. Este módulo representa uma API REST desenvolvida com o *framework* Spring Boot[1]. Em síntese, essa API disponibiliza seis *endpoints* para simular o funcionamento dos seguintes sensores:

1. Fluxo de ar (*Air flow*);
2. Pressão sanguínea (*Blood pressure*);
3. Glicemia (*Glucose*);
4. Frequência cardíaca (*Heart rate*);
5. Oxigenação (*Pulse oxygen*);
6. Temperatura (*Temperature*).

[1] <https://spring.io/projects/spring-boot>

Figura 28 – Organização modular do sistema SHHC



Autoria própria

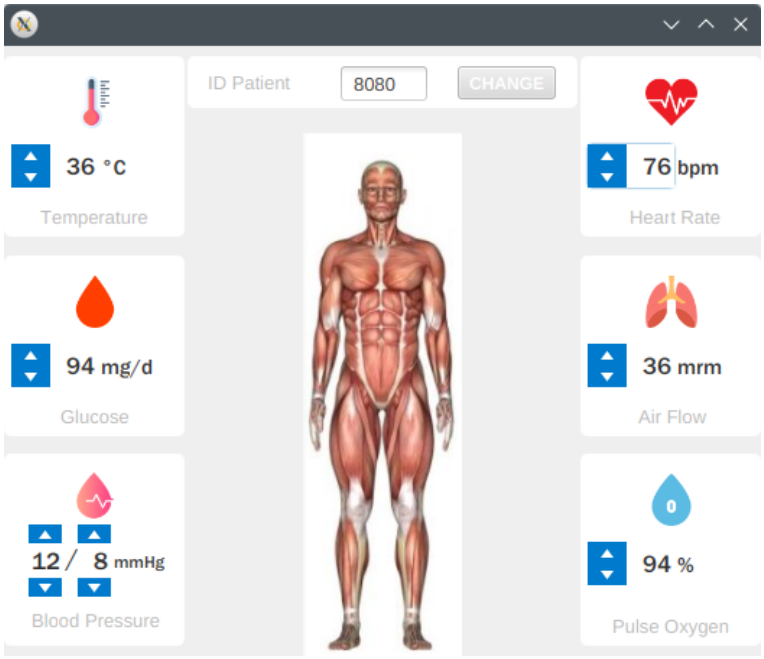
Em termos operacionais, cada API oferece dois tipos de operações, sendo uma para coletar/ler os dados de um sensor e outra para atualizar/salvar seu valor. Dessa forma, pode-se dizer que esse módulo tem por objetivo atuar como um paciente no contexto de um Self-CPS. Já em termos de implementação, alguns princípios da arquitetura REST foram incorporados nessas APIs, a saber: (i) comunicação *stateless*, (ii) identificação dos recursos disponibilizados, (iii) protocolo HTTP. Além disso, por fatores de simplificação, a API suporta somente dados representados no formato JSON (do inglês, *JavaScript Object Notation*) e não foi implementado nenhum mecanismo de autenticação/autorização dos clientes, como *OAuth* 2.0 (do inglês, *Open-Authorization*)[2] e JWT (do inglês, *JSON* Web Token)[3].

`ControlPanelJavaFX`. Este módulo representa uma aplicação JavaFX[4], que fornece um meio para alterar certos atributos do corpo humano, como ilustrado na Figura 29. O usuário dessa aplicação pode ajustar (ou seja, aumentar ou diminuir) os valores dos sensores

[2] <https://oauth.net/2/>
[3] <https://jwt.io/>
[4] <https://openjfx.io/>

supostamente acoplados ao paciente. Em relação aos cenários de testes, vale ressaltar que essa aplicação não contempla uma simulação baseada em cenários de enfermidade. Por exemplo, um paciente que apresenta um sintoma de febre pode ter sua frequência cardíaca alterada, além de outras alterações. Além disso, outros fatores devem ser levados em consideração para modelar cenários de pacientes, tais como: idade, peso, altura, índice de massa corpórea, entre outros. Nesse sentido, um profissional de saúde deve ser consultado para desenvolver cenários que possam se aproximar do funcionamento real desse tipo de sistema.

Figura 29 – Interface principal da aplicação *Control Panel*



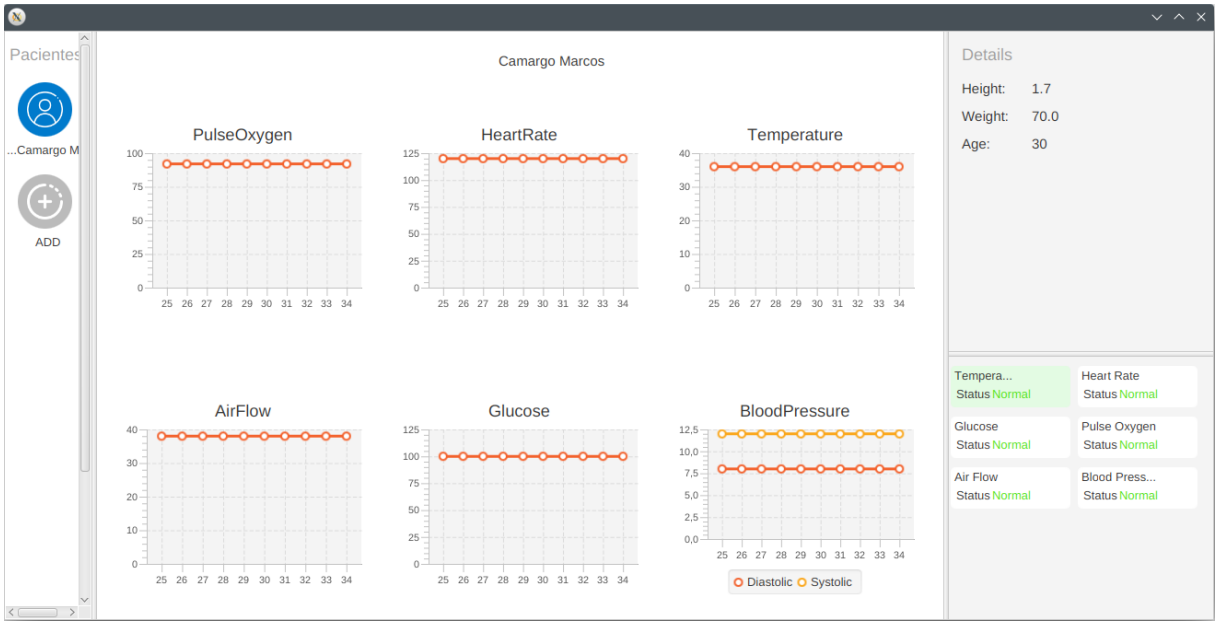
Autoria própria

`SHHCModel`. Este módulo contém as classes lógicas implementadas na linguagem de programação Java[5] para cada sensor citado no módulo `SHHCSensorAPI`. Conforme pode-se observar pela Figura 28 este módulo é utilizado ainda pelos módulos `ControlPanelJavaFX` e `Dashboard`.

`Dashboard`. Este módulo representa uma aplicação JavaFX que exibe seis gráficos, sendo um para cada sensor, como ilustra a Figura 30. Para isso, essa aplicação consome, a cada intervalo de tempo de 5 segundos, os valores vigentes dos sensores por meio de serviços REST, que são disponibilizados pelo módulo `SHHCSensorAPI`. Em relação ao processo de criação dos pacientes para realizar os cenários de simulação do sistema SHHC, optou-se por automatizar essa etapa em dois passos:

[5] <https://openjdk.org/projects/jdk/>

Figura 30 – Interface principal da aplicação *Dashboard*



Autoria própria

1. Inicialmente, uma imagem do módulo `SHHCSensorAPI` foi gerada via Docker[6]. Para isso, o seguinte comando deve ser executado:

   ```
   $ docker build -t shhcapi .
   ```

   O referido comando deve ser executado no diretório que contém o arquivo `Dockerfile` da aplicação, assim como o arquivo `.jar` da aplicação. Na Listagem 1 é apresentada a sequência de comandos para criação de uma imagem para o módulo `SHHCSensorAPI`.

   Listagem 1 – Sequência de comando do arquivo `Dockerfile`

   ```
   1  FROM openjdk:17
   2  RUN mkdir -p /app
   3
   4  ENV APP_NAME=SHHCSensorAPI-0.0.1-SNAPSHOT.jar
   5  COPY ${APP_NAME} /app/${APP_NAME}
   6
   7  EXPOSE 8080
   8  ENTRYPOINT [ "java", "-jar", "/app/SHHCSensorAPI-0.0.1-SNAPSHOT.jar" ]
   ```

   Fonte: Elaborado pelo autor

2. Em seguida, a cada paciente inserido no sistema SHHC (botão `Add` – Figura 30), uma nova imagem é criada em um contêiner que irá responder em uma porta diferente da anterior. Na Listagem 2 é apresentado o método responsável pela criação dos contêineres. Basicamente, este método automatiza os comandos que seriam execu-

6 <https://www.docker.com/>

tados via terminal de comandos através da classe `ProcessBuilder`[7] da linguagem de programação Java. Maiores detalhes sobre o funcionamento dessa classe podem ser obtidos na documentação da linguagem.

Listagem 2 – Método `createContainer.java`

```
1  public class ControllerView implements Initializable {
2      //...
3      private void createContainer(Patient newPaciente) {
4          String port = newPaciente.getPort() + ":8080";
5          String containerName =
               ↪ Integer.toString(newPaciente.getIdPaciente()) +
               ↪ newPaciente.getFirstName();
6          System.out.println(port);
7          try {
8              String dockerCommand = "docker";
9              String[] dockerArgs = {"run", "-p", port, "--name",
                   ↪ containerName, "shhcapi"};
10
11             // Criação do processo para executar o comando Docker
12             ProcessBuilder processBuilder = new
                   ↪ ProcessBuilder(dockerArgs);
13             processBuilder.command().add(0, dockerCommand);
14             processBuilder.inheritIO(); // Redireciona os streams de
                   ↪ entrada e saída padrão do processo Java para o
15             // processo Docker
16             processBuilder.start();
17
18         } catch (IOException e) {
19             e.printStackTrace();
20         }
21     }
22 }
```

Fonte: Elaborado pelo autor

Ainda sobre a funcionalidade do *dashboard* ilustrado na Figura 30, pode-se destacar o quadro de notificação no canto inferior à direita. Conforme pode ser observado, nessa área são exibidos os *status* de cada sensor com base em um modelo de classificação baseado em regras. Para ilustrar o comportamento desse modelo de classificação, apenas o sensor de temperatura será considerado. Assim, pode-se dizer que um paciente tem a temperatura "Normal" quando o valor é menor ou igual a 36 graus. O estado "Febril" é caracterizado por uma temperatura entre 36 e 37 graus. Já um paciente com "Febre" deve ter sua temperatura entre 37 e 38 graus. Na Listagem 3 são apresentadas as regras que permitem classificar os *status* descritos. O objetivo dessa funcionalidade no *dashboard* é exibir um alerta em tela sobre o estado de saúde do paciente e enviar uma notificação para

[7] <https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html>

o aplicativo móvel do profissional de saúde local (ou seja, enfermeiro(a)). Para atingir o objetivo funcional descrito, optou por utilizar o *framework* DROOLS[8] como um *engine* de processamento de regras. Por fim, vale destacar que um assistente que permite configurar e gerar os arquivos de regras foi desenvolvido pelo nosso grupo de pesquisa em trabalhos anteriores (veja Apêndice C). Como esse assistente não faz parte do sistema SHHC, os detalhes de funcionamento não serão apresentados nesta seção.

Listagem 3 – Trecho de código do arquivo `regrasTemperature.drl`

```
1  package br.unesp.rc.shhc.rules
2
3  import br.unesp.rc.shhc.SHHCModel.model.*;
4
5  rule "Temperatura-Normal"
6      when
7          temp : Temperature(value >= 36 && value < 37)
8      then
9          temp.setStatus("Normal");
10 end
11
12 rule "Temperatura-Febril"
13     when
14         temp : Temperature(value >= 37 && value < 38)
15      then
16         temp.setStatus("Febril");
17 end
18
19 rule "Temperatura-Febre"
20     when
21         temp : Temperature(value >= 38 && value < 39)
22     then
23         temp.setStatus("Febre");
24 end
```

Fonte: Elaborado pelo autor

`SHHCPacientModel`. Este módulo tem por objetivo armazenar a lista de pacientes que são inseridos no sistema SHHC. Para isso, implementa uma funcionalidade para garantia de unicidade do número de portas e identificadores dos pacientes. Como sua lógica de programação não apresenta elevado grau de complexidade, optou-se por não apresentar sua implementação por razões de espaço e escopo.

No que diz respeito à organização, pode-se dizer que o módulo `SHHCSensorAPI` é o mais complexo do sistema SHHC por apresentar dependência com os quatro outros módulos, a saber: `GsonUtils`, `HTTPClientUtils`, `SHHCModel` e `SHHCPatientModel`.

[8] <https://www.drools.org/>

5.2 Instanciando a RA4Self-CPS

O objetivo desta seção é apresentar os detalhes de instância da RA4Self-CPS (veja Figura 23) como uma arquitetura concreta para o sistema SHHC descrito na Seção 5.1.

Ambiente físico

Como o sistema SHHC atua em um ambiente simulado de computação, o ser humano, que está inserido no ambiente físico, foi representado pelo módulo `SHHCModel`. Esse módulo contém um conjunto de classes lógicas em Java que representam os sensores acoplados ao paciente e representam o **ambiente físico** de um Self-CPS. Por razões de espaço e objetividade na apresentação do sistema, somente o código fonte para um sensor será descrito neste estudo de caso porque os demais são análogos quanto ao desenvolvimento e generalização de conteúdo. Na Listagem 4 é apresentado o código fonte para a classe `Temperature.java`.

Listagem 4 – Código fonte da classe `Temperature.java`

```
1  package br.unesp.rc.shhc.SHHCModel.model;
2
3  public class Temperature {
4
5      private int value;
6      private String status;
7      private String ID;
8
9      public Temperature() {
10     }
11
12     public Temperature(int value, String status, String ID) {
13         this.value = value;
14         this.status = status;
15         System.out.println("ID" + ID);
16         this.ID = ID;
17     }
18
19     public int getValue() {
20         return value;
21     }
22
23     public void setValue(int value) {
24         this.value = value;
25     }
26
27     public String getStatus() {
28         return status;
29     }
30
31     public void setStatus(String status) {
```

```
32          this.status = status;
33      }
34
35      public String getID() {
36          return ID;
37      }
38
39      public void setID(String ID) {
40          this.ID = ID;
41      }
42
43      @Override
44      public String toString() {
45          return "Temperature{" + "value=" + this.value + ", status=" + this.status +
                ↪ ", ID=" + this.ID + '}';
46      }
47
48  }
```

Fonte: Elaborado pelo autor

Camada *cyber*

Em seguida, a classe `Temperature.java` (Listagem 4 – módulo `SHHCModel`) foi utilizada na implementação da API para simular o comportamento do sensor temperatura no módulo `SHHCSensorAPI`. Vale destacar que este último módulo foi implementado na **camada cyber**, representando especificamente o módulo de `Comunicação`. Na Listagem 5 é apresentado o código fonte para a classe `TemperatureResource.java`, onde é possível observar a rota de mapeamento (`@RequestMapping("/shhc/Temperature")`) para o sensor (linha 12), os *end-points* e métodos salvar e atualizar os dados do sensor. O *end-point* `@PutMapping("/update")` mapeia o método `updateTemperature` (linhas 27 a 32) e o *end-point* `@PostMapping("/save")` mapeia o método `saveTemperature` (linhas 34 a 39).

Listagem 5 – Código fonte da classe `TemperatureResource.java`

```
1  package br.unesp.rc.shhc.SHHCSensorAPI.resources;
2
3  import br.unesp.rc.shhc.SHHCModel.model.Temperature;
4  import org.springframework.web.bind.annotation.GetMapping;
5  import org.springframework.web.bind.annotation.PostMapping;
6  import org.springframework.web.bind.annotation.PutMapping;
7  import org.springframework.web.bind.annotation.RequestBody;
8  import org.springframework.web.bind.annotation.RequestMapping;
9  import org.springframework.web.bind.annotation.RestController;
10
11 @RestController
12 @RequestMapping("/shhc/Temperature")
13 public class TemperatureResource {
```

```
14
15      private static Temperature temp;
16
17      static {
18          temp = new Temperature(36, "Offline", "1");
19      }
20
21      @GetMapping("/")
22      public Temperature getTemperature() {
23          this.print();
24          return temp;
25      }
26
27      @PutMapping("/update")
28      public void updateTemperature(@RequestBody Temperature temperature) {
29          System.out.println("Value to update: " + temperature);
30          temp = temperature;
31          this.print();
32      }
33
34      @PostMapping("/save")
35      public void saveTemperature(@RequestBody Temperature temperature) {
36          System.out.println("Value to save: " + temperature);
37          temp = temperature;
38          this.print();
39      }
40
41      private void print() {
42          System.out.println("----------------------------------------");
43          System.out.println(temp);
44          System.out.println("----------------------------------------");
45      }
46  }
```

Fonte: Elaborado pelo autor

Diante do exposto nesta seção, alguns considerações devem ser destacadas sobre a instância dos módulos da RA4Self-CPS, a saber:

- Como o sistema SHHC foi desenvolvido em um ambiente simulado com base em apenas uma linguagem de programação, a instância dos módulos de tradução/mapeamento de dados não foi necessária (módulos `Parser(in)` e `Parser(out)`);
- O módulo que também não foi instanciado é o `Gateway` (**camada cyber**), pois optou-se por não definir mapeamento de rotas em caso de falhas nos sensores. Como os pacientes e sensores foram instanciados para serem executados em contêineres, a operação de parada de um contêiner já será identificada pelo módulo `Monitor` (**camada de monitoramento**) através do `Dashboard`;

- Por fim, o módulo `Serviços` não foi instanciado na totalidade para esse estudo de caso pela não necessidade de processamento de serviços de terceiros no sistema SHHC. No entanto, vale destacar que apenas os serviços que representam os sensores foram instanciados para utilização da ferramenta de observabilidade (veja **Camada de monitoramento: módulo Monitor**).

Conforme reportado na Seção 5.1, o módulo `ControlPanelJavaFX` foi desenvolvido para apoiar a simulação do comportamento do ser humano em cenários de algum tipo de enfermidade. Para isso, a aplicação desse módulo (veja Figura 29) foi acoplada ao módulo `SHHCSensorAPI`, permitindo modificar os valores dos sensores para cada paciente que está sendo monitorado. Detalhes sobre a implementação deste módulo (`ControlPanelJavaFX`) não serão apresentados nesta seção por essa aplicação ser complementar ao desenvolvimento do sistema SHHC e não ser um módulo integrante da RA4Self-CPS.

Camada de monitoramento: módulo Monitor

Visando explorar os pontos positivos e negativos do módulo `Monitor` (**camada de monitoramento**), um componente responsável pela concepção das **Dashboards** foi implementado com base no conceito de observabilidade de uma aplicação. Assim, esse **Dashboard** tem por objetivo lidar com os seguintes tipos de informações:

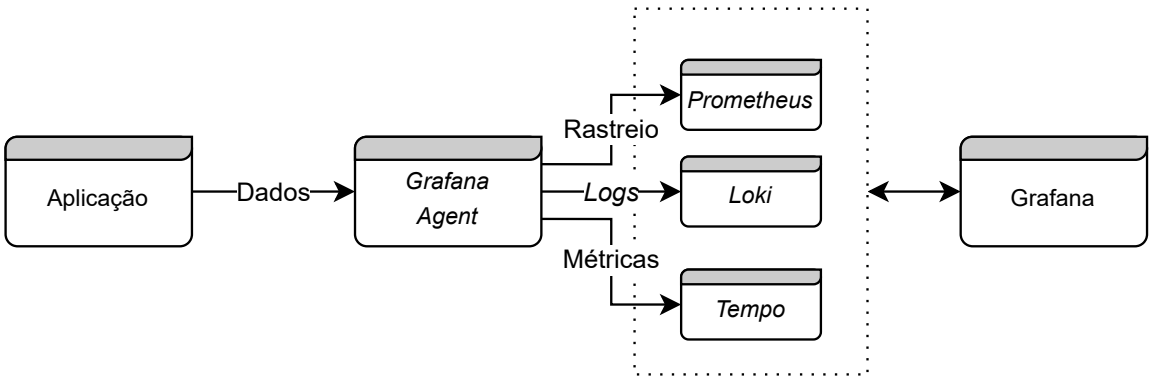
- **Rastreio.** Modo de representar uma série de eventos (ou seja, sequência e ordem) ocorridos durante uma operação realizada no software;
- **Métricas.** São representações numéricas de um objeto de estudo capturados em intervalos de tempo. Por intermédio destes dados, é possível obter ciência do estado do sistema em intervalos de tempo; e
- ***Logs*.** Composto por dados estruturados por texto que descreve em detalhes eventos ocorridos no software. Em resumo, estes dados atuam como um registro histórico de diversos tipos, como informações acerca do fluxo de execução realizado em um dado momento, e informações a respeito de comportamentos imprevisíveis, destacando os componentes que atuaram na sua execução.

Dentre as soluções disponíveis para aplicar o conceito de observabilidade, pode-se destacar *Wavefront*[9], *Grafana*[10], *SigNoz*[11] e *Dynatrace*[12]. Para a concepção deste estudo, a ferramenta *Grafana* foi selecionada de maneira arbitrária e sem qualquer tipo de viés tecnológico

[9] <https://docs.wavefront.com/wavefront_introduction.html>
[10] <https://grafana.com>
[11] <https://signoz.io>
[12] <https://www.dynatrace.com>

ou conhecimento prévio. Na Figura 31 é ilustrado o fluxo de observabilidade da ferramenta *Grafana*. Como pode ser observado nessa figura, a aplicação a ser observada é encarregada pelo envio de dados de natureza diversa. Neste estudo de caso, as APIs, em conjunto com os serviços implementados (ou seja, módulo `SHHCSensorAPI`), representam a aplicação a ser observada destacada (elemento **Aplicação**). Em seguida, o componente denominado *Grafana Agent* realiza a categorização dos dados, separando em dados de rastreio, *logs* e métricas, para que as aplicações responsáveis por cada tipo de dado realizem a captura. Por fim, a aplicação Web (elemento **Grafana**) exibe as tabelas e gráficos oriundos do processamento de dados dos componentes *Prometheus*, *Loki* e *Tempo*. A seguir, o monitoramento elaborado foi organizado de acordo com o tipo de dado trabalhado, a saber: rastreio, métricas e *logs*.

Figura 31 – Fluxo de observabilidade da ferramenta *Grafana*



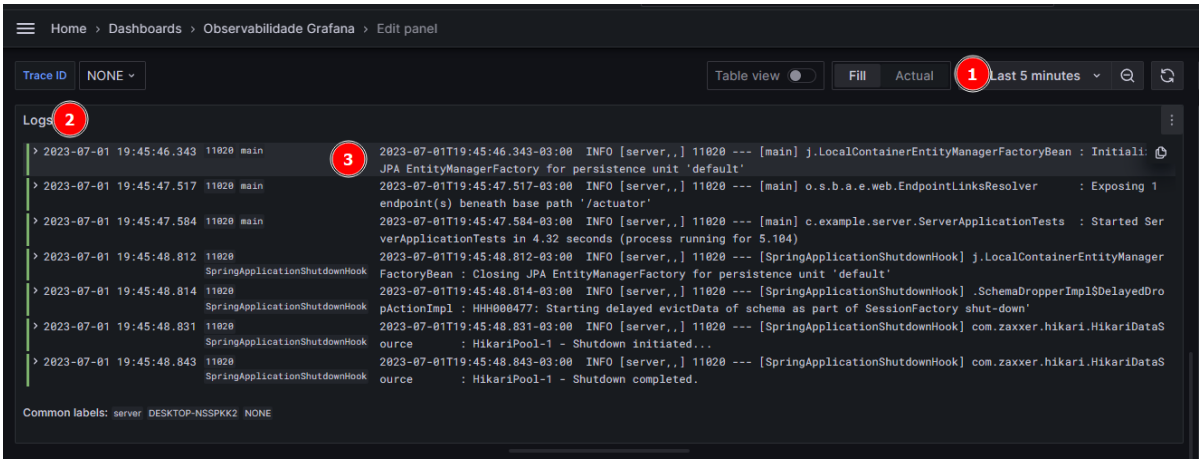
Adaptado de (VILLA, 2022)

Antes de iniciar a apresentação das informações do monitoramento, vale destacar que a ferramenta *Grafana* deve ser instalada no *host* que hospedará a aplicação e, posteriormente, configurada para que a observabilidade possa ser inicializada. Como esse procedimento é específico para cada solução, seus detalhes não serão reportados nesta seção. No entanto, esses detalhes podem ser visualizados no site oficial da ferramenta[13]. Após configurar a ferramenta, o *Dashboard* para registros de *logs* é apresentado no *Grafana* por meio dos dados processados pelo componente *Loki*, conforme evidenciado na Figura 32. O Indicador **1** indica que os registros capturados pela ferramenta são referentes aos últimos 5 minutos. Contudo, isso pode ser facilmente ajustado de acordo com os interesses do especialista da aplicação. Já Indicador **2** destaca o nome da tabela (neste caso, denominado de *Logs*) em que os registros serão apresentados e o Indicador **3** aponta para um registro de *log* em específico.

Como ilustrado na Figura 32, os registros de *logs* podem ser localizados por meio de mecanismos de manipulação de visualização dos registros, conforme ilustrado na Figura 33. Nessa figura é possível visualizar apenas uma manipulação sendo aplicada (Indicador **1**) por meio de uma operação denominada por filtragem por *label*. Em outras palavras, conforme

[13] <https://grafana.com/docs/grafana/latest/setup-grafana>

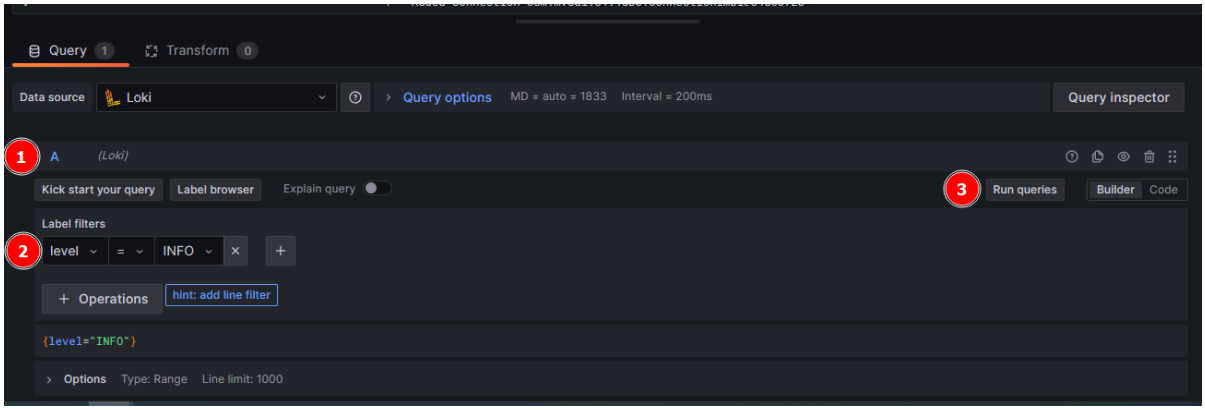
Figura 32 – Visão geral dos registros de *log*



Autoria Própria

ilustrado na referida figura, somente *logs* de natureza informativa (INFO) (Indicador **2**) serão relatados. Após a elaboração dos filtros, é necessário executar a busca dos registros (Indicador **3**).

Figura 33 – Manipulação dos registros de *logs*

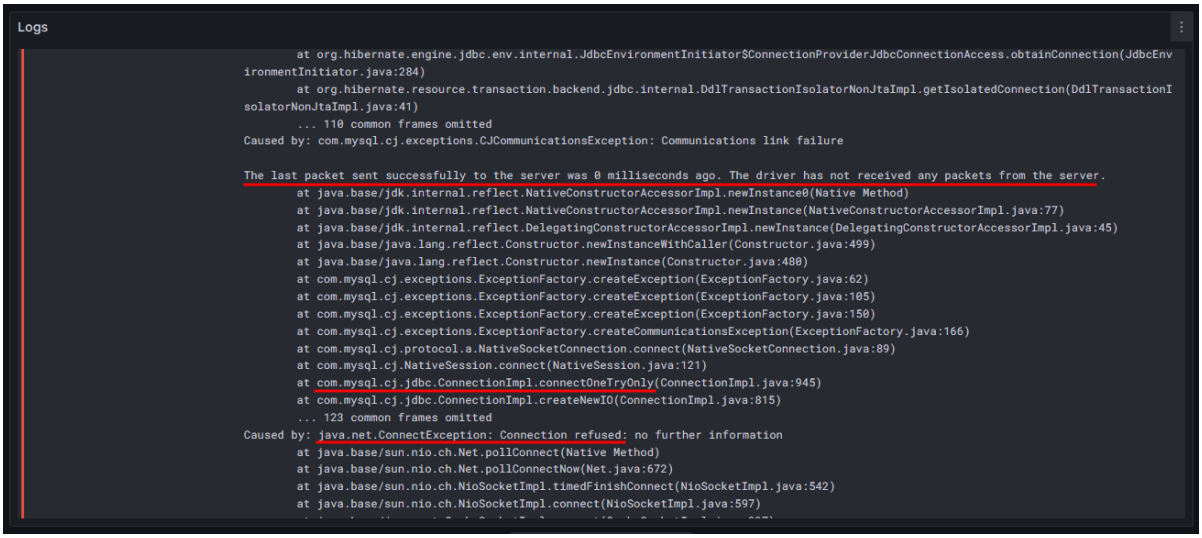


Autoria Própria

Na Figura 34 é evidenciado o rastreamento de pilha, exibindo um conjunto de chamadas dos métodos que geraram uma exceção para inicialização incorreta da ferramenta. Esse cenário é útil para os administradores de cada aplicação (por exemplo, sistema SHHC) para verificação inicial da atividade de observabilidade. A visualização desses registros requer um ajuste da *label* de busca para `ERROR`. Em cenário de computação mais complexos pode ser necessário a adição de mais filtros para facilitar a identificação dos registros.

No que diz respeito à rastreabilidade, uma situação envolvendo a interação entre a aplicação cliente e servidora será considerada. Inicialmente, a aplicação cliente realiza uma requisição `POST` para o servidor com o propósito de obter os dados de temperatura do paciente. Por meio do identificador de rastreio (por exemplo, `dc37ea1499b2def6c1ce9cdf9aa1756f` - sublinhado em amarelo) é possível identificar o fluxo de execução para efetuar a operação de

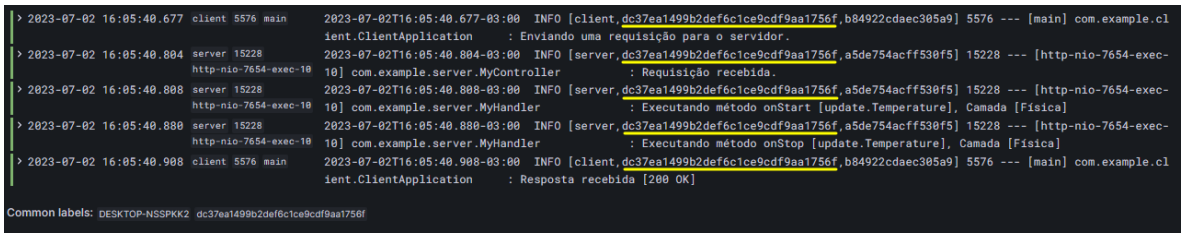
Figura 34 – Registro da exceção disparada



Autoria Própria

interesse em ambos os lados (cliente e servidor). Na Figura 35 é evidenciado os *logs* envolvendo a requisição realizada.

Figura 35 – Registros de *Logs* da requisição POST efetuada

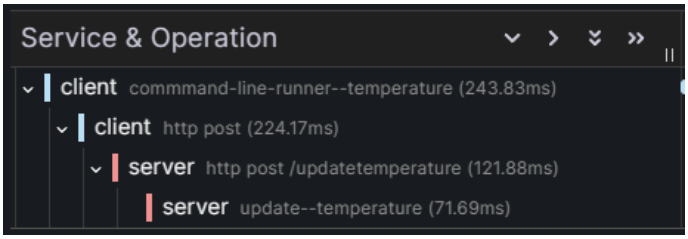


Autoria Própria

Para se obter informações de rastreio é necessário interagir com elemento denominado *Tempo* da ferramenta *Grafana* (veja Figura 31). Utilizando o mesmo identificador de rastreio (`dc37ea1499b2def6c1ce9cdf9aa1756f`) é possível identificar o fluxo de execução que foi efetuado durante este processo, como ilustrado na Figura 36. Além disso, esse elemento permite extrair/visualizar o tempo de resposta para uma chamada de serviço, como ilustrado na Figura 37. Conforme pode-se observar nessa figura, a requisição de temperatura de um paciente tem tempo de execução de `71.69ms` para executar completamente o serviço. Esse tipo de informação pode evidenciar aos interessados/administradores da aplicação diferentes linhas de comportamento dos serviços de um sistema. Evidências sobre algum tipo de anomalia que o sistema esteja sofrendo em um determinado momento, ou degradação de qualidade dos serviços fornecidos pela sobrecarga de execução são alguns dos exemplos dessas linhas de comportamentos.

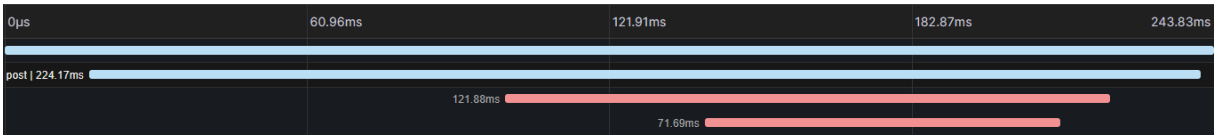
Além das informações exibidas nesta seção, a ferramenta (elemento `Tempo`) permite outras informações de interesse que não serão reportadas aqui por razões de espaço e escopo.

Figura 36 – Fluxo de execução.



Autoria Própria

Figura 37 – Tempo gasto para execução.



Autoria Própria

Nesse sentido, vale destacar que cabe aos interessados/administradores selecionar as informações a serem extraídas, pois estas têm relação direta com o nível de criticidade e domínio de cada aplicação (ou seja, Self-CPS).

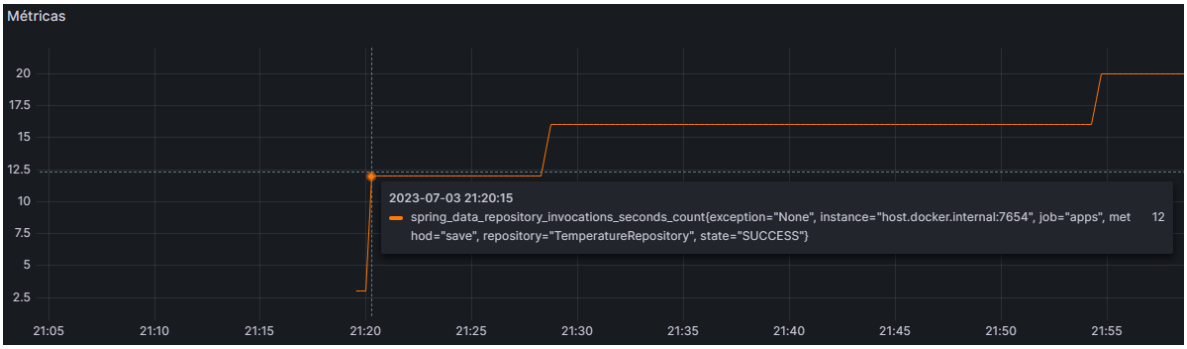
No que tange às métricas de uma aplicação, o objetivo é apresentar de forma concisa o estado atual da aplicação observada (por exemplo, sistema SHHC). Os dados relacionados às métricas enviadas pela aplicação são processados pelo elemento denominado `Prometheus` da ferramenta *Grafana* (veja Figura 31). Em resumo, esse elemento sintetiza a saída desse processamento por meio da disponibilização de gráficos interativos ao usuário. Dessa maneira, o especialista consegue obter ciência sobre a saúde dos componentes de maneira ágil e eficiente. Para isso, uma série de métricas são fornecidas pelo elemento `Prometheus`, a saber: (i) uso de CPU em porcentagem contabilizando todos os processos do sistema operacional; (ii) uso de CPU em porcentagem em relação a um processo do sistema; (iii) contagem de uma requisição HTTP; (iv) tempo de CPU gasto ao todo em uma requisição HTTP; (v) conexões ao banco de dados ativas; (vi) quantidade em *bytes* utilizada por um recurso; (vii) quantidade de memória máxima da aplicação, entre outras. A seguir, são apresentados gráficos sobre as métricas exploradas pelo sistema SHHC.

Na Figura 38 é ilustrada a contagem total de invocações para o serviço que lida com o sensor de temperatura do paciente. Para isso, a seguinte *query* foi utilizada:

```
spring_data_repository_invocations_seconds_count{repository=
                                        "TemperatureRepository"}
```

Nas Figuras 39 e 40 é ilustrada a carga de CPU em porcentagem. A primeira ilustra

Figura 38 – Métricas.



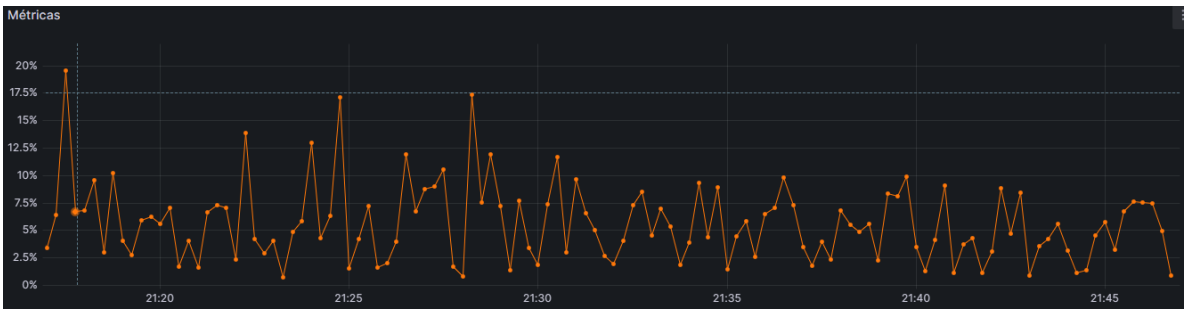
Autoria Própria

a carga de CPU, considerando todos os processos executados pelo sistema operacional. Já a segunda apresenta a carga de CPU utilizada somente pelo servidor de aplicação durante uma janela de tempo arbitrária. Para gerar as referidas figuras, as seguintes *querys* foram executadas:

(1). `system_cpu_usage` (Figura 39)

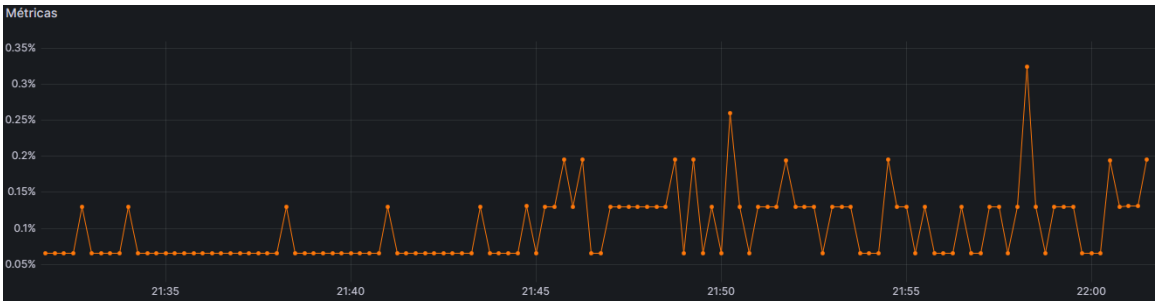
(2). `proccess_cpu_usage` (Figura 40)

Figura 39 – Total da carga de CPU utilizada pelo sistema operacional.



Autoria Própria

Figura 40 – Total da carga de CPU utilizada pelo servidor de aplicação.



Autoria Própria

Por fim, na Figura 41 é ilustrada a duração máxima da requisição responsável pela atualização de dados de temperatura, a qual é localizada pela interface de comunicação denominada

`updateTemperature` (ou seja, módulo `SHHCSensorAPI`). Cabe salientar que o valor máximo da requisição ilustrado no gráfico representa o limite de valor de uma janela de tempo e, após um determinado tempo, o valor máximo capturado é reinicializado. Para gerar a referida figura, a seguinte *query* foi utilizada:

```
http_server_requests_seconds_max {uri="/updateTemperature"}
```

Figura 41 – Tempo de processamento máximo da requisição.

Autoria Própria

Camada de monitoramento: módulo Autoproteção

Conforme reportado por Martins (2022), a abordagem de autoproteção proposta permite monitorar requisições do tipo HTTP em aplicações Web e APIs. Como o sistema SHHC, descrito na Seção 5.1, possui apenas o `SHHCSensorAPI` como módulo para comunicação via serviços Web via REST, apenas este módulo deve ser modificado para que a abordagem de autoproteção seja a ele acoplada. Para acoplar tal abordagem ao módulo `SHHCSensorAPI` é necessário incluir a dependência no arquivo `pom.xml`, como apresentado na Listagem 6.

Listagem 6 – Código fonte do arquivo `pom.xml`

```
1  <dependencies>
2      <!--
3      Trechos omitidos
4      -->
5      <dependency>
6          <groupId>br.unesp.rc</groupId>
7          <artifactId>LibSelfProtection</artifactId>
8          <version>1.0-SNAPSHOT</version>
9      </dependency>
10 </dependencies>
```

Fonte: Elaborado pelo autor

Em seguida, é necessário inserir as seguintes informações (veja Listagem 7) no arquivo `application.properties` para funcionamento da abordagem. Entre as Linhas 4 e 6 é habilitado o processo de segurança, definida a porta de execução e o *host*. Essas configurações são

utilizadas pela aplicação `AppMape` da abordagem de autoproteção. Por fim, na Linha 8 é definido o identificador para a aplicação (`SHHCSensorAPI`).

Listagem 7 – Código fonte do arquivo `application.properties`

```
1 server.port=8084
2 spring.data.mongodb.ignore-unknown-exceptions=true
3
4 mape.enable=true
5 mape.port=8081
6 mape.url=http://localhost
7
8 appmanaged.id=63152073fe5ad305865bc6c9
```

Fonte: Elaborado pelo autor

Uma vez definida a etapa de configuração, um filtro deve ser criado no projeto do referido módulo (`SHHCSensorAPI`) para que todos os *headers* e dados das requisições sejam capturados. Na Listagem 8 é apresentado o código fonte para o filtro das APIs existentes no módulo `SHHCSensorAPI`.

Listagem 8 – Código fonte da classe `CustomFilter.java`

```
1  package br.unesp.rc.shhc.SHHCSensorAPI.filter;
2
3  import br.unesp.rc.libselfprotection.monitoringpoints.MonitoringPointRequest;
4  import jakarta.servlet.Filter;
5  import jakarta.servlet.FilterChain;
6  import jakarta.servlet.ServletException;
7  import jakarta.servlet.ServletRequest;
8  import jakarta.servlet.ServletResponse;
9  import jakarta.servlet.http.HttpServletRequest;
10 import java.io.IOException;
11 import java.net.URISyntaxException;
12 import org.springframework.beans.factory.annotation.Value;
13 import org.springframework.context.annotation.Configuration;
14 import org.springframework.core.annotation.Order;
15 import org.springframework.web.util.ContentCachingRequestWrapper;
16
17 @Configuration
18 @Order(1)
19 public class CustomFilter implements Filter {
20
21     @Value("${appmanaged.id}")
22     private String appID;
23     @Value("${mape.enable}")
24     private boolean mapeEnable;
25     @Value("${mape.port}")
26     private String mapePort;
27     @Value("${mape.url}")
28     private String mapeURL;
```

```
29
30      @Override
31      public void doFilter(ServletRequest request, ServletResponse response,
            ↪ FilterChain chain) throws IOException, ServletException {
32
33          try {
34              HttpServletRequest currentRequest = (HttpServletRequest) request;
35              ContentCachingRequestWrapper wrappedRequest = new
                    ↪ ContentCachingRequestWrapper(currentRequest);
36              chain.doFilter(wrappedRequest, response);
37
38              if (mapeEnable){
39                  MonitoringPointRequest mpr =
                        ↪ MonitoringPointRequest.getInstance(mapeURL, mapePort);
40                  mpr.run(wrappedRequest, appID);
41              }else{
42              System.out.println("MonitoringPointRequest não enviado");
43          }
44          } catch (URISyntaxException ex) {
45              ex.printStackTrace();
46          } catch (Exception ex) {
47          ex.printStackTrace();
48          }
49      }
50
51      @Override
52      public void destroy() {
53      }
54  }
```

Fonte: Elaborado pelo autor

Na Linha 1 da Listagem 8 é possível observar o nome do pacote onde o filtro foi criado. Entre as Linhas 21 e 28 são definidos os atributos para esse filtro, que são as informações definidas no arquivo `application.properties`. Na Linha 38 é verificado se a segurança está habilitada na aplicação para que o ponto de monitoramento seja instanciado na Linha 39 e colocado em execução na Linha 40. Como pode-se observar, todas as requisições (`wrappedRequest`) de uma aplicação (`appID`) são monitoradas pela abordagem.

Camada de adaptação

Conforme reportado na descrição da RA4Self-CPS (veja Seção 4.3), a **camada de adaptação** deve ser projetada para atender os interesses de adaptação de um Self-CPS em tempo de execução. Em experiências anteriores de nosso grupo de pesquisa (PASSINI; AFFONSO, 2018a; PASSINI; AFFONSO, 2018b; PASSINI, 2020), um *framework* que permite lidar com a adaptação de serviços SOAP e REST foi desenvolvido. Em resumo, esse *framework* permite

que um serviço que apresente falha de execução ou algum tipo de degradação de qualidade seja substituído por um serviço similar em tempo de execução sem a percepção de seus usuários.

Partindo da definição da **camada de adaptação** e fazendo um paralelo (análise) com o sistema SHHC (especificamente no módulo `SHHCSensorAPI`), conclui-se que a organização desse sistema inviabiliza a utilização do referido *framework* para explorar cenários de adaptação de serviços em tempo de execução. Essa constatação foi obtida com base nas evidências de organização do referido módulo e na natureza do sistema SHHC. Como esse módulo (`SHHCSensorAPI`) representa um paciente com os seis sensores a ele acoplados, simular cenários de falhas seria considerar o paciente como um tudo e não os sensores de maneira individual. Assim, para simular cenários de falhas nos sensores é necessário refatorar esse módulo (`SHHCSensorAPI`) de modo que cada sensor passe a funcionar em um contêiner isolado. Além disso, mesmo que essa nova organização seja satisfeita, o sistema SHHC não apresenta potencial para explorar casos de degradação de qualidade, um dos requisitos para troca de serviços em tempo de execução. Haveria necessidade de projetar mais uma aplicação para simular a degradação de qualidade de serviço. Diante do exposto e com base nas evidências reportadas, essa camada não foi implementada para o sistema SHHC apresentado neste capítulo.

5.3 Considerações finais

Neste capítulo foi apresentado um estudo de caso, que teve como objetivo mostrar a aplicabilidade da RA4Self-CPS descrita no Capítulo 4 como uma arquitetura concreta para esse tipo de sistema. Para essa finalidade, um estudo de caso chamado *Smart Home Health Care* foi selecionado para explorar as características dos Self-CPS que estão sujeitos à organização proposta pela referida arquitetura. No Capítulo 6 são reportadas as principais contribuições deste mestrado acadêmico, assim como as principais perspectivas para trabalhos futuros.

6 CONCLUSÃO

Nesta dissertação foi abordado o projeto de uma arquitetura de referência para Self-CPS, a qual foi organizada em camadas de tal forma a integrar os principais conceitos desse tipo de sistema. A camada física é responsável por interagir com o ambiente físico através dos componentes sensores e atuadores. A camada *cyber* é encarregada de aplicar as regras de negócio que o domínio de aplicação necessita. Por fim, a camada adaptação contém um conjunto de mecanismos para que o sistema possa lidar com situações imprevistas em tempo de execução. Para o projeto dessa arquitetura, um arcabouço teórico foi reportado no Capítulo 2, detalhando os principais conceitos e fundamentos que compõem um sistema autoadaptativo, um sistema ciber-físico e a intersecção entre os mesmos. No Capítulo 3, com o intuito de apresentar o estado da arte sobre modelos e/ou arquiteturas de referência para Self-CPS, foi apresentada uma revisão sistemática da literatura onde foram sintetizados (i) os estudos secundários que nortearam a investigação acerca dos estudos que trabalharam com a intersecção dos temas deste mestrado acadêmico, e (ii) os estudos primários que permitiram o desenvolvimento da arquitetura de referência proposta neste trabalho. No Capítulo 4 foi apresentada a concepção da arquitetura, a qual foi denominada RA4-SelfCPS, com base no processo PROSA-RA combinada pela abordagem proposta por Tummers *et al.* (2021). Por fim, no Capítulo 5, um estudo de caso foi elaborado, tendo como objetivo explorar a aplicabilidade e viabilidade da referida arquitetura. Para finalizar essa dissertação, esse capítulo apresenta na Seção 6.1 as contribuições da dissertação, as dificuldades e limitações enfrentadas durante o desenvolvimento deste mestrado acadêmico são reportadas na Seção 6.2, e por fim, na Seção 6.3 são reportadas as perspectivas e trabalhos futuros a serem conduzidos.

6.1 Contribuições da dissertação

Conforme já exposto, este trabalho apresentou uma arquitetura de referência para apoiar o desenvolvimento de Self-CPS, abordando questões como a autoproteção, autoadaptação e observabilidade. Essas questões fornecem informações importantes para monitoramento de uma aplicação, permitindo identificar o nível de segurança, parâmetro de execução, rastreabilidade e registro. Além disso, a arquitetura proposta permite lidar com a adaptação de serviços em tempo de execução através da integração de um *framework* desenvolvido por Passini (2020).

Outra contribuição deste trabalho é o mapeamento sistemático da literatura apresentado no Capítulo 3. Para essa finalidade, o mapeamento foi baseado no protocolo formal de pesquisa (veja Apêndice A) para responder sete questões de pesquisa. A partir das informações coletadas,

foi possível estabelecer um panorama geral da área, identificando as lacunas, tendências, ambientes de aplicação, tecnologias utilizadas, entre outros tópicos. Dessa forma, espera-se fornecer à comunidade científica uma fonte de conhecimento acerca da intersecção destes sistemas.

6.2 Dificuldades e limitações

De forma geral, adversidades em vários níveis foram encontradas durante o desenvolvimento deste mestrado acadêmico. Dentre elas, pode-se citar a quantidade de conceitos relacionados ao objeto de investigação deste projeto (Self-CPS), além da complexidade dos ambientes de aplicação desse tipo de sistema. A seguir, detalhes dessas adversidades e limitações são apresentadas:

Quantidade elevada de conceitos. Partindo do pressuposto que uma arquitetura de referência pode ser guia para o desenvolvimento de sistemas (veja Seção 2.1) e que sua concepção deve conter as principais tendências e boas práticas de engenharia para o domínio de aplicação alvo, pode-se salientar que o desenvolvimento de uma arquitetura de referência é um processo lento e trabalhoso. Uma vez que a investigação das fontes de informações é uma etapa do PROSA-RA que antecede o desenvolvimento da arquitetura propriamente dita, um fator de dificuldade foi sintetizar o conhecimento adquirido destes sistemas, tendo em vista a riqueza e complexidades de informações evidenciadas através do mapeamento sistemático conduzido (veja Capítulo 3). Além disso, soma-se a característica de adaptação para que um Self-CPS possa ser projetado, o que acarreta a adição de novos conceitos e fundamentos. Por fim, foi constatado com a execução deste projeto de mestrado acadêmico a relação de dependência entre a adaptação e a organização do sistema. Diante do exposto, vale destacar que o desenvolvimento da camada de adaptação da arquitetura proposta neste projeto ficou restrita a um tipo de adaptação (ou seja, serviços Web) pela utilização de experiências anteriores de nosso grupo de pesquisa (AFFONSO; PASSINI; NAKAGAWA, 2019; PASSINI, 2020). Caso outro tipo/cenário de adaptação seja necessário, a incorporação de novos conceitos/fundamentos deve ser necessária, assim como adequação dos elementos da camada de adaptação para que novas modificações possam ser realizadas ao software (Self-CPS).

Complexidade dos ambientes de aplicação. Conforme mencionado no Seção 2.4, CPSs são sistemas dinâmicos e complexos e específicos em relação ao domínio (veja Capítulo 3). Assim, pode-se dizer que houve uma dificuldade em explorar os conceitos em sua totalidade que pertencem ao domínio (por exemplo, *e-Health*) durante a condução do estudo caso. Em termos práticos, isso ficou refletido através da diminuição do escopo do ambiente aplicado, uma vez que os dados sensoriados obtidos foram simulados, fatores como

idade, peso, altura não foram considerados durante a simulação destes dados e certos aspectos não foram contemplados, como as funcionalidades de adaptação. Além disso, a reprodução das possíveis falhas de execução a serem analisadas pela camada de adaptação seguiria abordagem de simulação, limitando a um baixo número de incertezas estudadas, tendo em vista a um elevado número de possibilidades a serem averiguadas.

Avaliação. Embora seja possível perceber o funcionamento e a viabilidade da RA4Self-CPS quando a arquitetura foi instanciada como uma arquitetura concreta para o sistema SHHC, não foi possível, por exemplo, conduzir uma avaliação direcionada a alguns aspectos como performance, adaptação e segurança. Além disso, pode-se dizer que somente uma aplicação, específica para um domínio, foi implementada/investigada. Essa limitação ocorreu especificamente em virtude das restrições de tempo inerentes em um projeto de mestrado acadêmico.

6.3 Trabalhos futuros

Considerando o escopo deste projeto de mestrado acadêmico, alguns aspectos foram categorizados como secundários ou até mesmo não foram contemplados neste trabalho. Nesse sentido, esses aspectos foram sintetizados em três atividades como trabalhos futuros, a saber:

1. **Instância da RA4Self-CPS em outros domínios.** Visando obter novos indicativos sobre a aplicabilidade e viabilidade da RA4Self-CPS, um estudo de caso para um sistema SHHC foi conduzido. Contudo, certos elementos tiveram seu escopo minimizado, a saber: (i) implementação dos sensores no ambiente de aplicação em modo simulado, (ii) erros/incertezas do ambiente físico tiveram representatividade reduzida e foram reproduzidos via simulação, e (iii) ausência da avaliação do módulo de autoproteção da arquitetura. Nesse sentido, a condução de novos estudos de caso envolvendo o mesmo domínio e/ou outros domínios deve ser conduzida como atividade futura. O objetivo dessa atividade é instanciar e avaliar a RA4Self-CPS em outros cenários de execução.

2. **Execução da arquitetura em um ambiente real.** Considerando a natureza dos (Self-)CPSs, uma outra atividade pretendida é a utilização da RA4Self-CPS no contexto fora da academia para que se possa avaliar seu comportamento, quando a arquitetura for instanciada em um ambiente real de execução.

3. **Explorar cenários de adaptação.** Conforme exposto neste trabalho, o *framework* desenvolvido por Passini (2020) necessita de um ambiente computacional maior para que serviços preferenciais e alternativos sejam desenvolvidos. Embora o *framework* tenha se mostrado efetivo em experiências anteriores de nosso grupo de pesquisa, a utilização do

mesmo em um Self-CPS não foi devidamente explorada neste trabalho. Assim, a condução de novos estudos de casos que possam explorar de maneira efetiva a adaptação de serviços é uma atividade a ser investigada no futuro para que se possa ter indicativos sobre o comportamento da RA4Self-CPS em ambientes computacionais mais complexos.

REFERÊNCIAS

ABDULMALEK, S. *et al.* Iot-based healthcare-monitoring system towards improving quality of life: A review. *Healthcare*, v. 10, n. 10, 2022. ISSN 2227-9032. Disponível em: <https://www.mdpi.com/2227-9032/10/10/1993>. Citado na página 18.

AFFONSO, F. J. *et al.* A framework based on learning techniques for decision-making in self-adaptive software. In: *The 27th International Conference on Software Engineering & Knowledge Engineering (SEKE' 15)*. Pittsburgh, USA: [s.n.], 2015. p. 24–29. ISSN 2325-9086. Citado na página 75.

AFFONSO, F. J.; NAKAGAWA, E. Y. A reference architecture based on reflection for self-adaptive software. In: *The 7th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS' 13)*. [S.l.: s.n.], 2013. p. 129–138. Citado 2 vezes na(s) página(s) 13 e 61.

AFFONSO, F. J.; PASSINI, W. F.; NAKAGAWA, E. Y. A reference architecture to support the development of mobile applications based on self-adaptive services. *Pervasive and Mobile Computing*, v. 53, p. 33 – 48, 2019. ISSN 1574-1192. Citado 3 vezes na(s) página(s) 13, 61 e 99.

ALMUTAIRI, A.; ALHARBI, M. A survey of adaptation and the best approach for ubiquitous systems. *International Journal of Computing and Digital Systems*, v. 6, p. 277–284, 09 2017. Citado na página 22.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. Pearson Education, 2012. (SEI Series in Software Engineering). ISBN 9780321815736. Disponível em: <https://books.google.com.br/books?id=jpUHuAAACAAJ>. Citado 2 vezes na(s) página(s) 13 e 16.

BELLMAN, K. *et al.* Self-aware cyber-physical systems. *ACM Trans. Cyber-Phys. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 4, n. 4, jun. 2020. ISSN 2378-962X. Citado 4 vezes na(s) página(s) 22, 26, 37 e 38.

BENNACEUR, A. *et al.* Modelling and analysing resilient cyber-physical systems. In: *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. [S.l.]: IEEE Press, 2019. (SEAMS '19), p. 70–76. Citado na página 36.

BRINGS, J.; SALMON, A.; SARITAS, S. Context uncertainty in requirements engineering: Definition of a search strategy for a systematic review and preliminary results. In: A., H. *et al.* (Ed.). [S.l.]: CEUR-WS, 2015. v. 1342, p. 171–178. ISSN 16130073. Cited By 2. Citado na página 35.

BRUN, Y. *et al.* Engineering self-adaptive systems through feedback loops. In: ____. *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 48–70. ISBN 978-3-642-02161-9. Disponível em: <https://doi.org/10.1007/978-3-642-02161-9_3>. Citado na página 18.

CASARIN, J. *UMA ARQUITETURA PERSUASIVA BASEADA EM SISTEMAS CIBER-FÍISICOS* . Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO RIO GRANDE, Brasil, 2016. Citado na página 25.

CHENG, B. H. C. *et al.* Software engineering for self-adaptive systems: A research roadmap. In: ____. *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 1–26. ISBN 978-3-642-02161-9. Disponível em: <https://doi.org/10.1007/978-3-642-02161-9_1>. Citado 3 vezes na(s) página(s) 12, 18 e 63.

DIESTE, O.; GRIMáN, A.; JURISTO, N. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 14, n. 5, p. 513–539, 2009. ISSN 1382-3256. Citado na página 112.

DIESTE, O.; PADUA, A. G. Developing search strategies for detecting relevant experiments for systematic reviews. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. [S.l.: s.n.], 2007. p. 215–224. Citado na página 30.

DYBA, T.; DINGSOYR, T.; HANSSEN, G. K. Applying systematic reviews to diverse study types: An experience report. In: *Proceedings of the 1st Empirical Software Engineering and Measurement (ESEM'07)*. Madri, Spain: IEEE Computer Society, 2007. p. 225–234. ISSN 0-7695-2886-4. Citado 2 vezes na(s) página(s) 30 e 40.

EELES, P.; CRIPPS, P. *The Process of Software Architecting*. Addison-Wesley, 2010. ISBN 9780321357489. Disponível em: <https://books.google.com.br/books?id=Cq-TmAEACAAJ>. Citado na página 15.

ERICSSON. *Streaming video – from megabits to gigabytes*. 2018. <https://www.ericsson.com/4a30ac/assets/local/mobility-report/documents/2018/emr-nov-2018-streaming-video.pdf>. Online; acessado 1 de Agosto de 2021. Citado na página 11.

ERICSSON. *Gaming on the move*. 2020. <https://www.ericsson.com/4a5888/assets/local/mobility-report/documents/2020/gaming-on-the-move-article.pdf>. Online; acessado 1 de Agosto de 2021. Citado na página 11.

FONSECA, M. *et al.* E-health practices and technologies: A systematic review from 2014 to 2019. *Healthcare*, v. 9, p. 1192, 09 2021. Citado na página 17.

GEROSTATHOPOULOS, I. *et al.* Self-adaptation in software-intensive cyber–physical systems: From system goals to architecture configurations. *Journal of Systems and Software*, v. 122, p. 378–397, 2016. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121216000601>. Citado 2 vezes na(s) página(s) 12 e 68.

GREER, C. *et al.* Cyber-physical systems and internet of things. *NIST Pubs*, 07 2019. Citado na página 63.

GROWIN, J.; WEYNS, D.; HOLVOET, T. Design patterns for multi-agent systems: A systematic literature review. *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, v. 9783642544323, p. 79–99, 02 2014. Citado na página 23.

GRUA, E.; SANCTIS, M. D.; LAGO, P. A reference architecture for personalized and self-adaptive e-health apps. In: ____. [S.l.: s.n.], 2020. p. 195–209. ISBN 978-3-030-59154-0. Citado 2 vezes na(s) página(s) 12 e 17.

GUO, S.; ZENG, D. *Cyber-physical Systems: Architecture, Security and Application*. Springer, 2019. (EAI/Springer innovations in communication and computing). ISBN 9783319925653. Disponível em: <https://books.google.com.br/books?id=o49MzQEACAAJ>. Citado na página 28.

HINCHEY, M.; STERRITT, R. Self-managing software. *Computer*, IEEE, v. 39, n. 2, p. 107–109, feb 2006. ISSN 0018-9162. Citado na página 20.

HOSSAIN, M. J.; BARI, M. A.; KHAN, M. M. Development of an iot based health monitoring system for e-health. In: *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.: s.n.], 2022. p. 0031–0037. Citado na página 17.

IBM. *An architectural blueprint for autonomic computing*. 2005. [On-line]. Available: <https://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf>, Third Edition, Accessed on 6 de outubro de 2023. Citado 3 vezes na(s) página(s) 23, 74 e 75.

IGNATIUS, H. T.; BAHSOON, R. A conceptual reference model for human as a service provider in cyber physical systems. In: *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. [s.n.], 2021. p. 1–10. ISSN 2157-2321. Disponível em: <10.1109/SEAMS51251.2021.00012>. Citado na página 63.

ISLAM, S. M. R. *et al.* The internet of things for health care: A comprehensive survey. *IEEE Access*, v. 3, p. 678–708, 2015. Citado na página 17.

KAGERMANN, H. *et al. Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry ; Final Report of the Industrie 4.0 Working Group*. Forschungsunion, 2013. Disponível em: <https://books.google.com.br/books?id=AsfOoAEACAAJ>. Citado na página 26.

KANG, H. *et al.* Smart manufacturing: Past research, present findings, and future directions. *International Journal of Precision Engineering and Manufacturing-Green Technology*, v. 3, p. 111–128, 01 2016. Citado na página 11.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, v. 36, n. 1, p. 41–50, 2003. Citado 2 vezes na(s) página(s) 18 e 20.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Citado 6 vezes na(s) página(s) 13, 30, 31, 40, 112 e 114.

KITCHENHAM, B. *et al.* Systematic literature reviews in software engineering – a tertiary study. *Information and Software Technology*, v. 52, n. 8, p. 792–805, 2010. ISSN 0950-5849. Citado 2 vezes na(s) página(s) 30 e 114.

KITCHENHAM, B. A.; DYBA, T.; JORGENSEN, M. Evidence-based software engineering. In: *Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004. (ICSE '04), p. 273–281. ISBN 0-7695-2163-0. Citado 3 vezes na(s) página(s) 13, 30 e 31.

KLUGE, T. A role-based architecture for self-adaptive cyber-physical systems. In: *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. New York, NY, USA: Association for Computing Machinery, 2020. (SEAMS '20), p. 120–124. ISBN 9781450379625. Disponível em: <https://doi.org/10.1145/3387939.3391601>. Citado na página 12.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2004. (Addison-Wesley object technology series). ISBN 9780321197702. Disponível em: <https://books.google.com.br/books?id=RYCMx6o47pMC>. Citado na página 16.

KRUPITZER, C. *et al.* A survey on engineering approaches for self-adaptive systems. *Pervasive Mob. Comput.*, Elsevier Science Publishers B. V., NLD, v. 17, n. PB, p. 184–206, fev. 2015. ISSN 1574-1192. Disponível em: <https://doi.org/10.1016/j.pmcj.2014.09.009>. Citado 2 vezes na(s) página(s) 22 e 23.

LEE, E. Cyber-physical systems - are computing foundations adequate? 01 2006. Citado na página 25.

LEITÃO, P.; COLOMBO, A. W.; KARNOUSKOS, S. Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Computers in Industry*, v. 81, p. 11–25, 2016. ISSN 0166-3615. Emerging ICT concepts for smart, safe and sustainable industrial systems. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0166361515300348>. Citado 2 vezes na(s) página(s) 26 e 63.

LI, J.-Q. *et al.* Industrial internet: A survey on the enabling technologies, applications, and challenges. *IEEE Communications Surveys Tutorials*, v. 19, n. 3, p. 1504–1526, 2017. Citado na página 11.

LI, T. *et al.* From offline toward real-time: A hybrid systems model checking and cps co-design approach for medical device plug-and-play (mdpnp). In: *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*. [S.l.: s.n.], 2012. p. 13–22. Citado na página 25.

MARTINS, A.; MARTINS, A. F. Sistema físico cibernético multiagente para monitoramento remoto de pacientes. In: . [S.l.: s.n.], 2015. Citado na página 25.

MARTINS, R. R. *Autoproteção para a camada de aplicação: uma abordagem baseada em técnicas de aprendizado e no laço de controle MAPE-K*. Dissertação (Dissertação de Mestrado) — Universidade Estadual Paulista (UNESP), Instituto de Geociências e Ciências Exatas (IGCE), Rio Claro, 2022. Programa de Pós-Graduação em Ciência da Computação (PPGCC). Citado 3 vezes na(s) página(s) 14, 73 e 94.

MARTINS, R. R. *et al.* A self-protecting approach for service-oriented mobile applications. In: FILIPE, J. *et al.* (Ed.). *Proceedings of the 23rd International Conference on Enterprise Information Systems, ICEIS 2021, Online Streaming, April 26-28, 2021, Volume 2*. SCITEPRESS, 2021. p. 313–320. Disponível em: <https://doi.org/10.5220/0010448603130320>. Citado na página 73.

MARWEDEL, P. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. [S.l.: s.n.], 2010. ISBN 9789400702561. Citado na página 25.

MATHUR, S.; ARORA, A. Internet of things (iot) and pki-based security architecture. In: ____. [S.l.: s.n.], 2020. p. 25–46. ISBN 9781799828051. Citado na página 68.

MESLI-KESRAOUI, S. *et al.* Formal verification of software-intensive systems architectures described with piping and instrumentation diagrams. In: TEKINERDOGAN, B.; ZDUN, U.; BABAR, A. (Ed.). *Software Architecture*. Cham: Springer International Publishing, 2016. p. 210–226. ISBN 978-3-319-48992-6. Citado na página 15.

MUCCINI, H.; SHARAF, M.; WEYNS, D. Self-adaptation for cyber-physical systems: A systematic literature review. In: *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. [S.l.: s.n.], 2016. p. 75–81. Citado 5 vezes na(s) página(s) 12, 26, 28, 37 e 38.

MUELLER, I.; BRAUN, P.; KOWALCZYK, R. Design patterns for agent-based service composition in the web. In: . [S.l.: s.n.], 2005. v. 2005, p. 425–430. ISBN 0-7695-2472-9. Citado na página 25.

MULLER, G. A reference architecture primer. In: . [S.l.: s.n.], 2008. Citado na página 16.

MUSIL, A. *et al. Patterns for self-adaptation in Cyber-Physical Systems*. [S.l.]: Springer International Publishing, 2017. 331–368 p. Cited By 17. ISBN 9783319563459; 9783319563442. Citado 5 vezes na(s) página(s) 12, 19, 26, 28 e 36.

MUSIL, J.; MUSIL, A.; BIFFL, S. Introduction and challenges of environment architectures for collective intelligence systems. In: *Revised Selected and Invited Papers of the 4th International Workshop on Agent Environments for Multi-Agent Systems IV - Volume 9068*. Berlin, Heidelberg: Springer-Verlag, 2014. p. 76–94. ISBN 9783319238494. Disponível em: <https://doi.org/10.1007/978-3-319-23850-0_6>. Citado na página 36.

NAKAGAWA, E.; OQUENDO, F.; BECKER, M. Ramodel: A reference model for reference architectures. In: . [S.l.: s.n.], 2012. p. 297–301. ISBN 978-1-4673-2809-8. Citado 2 vezes na(s) página(s) 13 e 16.

NAKAGAWA, E. Y. *et al.* Consolidating a process for the design, representation, and evaluation of reference architectures. In: *2014 IEEE/IFIP Conference on Software Architecture*. [S.l.: s.n.], 2014. p. 143–152. Citado na página 16.

NORD, J. H.; KOOHANG, A.; PALISZKIEWICZ, J. The internet of things: Review and theoretical framework. *Expert Systems with Applications*, v. 133, p. 97–108, 2019. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417419303331>. Citado na página 11.

OREIZY, P. *et al.* An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and their Applications*, v. 14, n. 3, p. 54–62, 1999. Citado na página 18.

PARASHAR, M.; HARIRI, S. Autonomic computing: An overview. In: . [S.l.: s.n.], 2004. v. 3566, p. 257–269. Citado na página 20.

PASSINI, W. F. *Desenvolvimento de serviços compostos autoadaptativos: um framework baseado em implantação dinâmica, métricas de QoS e informação semântica*. Dissertação (Mestrado) — Universidade Estadual Paulista (UNESP), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto, Programa de Pós-Graduação em Ciência da Computação (PPGCC), 2020. Citado 7 vezes na(s) página(s) 14, 61, 72, 96, 98, 99 e 100.

PASSINI, W. F.; AFFONSO, F. J. Developing self-adaptive service-oriented mobile applications: A framework based on dynamic deployment. *International Journal of Software Engineering and Knowledge Engineering*, v. 28, n. 11n12, p. 1537–1558, 2018. Disponível em: <https://doi.org/10.1142/S0218194018400168>. Citado 2 vezes na(s) página(s) 72 e 96.

PASSINI, W. F.; AFFONSO, F. J. A framework to support the development of self-adaptive service-oriented mobile applications. In: *The 30th International Conference on Software Engineering and Knowledge Engineering. (SEKE' 2018)*. Redwood City, San Francisco Bay, California, USA: Knowledge Systems Institute Graduate School, 2018. p. 1–6. ISSN 2325-9086. Citado 2 vezes na(s) página(s) 72 e 96.

PASSINI, W. F. *et al.* Design of frameworks for self-adaptive service-oriented applications: A systematic analysis. *Software: Practice and Experience*, n/a, n. n/a, p. 1–34, 2021. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3013>. Citado na página 72.

PETERSEN, K. *et al.* Systematic mapping studies in software engineering. In: *The 12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, UK: BCS Learning & Development Ltd., 2008. p. 68–77. Citado na página 109.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, v. 64, p. 1–18, 2015. ISSN 0950-5849. Citado 5 vezes na(s) página(s) 30, 40, 109, 112 e 114.

PISCHING, M. A. *Arquitetura para descoberta de equipamentos em processos de manufatura com foco na Indústria 4.0* . Dissertação (Mestrado) — Universidade de São Paulo, Brasil, 2008. Citado na página 26.

RAJKUMAR, R. R. *et al.* Cyber-physical systems: The next computing revolution. In: *Proceedings of the 47th Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2010. p. 731–736. ISBN 9781450300025. Disponível em: <https://doi.org/10.1145/1837274.1837461>. Citado na página 25.

RATASICH, D. *et al.* A self-healing framework for building resilient cyber-physical systems. In: *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*. [S.l.: s.n.], 2017. p. 133–140. Citado na página 12.

RAWAT, D. B.; RODRIGUES, J. J. P. C.; STOJMENOVIC, I. *Cyber-Physical Systems: From Theory to Practice*. USA: CRC Press, Inc., 2015. ISBN 1482263327. Citado 2 vezes na(s) página(s) 26 e 28.

ROBERTSON, P.; SHROBE, H.; LADDAGA, R. *Self-Adaptive Software: First International Workshop, IWSAS 2000 Oxford, UK, April 17-19, 2000 Revised Papers*. Springer Berlin Heidelberg, 2003. (Lecture Notes in Computer Science). ISBN 9783540445845. Disponível em: <https://books.google.com.br/books?id=QfxsCQAAQBAJ>. Citado na página 18.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. USA: Prentice Hall Press, 2009. ISBN 0136042597. Citado na página 22.

SAHI, M. A. *et al.* Privacy preservation in e-healthcare environments: State of the art and future directions. *IEEE Access*, v. 6, p. 464–478, 2018. Citado na página 17.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, may 2009. ISSN 1556-4665. Disponível em: <http://doi.acm.org/10.1145/1516533.1516538>. Citado 6 vezes na(s) página(s) 12, 19, 20, 21, 23 e 63.

SÁNCHEZ, B. B. *et al.* Process execution in cyber-physical systems using cloud and cyber-physical internet services. *The Journal of Supercomputing*, 05 2018. Citado na página 25.

SILVA, L. C. e. *Uma Abordagem Baseada em Modelos para Suporte à Validação de Sistemas Médicos Físico-Cibernéticos*. Tese (Doutorado) — Universidade Federal de Campina Grande, Brasil, 2015. Citado na página 25.

SINGH, P. Internet of things based health monitoring system : Opportunities and challenges. *International Journal of Advanced Computer Research*, Volume 9 , Jan- Feb 2018, 02 2018. Citado na página 17.

SUN, Y.; YANG, G.; ZHOU, X.-s. A survey on run-time supporting platforms for cyber physical systems. *Frontiers of Information Technology & Electronic Engineering*, v. 18, n. 10, p. 1458, 2017. ISSN 2095-9230. Citado na página 35.

TOZZI, C. *The 3 pillars of observability: Logs, metrics and traces*. 2023. Available: <https://www.techtarget.com/searchitoperations/tip/The-3-pillars-of-observability-Logs-metrics-and-traces>, Accessed on 6 de outubro de 2023. Citado 2 vezes na(s) página(s) 23 e 74.

TUMMERS, J. *et al.* Designing a reference architecture for health information systems. *BMC Medical Informatics and Decision Making*, v. 21, n. 1, p. 210, Jul 2021. ISSN 1472-6947. Disponível em: <https://doi.org/10.1186/s12911-021-01570-2>. Citado 5 vezes na(s) página(s) 16, 61, 64, 76 e 98.

VILLA, J. *Introducing exemplar support in Grafana Cloud, tightly coupling traces to your metrics*. 2022. <https://grafana.com/blog/2022/02/23/introducing-exemplar-support-in-grafana-cloud-tightly-coupling-traces-to-your-metrics/>. Online; acessado 09 de Agosto de 2023. Citado na página 89.

WANG, J. *et al.* A secured health care application architecture for cyber-physical systems. *Control Engineering and Applied Informatics*, v. 13, 12 2011. Citado na página 27.

WEYNS, D. Software engineering of self-adaptive systems: An organised tour and future challenges. In: . [S.l.]: Springer, 2017. Citado 3 vezes na(s) página(s) 18, 23 e 63.

WEYNS, D. *et al.* On patterns for decentralized control in self-adaptive systems. In: *Software Engineering for Self-Adaptive Systems*. [S.l.: s.n.], 2010. Citado na página 24.

ZEADALLY, S.; SANISLAV, T.; MOIS, G. Self-adaptation techniques in cyber-physical systems (cpss). *IEEE Access*, PP, p. 1–1, 11 2019. Citado na página 11.

ZHANG, Y. *et al.* Health-cps: Healthcare cyber-physical system assisted by cloud and big data. *IEEE Systems Journal*, v. 11, n. 1, p. 88–95, 2017. Citado 2 vezes na(s) página(s) 16 e 17.

ZHOU, P. *et al.* A comprehensive technological survey on the dependable self-management cps: From self-adaptive architecture to self-management strategies. *Sensors (Switzerland)*, MDPI AG, v. 19, n. 5, 2019. ISSN 14248220. Cited By 3. Citado 3 vezes na(s) página(s) 12, 33 e 38.

APÊNDICE A – PROTOCOLO DE PESQUISA

A.1 Introdução

Este estudo visa estabelecer uma visão abrangente sobre o projeto de modelos e/ou arquiteturas de referência para apoiar o desenvolvimento de Self-CPS. Como essa visão pode resultar em modelos e/ou arquiteturas voltados para diferentes domínios de sistemas, uma classificação para obtenção de estudos voltados para o domínio da saúde (por exemplo, *e-Helath*, *Heathcare*, entre outras derivações) deve ser realizada. Para isso, foi adotada a técnica de mapeamento sistemático (do inglês, *Systematic Mapping Study – SMS*) baseada no processo proposto por (PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA; KUZNIARZ, 2015), o qual é composto por cinco passos principais:

- **[Passo 1]** ***Definição das Questões de Pesquisa.*** Neste passo são definidos os objetivos da pesquisa e o protocolo de mapeamento sistemático, que representa um plano pré-determinado que descreve questões de pesquisa, assim como o mapeamento sistemático deve ser conduzido;
- **[Passo 2]** ***Condução da Pesquisa.*** Neste passo os estudos primários são recuperados e identificados;
- **[Passo 3]** Triagem dos Artigos. Neste passo os estudos identificados são avaliados de acordo com os critérios de inclusão e exclusão definidos no protocolo de pesquisa;
- **[Passo 4]** ***Elaboração dos resumos.*** Neste passo os estudos incluídos são analisados e classificados; e
- **[Passo 5]** ***Extração de dados e mapeamento dos estudos.*** Neste passo ocorre a construção da representação final do mapeamento, que deve ser realizada com base no esquema final criado no Passo 4.

Como forma de apresentar os passos supracitados, este documento foi organizado em outras cinco seções. Na Seção A.2 é apresentada as questões de pesquisa proposta para este SMS. Já na Seção A.3 é reportada a estratégia de pesquisa utilizada neste SMS, sendo destacada a *string* de busca, as bases de pesquisa utilizadas e os critérios de inclusão e exclusão utilizados para selecionar os estudos. Em seguida, na Seção A.4 é mostrado como os dados foram coletados

e sintetizados. Na Seção A.5 são reportadas as limitações deste tipo de trabalho. Por fim, na Seção A.6 é apresentado um breve relato sobre como os resultados devem ser reportados.

A.2 Questões de pesquisa

O objetivo principal deste SMS é identificar estudos primários voltados para o projeto de modelos e/ou arquiteturas de referência para apoiar o desenvolvimento de Self-CPS. As Questões de Pesquisa (QP) estabelecidas são:

- **QP 1:** Quais são os modelos de referência e/ou arquiteturas de referência voltados ao desenvolvimento de Self-CPS?
  O objetivo dessa questão é identificar os modelos de referência e/ou arquiteturas existentes na literatura. Como o resultado dessa questão pode apresentar modelos e/ou arquiteturas para diferentes domínios de sistemas de software, uma classificação dos estudos resultantes deve ser realizada para seleção somente de trabalhos voltados para o domínio de saúde.

- **QP 2:** Quais domínios de aplicação foram beneficiados por tais modelos/arquiteturas para Self-CPS?
  O objetivo desta questão é mapear os domínios de aplicação que têm utilizado modelos e/ou arquiteturas de referência no projeto de sistemas. Como exemplo, pode-se citar o domínio de sistemas críticos voltado para saúde em ambientes domésticos ou hospitalares.

- **QP 3:** Quais são os atributos de qualidade adotados no projeto de tais modelos e/ou arquiteturas?
  O objetivo dessa questão é elencar os atributos de qualidade utilizados no projeto desses modelos e/ou arquiteturas. Exemplos: confiabilidade, disponibilidade, tempo de resposta, etc. Além disso, essa questão pode fornecer evidências de como tais atributos têm sido utilizados em tais sistemas. Por exemplo: níveis de QoS entre ambiente físico e sistema de software.

- **QP 4:** Como esses modelos e/ou arquiteturas têm sido avaliados(as)?
  O objetivo dessa questão é entender como esses modelos e/ou arquiteturas têm sido avaliados. Por exemplo: prova de conceito, estudo de caso, experimentos, casos reais em ambientes industriais.

- **QP 5:** Quais são as evidências que motivam a adoção desses modelos e/ou arquiteturas pelos interessados?
  O objetivo dessa questão é reunir evidências que possam nortear a adoção de modelos

e/ou arquiteturas. Por exemplo: nível de maturidade, aplicabilidade na indústria, entre outros.

- **QP 6:** Como esses modelos e/ou arquiteturas foram projetados(as)?
  O objetivo dessa questão é entender como tais modelos e/ou arquitetura têm sido projetados. Como essa questão está associada a vários outros aspectos, a mesma foi organizada em quatro subquestões:

  - **QP 6.1:** Como esses modelos e/ou arquiteturas têm sido organizados?
    O objetivo dessa questão é entender como tais modelos e/ou arquiteturas foram organizadas. Em outras palavras, quais estilos arquiteturais têm sido empregados. Exemplo: arquitetura em camadas, arquitetura em módulos, arquitetura em componentes, entre outros.
  - **QP 6.2:** Como os modelos e/ou arquiteturas lidam como o monitoramento do ambiente de execução?
    O objetivo dessa questão é entender como a atividade de monitoramento do ambiente de execução tem lido organizada/projetada em tais modelos e/ou arquiteturas. Exemplo: monitoramento via sensores, otimização do monitoramento (detecção de falhas e/ou degradação de QoS), entre outros.
  - **QP 6.3:** Como a mudança de requisitos e/ou de contexto em tempo de execução tem sido tratada em tais modelos/arquiteturas?
    O objetivo dessa questão é entender como a mudança de requisitos (funcionais e não funcionais) tem sido tratada em tais modelos e/ou arquiteturas. No que se refere aos requisitos funcionais espera-se mapear como essas mudanças são incorporadas ao software em tempo de execução. Em relação aos requisitos não funcionais, espera-se mapear como as mudanças de contexto podem afetar a integridade/funcionamento do sistema.
  - **QP 6.4:** Quais recursos têm sido empregados em tais modelos e/ou arquiteturas?
    O objetivo dessa questão é elencar os principais recursos tecnológicos que se fazem presente no projeto desses modelos e/ou arquiteturas. Tais recursos podem viabilizar a comunicação entre os componentes de um Self-CPS, assim como auxiliar nas abordagens de adaptações. Exemplos: Cloud Computing, Internet of Things, Machine Learning.

- **QP 7:** Como os interesses de adaptação em tempo de execução têm sido abordados em tais modelos e/ou arquiteturas?
  O objetivo dessa questão é entender como tais modelos e/ou arquitetura foram projetados em relação aos interesses de adaptação, ou seja, como a lógica de adaptação foi orga-

nizada nesse tipo de aplicação (Self-CPS). Como essa questão está associada a vários outros aspectos, a mesma foi organizada em duas subquestões:

- **QP 7.1:** Quais tipos de incertezas têm sido tratadas nesses modelos e/ou arquiteturas?
  O objetivo é entender como tais modelos e/ou arquiteturas têm lidado com as situações de incertezas para preservar seu estado de execução. Exemplos: condições dinâmicas, imprevisibilidade de requisitos, precisão dos dados coletados do ambiente, disponibilidade de recursos, entre outros.
- **QP 7.2:** Quais técnicas de inteligência artificial têm sido empregadas no projeto desses modelos e/ou arquiteturas para lidar com os interesses de autoadaptação?
  O objetivo dessa questão é estudar e elencar quais são as abordagens de adaptações que vem sido integradas no projeto desses modelos/arquiteturas. Indiretamente, essa questão pode fornecer evidências sobre a organização da lógica de adaptação desses sistemas (Self-CPS). Exemplos: multi-agent system, smart elements, loop MAPE-K, autonomous entities, técnicas de swarm, propriedades self-.*

A.3 Estratégia de pesquisa

A estratégia de pesquisa foi definida com base nas QPs apresentadas na Seção A.2. Essa estratégia é composta por critério de seleção das bases de pesquisa, lista de bases pesquisadas, idioma do estudo e, palavras-chave e seus termos relacionados. A seguir, detalhes dessa estratégia são apresentados:

1. **Critério de seleção.** Os seguintes critérios foram utilizados para selecionar as bases de pesquisa deste mapeamento (DIESTE; GRIMáN; JURISTO, 2009): (i) frequência de atualização de conteúdo; (ii) disponibilidade de textos na íntegra; (iii) qualidade dos resultados apresentados pela base; e (iv) versatilidade para exportar os dados obtidos;

2. **Lista de bases pesquisadas.** As bases de pesquisa selecionadas para o mapeamento são apresentadas na Tabela 5. Segundo Kitchenham e Charters (2007) e Petersen, Vakkalanka e Kuzniarz (2015), tais bases atendem aos critérios de seleção de bases de pesquisa supracitados e são consideradas suficientes para a condução desse tipo de estudo para a área de **engenharia de software**.

3. **Idioma dos estudos.** Apenas estudos primários escritos em inglês devem ser considerados neste SMS. Essa restrição deve-se ao fato desse idioma ser o mais comum em publicações científicas, além de auxiliar em futuras replicações deste trabalho.

Tabela 5 – Lista de bases de pesquisa

Base de pesquisa	**Endereço**
ACM Digital Library	<dl.acm.org>
IEEE Xplore	<www.ieeexplore.ieee.org>
ScienceDirect	<www.sciencedirect.com>
Scopus	<www.scopus.com>
Web of Science	<www.isiknowledge.com>

4. **Palavras-chave:** *reference architecture*, *cyber-physical systems*, *self-adaptive systems*, com os seguintes sinônimos:

   a) ***reference architecture:*** *reference architecture, reference architectures, reference model, reference models*;

   b) ***cyber-physical systems:*** *CPS, cyber physical, cyber-physical, cyberphysical, embedded*;

   c) ***self-adaptive systems:*** *autonomic, autonomous, dynamic, self-adaptable, self-adaptation, self-adapting, self-adaptive*;

5. ***String* de pesquisa:** O operador booleano `OR` foi utilizado para relacionar os termos principais e seus respectivos sinônimos. Todos esses termos foram combinados utilizando o operador booleano `AND`. Portanto, a *string* de busca final é apresentada na Tabela 6.

Tabela 6 – *String* de pesquisa

```
(reference architecture OR reference architectures OR reference
model OR reference models)
AND
(CPS OR cyber physical OR cyber-physical OR cyberphysical OR
embedded )
AND
(autonomic OR autonomous OR dynamic OR self-adaptable OR
self-adaptation OR self-adapting OR self-adaptive)
```

Definir os critérios de inclusão (do inglês, *Inclusion Criteria* – IC) e critérios de exclusão (do inglês, *Exclusion Criteria* – EC) para um mapeamento é um importante elemento para condução e execução do mesmo. A partir da definição desses critérios é possível incluir estudos primários relevantes para responder às QPs e excluir estudos que não permitem extrair respostas para as mesmas. Para este SMS, o seguinte IC foi definido:

- **IC 1:** O estudo primário apresenta um modelo e/ou arquitetura de referência para apoiar o desenvolvimento de Self-CPS.

Em contrapartida, os ECs definidos para este mapeamento foram:

- **EC 1:** O estudo não evidencia modelo ou arquitetura de referência que possa apoiar o desenvolvimento de Self-CPS;
- **EC 2:** O estudo é um editorial, artigo de opinião, palestra, opinião, tutorial, pôster ou painel;
- **EC 3:** O estudo foi escrito em um idioma diferente do definido pela pesquisa;
- **EC 4:** O estudo possui apenas um resumo ou não está disponível para ser lido na íntegra;
- **EC 5:** O estudo é um trabalho desenvolvido anteriormente pelo mesmo autor. Nesse caso, apenas a versão mais recente do estudo ou a mais completa será considerada.
- **EC 6:** O estudo é um trabalho secundário.

Por fim, vale destacar que nenhum critério de qualidade será aplicado ao mapeamento devido ao número de pesquisadores envolvidos (KITCHENHAM; CHARTERS, 2007; KITCHENHAM *et al.*, 2010). Para isso, a *string* de busca (Tabela 6) será customizada e executada em cada uma das bases de pesquisa explorando os seguintes campos de busca: título, resumo e palavras-chave. Adicionalmente, o número de estudos resultantes para cada meio de publicação será registrado após a seleção primária dos estudos com base no título e resumo. Por fim, o número de estudos finalmente selecionado de cada base de pesquisa também será registrado. A partir do conjunto de estudos primários selecionados, o título e o resumo de cada estudo será lido e avaliado, conforme os ICs e ECs definidos nesta seção. Quando necessário, a introdução e a conclusão também devem ser lidas. Dúvidas na aplicação dos critérios serão resolvidas por meio de mediações com o orientador deste projeto de mestrado acadêmico.

A.4 Extração e síntese de dados

Esta fase resultará em um conjunto de dados que descreve cada estudo primário identificado na Seção A.3. Conforme sugerido por Petersen, Vakkalanka e Kuzniarz (2015), um formulário para cada estudo primário deve ser preenchido, visando facilitar as tarefas de extração e síntese de dados. Com base nos campos propostos pelos autores supracitados, um formulário específico para este mapeamento foi elaborado, como pode ser visualizado na Tabela 7. Esses tópicos de pesquisa desse formulário serão utilizados posteriormente para responder às questões de pesquisa estabelecidas na Seção A.2.

Tabela 7 – Formulário de extração de dados

Tópico de Pesquisa	**Valor**	**QP**
ID do estudo	Valor inteiro	—
Título	Título do artigo	—
Nome do autor	Nomes dos autores	—
Filiação	Filiação dos autores	—
Tipo publicação	Conferência, Periódico, Simpósio, Workshop	—
Veículo de publicação	Nome do veículo de publicação	—
Natureza	Natureza da publicação (academia, indústria)	—
Ano	Ano de publicação	—
Base de pesquisa	Nome da base de pesquisa que recuperou o artigo	—
Modelos/Arquiteturas de referência	Nome dos modelos e arquiteturas de referência	QP 1
Domínio de aplicação	Domínio em que modelo/arquitetura foi projetado(a)	QP 2
Atributos de qualidade	Nome do atributos de qualidade	QP 3
Avaliação	Prova de conceito, estudo de caso, experimentos, casos reais em ambientes industriais, entre outros.	QP 4
Evidências que motivam a adoção desses modelos/arquiteturas	Nível de maturidade, aplicabilidade na indústria, entre outros	QP 5
Estilo arquitetural	Arquitetura em camadas, arquitetura em módulos, arquitetura em componentes, entre outros	QP 6.1
Abordagens de monitoramento do ambiente	Sensores, otimização do ambiente (detecção de falhas e/ ou degradação do QoS), entre outros	QP 6.2
Mudanças de requisitos e/ou contexto em tempo de execução	Regras, objetivos, cooperação, entre outros	QP 6.3
Recursos tecnológicos	Computação em nuvem, internet das coisas, aprendizado de máquina, entre outros	QP 6.4
Incertezas requisitos/ambiente	Condições dinâmicas, imprevisibilidades de requisitos, precisão dos dados coletados do ambiente do ambiente, disponibilidade dos recursos, entre outros	QP 7.1
Técnicas de inteligência artificial	Multi-agent system, smart elements, MAPE, autonomous entities, swarm, self-organization, entre outros	QP 7.2

A.5 Limitações

Nesta seção são apresentadas as possíveis limitações que um trabalho dessa natureza (SMS) pode apresentar, a saber:

1. **Omissão de artigos.** O mapeamento utilizará um processo de pesquisa automatizado, sendo a *string* de busca adaptada e processada pelo mecanismo de busca/pesquisa de cada base. Conforme reportado na Seção A.3, cinco bases de pesquisa para a obtenção das publicações serão utilizadas e, apesar dos esforços em tornar este trabalho mais confiável, não é possível afirmar que todos os estudos primários relacionados à temática deste SMS poderão ser recuperados. Entretanto, com o intuito de calibrar a *string* final, pesquisas com diferentes variações deverão ser realizadas para validar se estudos que deveriam ter sido recuperados foram retornados. O caso oposto também deve ser validado, ou seja, estudos que não devem ser recuperados. Em seguida, a *string* final será posteriormente adaptada para os padrões de cada uma das bases de pesquisa. Portanto, acredita-se que dessa maneira nenhum estudo relevante seja excluído do mapeamento.

2. **Confiabilidade dos pesquisadores.** Todos os envolvidos no mapeamento são pesquisadores da área de Engenharia de Software. Nenhum dos estudos incluídos neste mapeamento são de autoria dos membros do grupo de pesquisa ou de pesquisadores relacionados com os autores. Sendo assim, acredita-se que os resultados apresentados neste SMS devam apresentar o atual estado da arte da área pesquisada sem a introdução involuntária de viés.

3. **Parcialidade.** Embora um conjunto de diretrizes seja definido com a finalidade de garantir que o mapeamento seja imparcial, não é possível garantir que toda a informação coletada seja completamente imparcial, pois está sujeita a interpretação da pessoa que está conduzindo o mapeamento.

4. **Extração de dados.** Outra eventual limitação deste mapeamento está relacionada ao processo de extração de dados. As informações utilizadas para responder as QPs podem nem sempre estar presentes de maneira explícita no texto e algumas informações devem ser interpretadas. Entretanto, quando houver dúvidas/divergências a respeito de um estudo, o mesmo deve ser discutido entre os pesquisadores envolvidos a fim de estabelecer um consenso quanto ao critério a ser aplicado e/ou dado a ser extraído. Adicionalmente, vale destacar que um formulário de pesquisa foi adotado com o objetivo de auxiliar na tarefa de coleta e síntese dos dados, produzindo um arquivo padronizado de dados.

5. **Avaliação da qualidade.** Como o objetivo deste mapeamento é identificar modelos ou arquiteturas de referência para apoiar o desenvolvimento de Self-CPS voltados para o

domínio da saúde, a qualidade dos estudos não será avaliada neste SMS. Entretanto, deve-se reconhecer a importância da avaliação de qualidade e esse passo poderá ser considerado em uma futura versão deste mapeamento.

6. **Replicabilidade.** Em função das bases de pesquisa utilizadas estarem disponíveis em ambiente virtual e serem constantemente atualizadas, é possível que em uma futura pesquisa os resultados não sejam exatamente iguais, o que poderia dificultar a reprodução dos dados coletados a partir deste SMS.

A.6 Síntese dos resultados

Ao reportar os resultados deste mapeamento, as seguintes atividades devem ser realizadas: (i) discussão sobre os resultados (ou seja, descrição de cada estudo primário e detalhes de qualquer meta-análise que tenha sido realizada); (ii) discussão das principais descobertas, forças e fraquezas do mapeamento e os significados dessas descobertas como, por exemplo, aplicabilidade, benefícios, efeitos adversos e riscos; (iii) conclusão, incluindo recomendações como, por exemplo, possíveis implicações práticas para o desenvolvimento de software e para a área de pesquisa e questões de pesquisa não respondidas; e, por fim, (iv) elaborada uma discussão sobre as limitações do mapeamento.

APÊNDICE B – LISTA DE ESTUDOS PRIMÁRIOS

Na Tabela 8 é apresentada a lista de estudos primários ordenada pelo título resultante da condução do mapeamento sistemático reportado no Capítulo 3.

Tabela 8 – Lista final de estudos selecionados para a extração e síntese de dados

#	Referência
S1	Ignatius, Hargyo T.N. and Bahsoon, Rami, A Conceptual Reference Model for Human as a Service Provider in Cyber Physical Systems, 2021
S2	Castillo-Effen, Mauricio and Visnevski, Nikita A., Analysis of autonomous deconfliction in Unmanned Aircraft Systems for Testing and Evaluation, 2009
S3	Giallonardo, E. and Poggi, F. and Rossi, D. and Zimeo, E., An architecture for context-aware reactive systems based on run-time semantic models, 2019
S4	Behere, Sagar and Torngren, Martin and Chen, De-Jiu, A reference architecture for cooperative driving, 2013
S5	Dias-Ferreira, Joao and Ribeiro, Luis and Akillioglu, Hakan and Neves, Pedro and Onori, Mauro, BIOSOARM: a bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors, 2018
S6	Franco Zambonelli, Engineering self-organizing urban superorganisms, 2015
S7	Jiang, Pingyu and Liu, Chao and Li, Pulin and Shi, Haoliang, Industrial Dataspace: A Broker to Run Cyber-Physical-Social Production System in Level of Machining Workshops, 2019
S8	Lee, J. and Azamfar, M. and Singh, J. and Siahpour, S., Integration of digital twin and deep learning in cyber-physical systems: Towards smart manufacturing, 2020
S9	Dominic, Derrick and Chhawri, Sumeet and Eustice, Ryan M. and Ma, Di and Weimerskirch, André, Risk Assessment for Cooperative Automated Driving, 2016
S10	Bhat, Anand and Aoki, Shunsuke and Rajkumar, Ragunathan, Tools and Methodologies for Autonomous Driving Systems, 2018
S11	Fortino, G. and Russo, W., Towards a cloud-assisted and agent-oriented architecture for the Internet of Things, 2013

Continua na próxima página

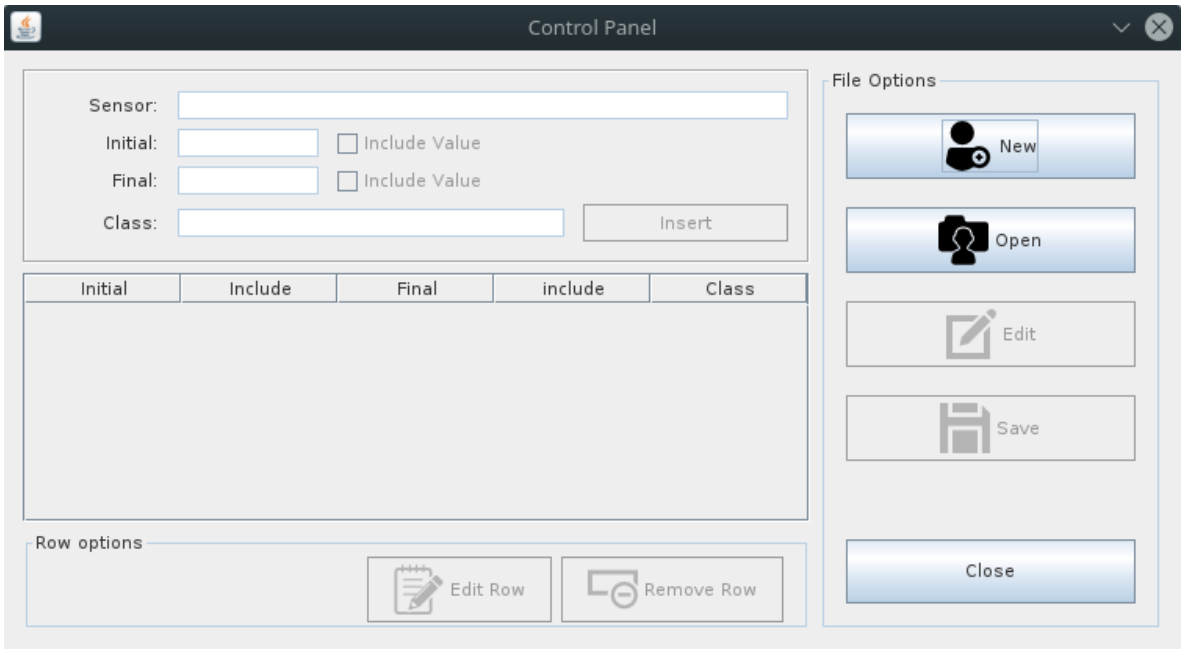
Continuando da página anterior

#	Referência
S12	Dobaj, Juergen and Krisper, Michael and Macher, Georg, Towards Cyber-Physical Infrastructure as-a-Service (CPIaaS) in the Era of Industry 4.0, 2019

APÊNDICE C – ASSISTENTE DE MODELAGEM DE SENSORES

Na Figura 42 é ilustrado o assistente de configuração de sensores. Em resumo, o usuário pode definir o nome do sensor que está sendo modelado (campo `Sensor`), o intervalo de valores (campos `Initial` e `Final`) e a classe desse intervalo (campo `Class`). No que diz respeito aos aspectos funcionais, esse assistente permite salvar os dados já no formato (extensão `.drl`) reconhecido pelo *framework* DROOLS e editar um projeto salvo previamente.

Figura 42 – Assistente de configuração de sensores



Autoria Própria