UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Rio Claro

William Filisbino Passini

Desenvolvimento de serviços compostos autoadaptativos: um *framework* baseado em implantação dinâmica, métricas de QoS e informação semântica

Rio Claro
2020

William Filisbino Passini

Desenvolvimento de serviços compostos autoadaptativos: um *framework* baseado em implantação dinâmica, métricas de QoS e informação semântica

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

Financiadora: FUNDUNESP – Proc. 017/00502-7

Orientador: Prof. Dr. Frank José Affonso

Rio Claro
2020

P288d

Passini, William Filisbino
Desenvolvimento de serviços compostos autoadaptativos: um framework baseado em implantação dinâmica, métricas de QoS e informação semântica / William Filisbino Passini. -- Rio Claro, 2020
170 p. : il., tabs.

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

1. framework. 2. arquitetura orientada a serviços. 3. sistemas autoadaptativos. 4. serviços semânticos. 5. QoS. 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.

William Filisbino Passini

Desenvolvimento de serviços compostos autoadaptativos: um *framework* baseado em implantação dinâmica, métricas de QoS e informação semântica

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

Financiadora: FUNDUNESP – Proc. 017/00502-7

Comissão Examinadora

Prof. Dr. Frank José Affonso (Orientador)
Departamento de Estatística, Matemática Aplicada e Computação (DEMAC)
Universidade Estadual Paulista "Júlio De Mesquita Filho" (UNESP) – Câmpus de Rio Claro

Prof. Dr. Valter Vieira de Camargo
Departamento de Computação (DC)
Universidade Federal de São Carlos (UFSCar) – Câmpus de São Carlos

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

Rio Claro
27 de maio de 2020

Agradecimentos

Agradeço à toda minha família, que durante toda a minha vida me incentivou a buscar novas experiências e conhecimentos, oferecendo-me total compreensão e paciência nos bons e maus momentos.

Também deixo meus sinceros agradecimentos ao meu professor e orientador, Prof. Dr. Frank José Affonso, por sua imensa contribuição durante os últimos anos, pela amizade conquistada e principalmente pela paciência, atenção e suporte que me foi dado durante todo o desenvolvimento deste trabalho.

Agradeço à todos os participantes do projeto LINEAS (Lineamentos e Zonas de Fraturas Continentais: Possíveis Corredores de Deformação da Bacia de Santos e seu Embasamento), representados especialmente nas figuras de Rodrigo Antonialli, que me proporcionou novas experiências e visões da área de computação e de Iata Anderson de Souza e Norberto Morales, responsáveis por me garantir o suporte necessário durante a condução do projeto.

Agradeço à universidade e todo o corpo docente do programa de pós-graduação, em especial ao Prof. Dr. Ivan Rizzo Guilherme por suas contribuições em um dos temas abordados nesta dissertação.

Agradeço à FUNDUNESP (Fundação para o Desenvolvimento da UNESP) pela concessão da bolsa de pesquisa em conjunto com a Petrobras, sob o processo nº 017/00502-7 (Projeto LINEAS, resultado de uma cooperação UNESP/Petrobras/FUNDUNESP).

Todas as pessoas e organizações mencionadas e também inúmeras outras com as quais tive breve experiências contribuíram para que eu pudesse chegar nesse momento e, por isso, sou grato.

As tecnologias mais profundas são aquelas que desaparecem.
Elas se tecem no tecido da vida cotidiana até que
eles sejam indistinguíveis disso.

Mark Weiser (WEISER, 1991)

Resumo

O cenário atual de desenvolvimento de software tem revelado um uso crescente de aplicações baseadas em serviços. Em um ambiente de computação distribuída, monitorar a qualidade de serviços para que essa aplicação não apresente algum tipo de anomalia ou interrupção pode ser considerada uma tarefa vital. Para isso, é necessário prover meios para o desenvolvimento de sistemas baseados em serviços capazes de observar seu estado interno de execução e/ou contexto ao qual estão inseridos e reagir diante de mudanças ou algum tipo de imprevisibilidade. Aplicações orientadas a serviços que permitem adaptação em tempo de execução podem ser classificadas como *Self-Apps* (do inglês, *Self-adaptive Service-oriented Applications*). Em função da complexidade intrínseca a esse tipo de aplicação, o uso de *frameworks* tem se mostrado uma alternativa viável por aliviar a carga cognitiva de desenvolvimento, a qual envolve o conhecimento de diversas áreas de pesquisa. Motivado por esse cenário, durante a realização deste trabalho foi conduzida uma revisão da literatura que engloba a condução de um mapeamento sistemático e consultas complementares em bases nacionais de teses e dissertações. Essa revisão permitiu obter um panorama detalhado referente ao estágio atual da pesquisa, seus desafios e as perspectivas futuras sobre *frameworks* para *Self-Apps*. Embasado nos resultados dessa revisão, o objetivo deste trabalho é o desenvolvimento do *framework* DynaMS (do inglês, *Dynamic Deployment, QoS Metrics and Semantic Search*) para apoiar a construção desse tipo de aplicação. Em linhas gerais, esse *framework* visa apoiar o desenvolvimento de aplicações baseadas em serviços simples e/ou compostos que utilizem os protocolos SOAP (do inglês, *Simple Object Access Protocol*) e REST (do inglês, *Representational State Transfer*), empregando os seguintes recursos: (i) técnicas de implantação dinâmica de serviços Web; (ii) métricas de QoS (do inglês, *Quality of Service*) para avaliação de tais serviços; e (iii) busca semântica baseada em critérios de similaridade. Como resultado, este trabalho apresenta duas contribuições relevantes: (i) o *framework* DynaMS, o qual abstrai a complexidade de desenvolvimento desse tipo de aplicação, além de propiciar vários benefícios aos interessados; e (ii) um mapeamento da literatura, pois acredita-se que esse mapeamento possa oferecer aos profissionais da indústria e comunidades científicas um panorama para nortear o aprimoramento e desenvolvimento de novas soluções para o domínio de *Self-Apps*. Por fim, cabe ressaltar que o trabalho foi avaliado por meio do desenvolvimento de uma aplicação móvel orientada a serviços para um restaurante inteligente, além de uma comparação empírica com outros *frameworks* disponíveis na literatura.

Palavras-chave: *framework*; arquitetura orientada a serviços; sistemas autoadaptativos; serviços semânticos; QoS.

Abstract

The current software development scenario has shown a crescent usage of service-based applications. In a distributed computing environment, the monitoring of the service's quality is a relevant task because aims to assure that it does not present any anomaly or interruption. To do so, it is necessary to provide means to develop service-based systems that are capable to observe their internal execution state and/or the context in which they are inserted and to react in face of changes or unforeseen circumstances. Service-oriented applications that are able to perform adaptation in runtime can be classified as Self-Apps (i.e., Self-adaptive Service-oriented Applications). Given the complexity involved in this type of application, the use of frameworks has been shown as a viable alternative to reduce the cognitive development load regarding the knowledge from different research areas. Motivated by this scenario, we conducted a literature review as a part of this work. This review includes a systematic mapping and complementary searches to national thesis and dissertations' databases. This review provided us a detailed panorama about the current stage of research, its challenges and the future perspectives regarding frameworks for Self-Apps. Based on the results from this review, the purpose of this Master's project is to present a framework called DynaMS (Dynamic Deployment, QoS Metrics and Semantic Search) to support the development of this type of application. In short, this framework aims to support the development of simple and composed service-based applications that use the SOAP (i.e., Simple Object Access Protocol) and REST (i.e., Representational State Transfer) protocols. To do so, our framework addresses three issues: (i) web service's dynamic deployment techniques; (ii) QoS (i.e., Quality of Service) metrics used to evaluate such services; and (iii) semantic search based on a set of similarity criteria. As a result, this work presents two relevant contributions: (i) the DynaMS framework, which reduces the complexity level involved in the development of this type of application and, at the same time, provides the stakeholders with other benefits; and (ii) a literature mapping, which leads us to believe that a panorama regarding both the enhancement and development of new solutions to the Self-Apps domain can be offered to industry professionals and scientific communities. Finally, it is important to highlight that we evaluated such framework through the development of a service-oriented mobile application for an intelligent restaurant and also through an empirical evaluation with other frameworks available in the literature.

Keywords: framework; service oriented architecture; self-adaptive systems; semantic services; QoS.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas e Técnicas
ACO	*Ant-Colony Optimization*
API	*Application Programming Interface*
BDTD	Biblioteca Digital de Teses e Dissertações
BiRO	*Bibliographic Reference Ontology*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CBR	*Case-Based Reasoning*
COS	*Computação Orientada a Serviços*
CPU	*Central Processing Unit*
DOM	*Document Object Model*
DynaMS	*Dynamic Deployment, QoS Metrics and Semantic Search*
EBSE	*Evidence-Based Software Engineering*
GPRS	*General Packet Radio Service*
GSM	*Global System for Mobile Communications*
HTTP	*HyperText Transfer Protocol*
IBM	*International Business Machines*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Internet Protocol*
IoT	*Internet of Things*
IRI	*Internationalized Resource Identifier*
ITU	*International Telecommunication Union*
JAX-WS	*Java API for XML Web Services*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
LED	*Light-Emitting Diode*
MAPE-K	*Monitor, Analyze, Plan, Execute – Knowledge*
NI	Não Informado
OE	Objetivo Específico
OWL	*Web Ontology Language*
OWL-S	*Web Ontology Language for Service*
PSO	*Particle Swarm Optimization*
QoS	*Quality of Service*
QP	Questão de Pesquisa
RBR	*Rule-Based Reasoning*
RDF	*Resource Description Framework*
REST	*Representational State Transfer*
RFID	*Radio Frequency Identification*
SaS	*Self-adaptive Systems*
SASeS	*Self-Adaptive Service-based Systems*
Self-Apps	*Self-adaptive Service-Oriented Applications*
SEVOCAB	*Software and Systems Engineering Vocabulary*

SC	Sistemas Críticos
SI	Sistemas da Informação
SLA	*Service Level Agreement*
SMS	*Systematic Mapping Study*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SPARQL	*SPARQL Protocol and RDF Query Language*
SQL	*Structured Query Language*
SWSO	*Semantic Web Service Ontology*
TCP	*Transmission Control Protocol*
TF	Trabalho Futuro
UDDI	*Universal Description, Discovery and Integration*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
WS	*Web-Service*
WSDL	*Web Service Description Language*
WSMO	*Web Service Modeling Ontology*
Xerox PARC	*Xerox Palo Alto Research Center*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition*

Sumário

1 Introdução

1.1 Contextualização

Sistemas de software têm ocupado um papel importante na sociedade atual, auxiliando diversos segmentos através da automatização parcial e, em alguns casos, completa de determinadas tarefas (STEPHAN *et al.*, 2012; BARRETO; AMARAL; PEREIRA, 2017; HATTON; SPINELLIS; VAN GENUCHTEN, 2017). Tal automatização tem proporcionado maior agilidade e conforto, introduzindo evoluções em diferentes áreas, novas opções de lazer à população, entre outros benefícios. Segundo Granados-Colmenares *et al.* (2018), as evoluções proporcionadas pela tecnologia podem ser classificadas a partir de três diferentes grupos, a saber: físico, digital e biológico.

O grupo **Físico** representa diferentes setores da indústria, os quais podem ser exemplificados pelos seguintes itens: (i) veículos autônomos, que pode ser traduzido na utilização de um carro dotado de sensores e técnicas de inteligência artificial. Esse tipo de veículo tem sido utilizado no transporte de cargas, na identificação de áreas congestionadas, no transporte de pessoas com deficiências visuais ou motoras, entre outros exemplos; (ii) indústria de materiais, que permite a criação de diferentes tipos de produtos que fujam dos padrões de processo de manufatura convencionais. Nesse sentido, pode-se citar a utilização de impressoras 3D para criação de produtos para diferentes segmentos: automotivo, têxtil, entretenimento, alimentício, saúde, entre outros; e (iii) assistentes virtuais baseados em inteligência artificial, os quais podem ser empregados no dia a dia como uma maneira de adequar um ambiente interno aos interesses individuais dos usuários e, ao mesmo tempo, oferecer informação sob demanda controlada por comandos de voz.

A camada intermediária entre as pessoas e os dispositivos que compõem o grupo físico é representada pelo grupo **Digital**. Neste último se encontram os sistemas de software responsáveis por controlar um ou mais dispositivos, além de desempenharem diferentes tipos de tarefas. Para esse grupo podem ser exemplificados sistemas de comércio de produtos e serviços via Internet, conectando dispositivos de diferentes usuários em locais fisicamente distintos para que possam, por exemplo, efetuar compras.

O grupo **Biológico** representa o setor de biotecnologia, o qual descreve qualquer aplicação tecnológica, por meio de pesquisa científica, que utilize sistemas biológicos, organismos vivos, ou seus derivados, para fabricar ou modificar produtos e/ou processos para utilização específica. Produtos alimentícios transgênicos e o desenvolvimento de vacinas/medicamentos são exemplos desse grupo.

Em nossa sociedade podem ser identificadas tecnologias relacionadas com os três grupos supracitados, mesmo que muitas vezes não seja possível identificá-las diretamente. Por exemplo, o setor de entretenimento tem se beneficiado especialmente dos grupos físico e digital. Dentro desse setor é interessante mencionar a indústria cinematográfica, que utiliza: (i) recursos modernos como ferramentas de animação e digitalização para a produção; e (ii) redes de Internet e plataformas de transmissão de conteúdo sob demanda para distribuição do conteúdo produzido ao consumidor final. Os segmentos que envolvem jogos, comunicação e planejamento de eventos também são bons exemplos que têm se beneficiado com a evolução dos sistemas de software (JONES, 2013; MARCHAND; HENNIG-THURAU, 2013). Todo esse ecossistema tem sido alavancado nos últimos anos, em função, principalmente, da adoção de dispositivos denominados inteligentes, a saber: *smartphones*, *tablets*, *smart-TVs*, entre outros. A adoção desses dispositivos possibilita acesso ao conteúdo produzido por diferentes segmentos de maneira constante, sendo necessário apenas que o usuário disponha de uma conexão de Internet adequada para acessar tal serviço/conteúdo. Todo esse progresso foi possível majoritariamente em decorrência de avanços obtidos em três diferentes áreas:

Hardware. Esta área foi beneficiada graças ao investimento constante em pesquisas voltadas para o desenvolvimento de componentes mais potentes, baratos e acessíveis, viabilizando a criação de dispositivos cada vez mais eficientes, baratos e distribuídos. Esses investimentos resultaram em avanços tecnológicos para diferentes dispositivos comumente utilizados em nosso dia a dia, tais como: televisores, geladeiras, carros, entre outros;

Software. O investimento em software traz uma experiência cada vez mais completa para uma grande parcela dos usuários, possibilitando utilizar diferentes dispositivos para cumprir tarefas antes realizadas exclusivamente em um computador comum ou até mesmo sem qualquer tipo de auxílio de uma máquina. A possibilidade de executar tarefas bancárias de maneira digital representa um bom exemplo dos resultados proporcionados pelo investimento em software; e

Redes de comunicação. O investimento nesta área proporciona o estabelecimento de métodos eficientes, rápidos e confiáveis de comunicação, aspectos importantes para as duas áreas supracitadas. Sendo assim, outra área de importância para o desenvolvimento de sistemas de software é a voltada para o setor de comunicação e transmissão de dados, a qual envolve os padrões de rede adotados e as evoluções das gerações da telefonia móvel (3G, 4G, 4G+ e, mais recentemente, o 5G).

Em decorrência da evolução das áreas supracitadas, tanto na indústria quanto na academia, tem-se notado esforços em estabelecer métodos de integração entre dispositivos inteligentes e sistemas de software (DÍAZ; MARTÍN; RUBIO, 2016; DATTA *et al.*, 2016). Assim, pode-se

observar algumas iniciativas na área de Engenharia de Software em desenvolver metodologias, modelos arquiteturais e processos que possam otimizar o uso de diferentes dispositivos como um meio de acesso aos sistemas de software existentes, como, por exemplo, aplicações corporativas, softwares de entretenimento, entre outros (JAMSHIDI; AHMAD; PAHL, 2013; AHMAD *et al.*, 2016; HAN; CRESPI, 2017). Nesse sentido, a integração de sistemas baseados em SOA (do inglês, *Service-Oriented Architecture*) com tais dispositivos, aliada à técnicas de encapsulamento de dados, tem se mostrado uma alternativa factível para viabilizar o desenvolvimento de tais sistemas (BARRY, 2003; TRINUGROTO; REICHERT; FENSLI, 2011; ZHANG; CHEN; CHENG, 2017; TAO *et al.*, 2018). Para esses autores, a adoção desse tipo de aplicação permite estabelecer uma "ponte" entre as aplicações implantadas em diferentes dispositivos, composta por um *front-end*, utilizado para interação com o usuário, e por um *back-end* corporativo, utilizado para a realização do processamento intensivo de dados. A conexão entre *front-end* e *back-end* é realizada através da utilização conjunta de protocolos de comunicação e de mensagens padronizadas, possibilitando a troca de informações entre diferentes serviços e, consequentemente, sua integração para atuar em um determinado conjunto de tarefas. Por exemplo, no contexto de SOA é comum encontrar dois protocolos de comunicação frequentemente utilizados: SOAP (do inglês, *Simple Object Access Protocol*) e REST (do inglês, *Representational State Transfer*) (ERL, 2007; MALYK, 2017; DAIGNEAU, 2011).

Baseado no contexto exposto, a redução, ou até mesmo mitigação, dos efeitos causados por falhas ou degradação de qualidade de serviço (do inglês, *Quality of Service* – QoS) pode ser considerada vital para o desenvolvimento de aplicações baseadas em serviços. Dessa forma, prover meios para o desenvolvimento de sistemas baseados em serviços capazes de observar seu estado interno de execução e/ou contexto ao qual estão inseridos e reagir diante de mudanças ou algum tipo de imprevisibilidade tem se mostrado uma alternativa viável (PSAIER *et al.*, 2010; ALHOSBAN *et al.*, 2015; SHAFIUZZAMAN; NAHAR; RAHMAN, 2015; KHANFIR; DJMEAA; AMOUS, 2017; GATOUILLAT; BADR; MASSOT, 2018; GILL *et al.*, 2019). Tais meios podem tornar esses sistemas mais robustos, confiáveis e resilientes face a algum tipo de adversidade/anomalia em seu estado interno ou de contexto. De acordo com Cugola *et al.* (2014), Menasce *et al.* (2011), aplicações orientadas a serviços que permitem adaptação em tempo de execução podem ser classificadas como serviços autoadaptativos. Esse tipo de aplicação será referenciado deste ponto em diante como *Self-Apps* (do inglês, *Self-adaptive Service-oriented Applications*). Nesse sentido, pode-se dizer que tais aplicações são projetadas para detectar e corrigir eventuais anomalias de maneira autônoma, atuando sem a necessidade de intervenção humana e preservando sua integridade em tempo de execução. Não apenas isso, o desenvolvimento de sistemas autoadaptativos em geral possibilita atender a requisitos que não tenham sido previstos durante a fase de concepção, sem a necessidade de sua reinicialização. Ao tornar a aplicação adaptável em tempo de execução, é possível evitar possíveis transtornos

causados aos seus usuários, como, por exemplo, perda monetária em cenários que envolvam operações financeiras (SALEHIE; TAHVILDARI, 2009; CHENG *et al.*, 2009; LEMOS *et al.*, 2013; LEMOS *et al.*, 2017).

O desenvolvimento de *Self-Apps* envolve um elevado grau de dificuldade, pois a interação entre todos os elementos supracitados resulta em um complexo ecossistema composto por (i) serviços Web, os quais podem ser implementados de diferentes maneiras e utilizar diferentes protocolos de comunicação; (ii) sistemas autoadaptativos, que podem utilizar diferentes abordagens de monitoramento e tomada de decisão; e (iii) necessidade de se manter um determinado grau de qualidade e resiliência. Dado esse cenário, a utilização de *frameworks* durante o desenvolvimento desse tipo de aplicação pode se mostrar vantajosa. Sua adoção possibilita ao desenvolvedor abstrair parte dessa complexidade, permitindo aproveitar-se dos benefícios proporcionados pela área de *Self-Apps* e, ao mesmo tempo, focar especialmente no desenvolvimento de uma aplicação que atenda às necessidades e funcionalidades requisitadas por seu cliente (MWENDI, 2014; MYLLÄRNIEMI *et al.*, 2018).

1.2 Motivação

Motivado pelo contexto exposto e pela necessidade de se obter uma visão geral do tema de pesquisa entorno das áreas envolvidas, uma revisão da literatura foi conduzida durante o desenvolvimento deste trabalho. O principal intuito dessa revisão foi estabelecer um panorama sobre o desenvolvimento de sistemas de software baseados em serviços Web simples e/ou compostos que requerem características de autoadaptação. Mais especificamente, foi dada atenção a *frameworks* e/ou derivados que possam apoiar o desenvolvimento de tais sistemas, pois estes permitem abstrair uma porção considerável da complexidade de desenvolvimento. Tal abstração garante que o desenvolvedor desse tipo de aplicação possua acesso a um conjunto de ferramentas que possibilite a implementação dos recursos necessários para o desenvolvimento de maneira rápida e transparente. Para conduzir tal revisão, a técnica de mapeamento sistemático (do inglês, Systematic Mapping Study – SMS) foi adotada (KITCHENHAM; CHARTERS, 2007; KITCHENHAM; DYBA; JORGENSEN, 2004; PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Essa técnica permite que os pesquisadores realizem uma avaliação mais completa e justa sobre um tópico de pesquisa através de uma metodologia confiável, rigorosa, auditável e isenta de influências pessoais. Além disso, vale destacar que essa revisão foi complementada através de consultas similares em bases nacionais de teses e dissertações. A partir da análise do conjunto de estudos obtidos por essa revisão da literatura, pôde-se observar que o tema deste trabalho (ou seja, *frameworks* para *Self-Apps*) é amplo e passível de investigação. Embora os estudos analisados reportem um conjunto comum de características sobre as técnicas e tecnologias mais utilizadas, pode-se observar que não há um

consenso sobre uma abordagem ou especificação de um padrão amplamente aceito dentro do contexto de *Self-Apps*.

Em paralelo, porém não menos importante, esse mapeamento também evidenciou a problemática por trás dos diferentes protocolos de comunicação utilizados na troca de dados entre a aplicação cliente e os serviços que estão sendo consumidos. Cada protocolo pode apresentar diferentes limitações no desenvolvimento e implantação de *Self-Apps*. Por exemplo, serviços baseados no protocolo SOAP requerem que sejam geradas *stubs* dos serviços dentro da aplicação cliente (GONCALVES, 2013; ERL, 2007). Essas *stubs* são geradas na fase de *design* de uma aplicação, possibilitando simular o comportamento de um código presente em uma máquina remota. Consequentemente, essa característica impõe um desafio ao projeto de serviços capazes de conduzir um processo de autoadaptação em tempo de execução. Quando não contornada, essa limitação inviabiliza a substituição de um serviço com algum tipo de anomalia (por exemplo, falha ou degradação de QoS) por um outro operacional, sem que a aplicação seja reinicializada.

Diante da problemática exposta nesta seção, pode-se notar que o desenvolvimento de sistemas dotados da capacidade de reconhecer seu ambiente de execução e atuar de maneira proativa e/ou reativa às alterações que ocorrem nesse meio é um assunto que vem sendo abordado nos últimos anos e ainda possui potencial a ser explorado pela comunidade científica (TREIBER *et al.*, 2010; LEITNER *et al.*, 2012; LI *et al.*, 2013; ARCAINI; RICCOBENE; SCANDURRA, 2015; GUPTA; KAMAL; SUMAN, 2018; ABDELHAK; FETH-ALLAH; MOHAMMED, 2019; ASCHOFF; ZISMAN; ALEXANDRE, 2019). No entanto, pela complexidade intrínseca desse tipo de aplicação, o uso de *frameworks* tem se mostrado uma alternativa viável, aliviando a carga de desenvolvimento (FORESTIERO *et al.*, 2010; LI; ZHANG; YANG, 2012; PARK; SONG, 2015; ACHTAICH *et al.*, 2018). Nesse sentido, sobre as lacunas e temas em aberto, ambos expostos nesta seção, pode-se dizer que nenhuma das iniciativas encontradas em nossa revisão da literatura apresenta uma solução que contemple, em sua totalidade, o seguinte conjunto de características: (i) ofereça suporte a serviços baseados em SOAP e REST; (ii) apresente metodologia e infraestrutura robusta para documentação e armazenamento de serviços; (iii) possibilite a implantação de serviços em tempo de execução; (iv) realize uma avaliação de serviços baseada em métricas de QoS; (v) realize uma seleção de serviços por similaridade; e (vi) disponha de um repositório semântico de serviços. Portanto, essas características podem ser consideradas como fatores de motivação para a realização deste projeto de mestrado acadêmico.

1.3 Objetivos

Este trabalho tem como objetivo geral o desenvolvimento do *framework* DynaMS (do inglês, *Dynamic Deployment, QoS Metrics and Semantic Search*) para apoiar a construção

de aplicações baseadas em *Self-Apps*. Em linhas gerais, esse *framework* visa apoiar o desenvolvimento de aplicações baseadas em serviços SOAP e REST por meio de uma abordagem de implantação dinâmica. Para que esse objetivo geral seja alcançado, os seguintes Objetivos Específicos (OE) são propostos:

OE-1. Possibilitar o desenvolvimento de *Self-Apps*, atuando em todo o ciclo de vida das aplicações, o qual compreende as fases de *design* (concepção) e *runtime* (tempo de execução). Para isso, modelos de representação de serviços Web e de coreografias devem ser elaborados para que o engenheiro de software faça o projeto de uma *Self-App*. Tais modelos devem ser utilizados na fase de *runtime* por processos automatizados para adaptação de serviços.

OE-2. Desenvolvimento de métodos responsáveis por interpretar a documentação de serviços SOAP e REST e traduzi-la para o modelo de representação de serviços proposto (OE-1), viabilizando sua posterior tradução para o repositório semântico (OE-4);

OE-3. Desenvolvimento de uma camada intermediária para atuar na comunicação entre a aplicação cliente e o serviço Web, possibilitando a realização das operações de adaptação em tempo de execução sem percepção do usuário;

OE-4. Especificação e desenvolvimento de um repositório semântico, responsável por armazenar e gerenciar os serviços Web. Tal repositório deve permitir localizar serviços pelos seguintes critérios de similaridade: (i) técnico, o qual envolve a análise de métodos e parâmetros; (ii) semântico, retratando informações de contexto do serviço; e (iii) híbrido, constituindo uma combinação das duas abordagens anteriores;

OE-5. Especificação e desenvolvimento de um sistema de monitoramento, o qual deve ser responsável por monitorar os serviços em tempo de execução. Em síntese, tal sistema deve monitorar o estado atual de um serviço e também prover candidatos à substituição de um serviço degradado ou quebrado, classificando-os de acordo com um conjunto de métricas de QoS;

OE-6. Minimizar a carga cognitiva de aprendizado dos interessados[1] nesse tipo de *framework*, fazendo com que os desenvolvedores não tenham uma carga de aprendizado elevada para sua utilização. Para isso, é necessário a investigação de ferramentas, técnicas e processos utilizados para desenvolvimento de serviços baseados em SOAP e REST; e

OE-7 Avaliação do *framework* DynaMS sob o ponto de vista do impacto de utilização. Além disso, essa avaliação deve permitir obter parâmetros quanto ao comportamento dos componentes desse *framework*, detalhados nos objetivos específicos supracitados. Para isso,

[1] Profissionais da indústria e comunidades científicas

pretende-se utilizar técnicas similares de avaliação às reportadas no mapeamento apresentado no Capítulo 5.

Conforme reportado na Seção 1.2, um mapeamento da literatura foi conduzido durante a execução deste trabalho. Portanto, outro objetivo geral deste trabalho é apresentar um panorama detalhado referente ao estágio atual, desafios de pesquisa e perspectivas futuras sobre o desenvolvimento de *frameworks* para *Self-Apps*. Essa investigação pode nortear pesquisadores e/ou profissionais da indústria no desenvolvimento desse tipo de aplicação e/ou no projeto de novos *frameworks* para a área.

1.4 Organização

O Capítulo 2 apresenta o conceito de computação orientada a serviços, onde são detalhados alguns dos elementos que compõem esse tipo de computação, tais como: serviços Web, SOA e QoS. No Capítulo 3 são introduzidos conceitos sobre sistemas autoadaptativos, o modelo de adaptação MAPE-K (do inglês, *Monitor, Analyze, Plan, Execute–Knowledge*), as características e propriedades envolvidas nesse tipo de sistema, além dos desafios e limitações envolvidos nessa área. No Capítulo 4 são apresentadas definições para ontologias e Web semântica, abordagens utilizadas neste trabalho para facilitar nas tarefas de descrição e pesquisa de serviços Web. Em seguida, no Capítulo 5 são reportados os resultados de um mapeamento sistemático da literatura e de pesquisas complementares em bases nacionais de teses e dissertação. No Capítulo 6 são apresentados os detalhes do *framework* DynaMS, proposto nesta dissertação, o qual é posteriormente avaliado no Capítulo 7 por meio de um estudo de caso e de uma análise comparativa com outros *frameworks* similares encontrados na revisão apresentada no Capítulo 5. Por fim, o Capítulo 8 apresenta as conclusões e contribuições oriundas deste trabalho, bem como perspectivas para trabalhos futuros.

2 Computação Orientada a Serviços

Segundo Erl (2007), o termo Computação Orientada a Serviços (COS) é muito abrangente, podendo ser considerada como uma nova geração da computação distribuída. Tipicamente, uma plataforma de computação orientada a serviços é composta por um certo conjunto de elementos, a saber: (i) arquitetura orientada a serviços; (ii) orientação a serviços; (iii) lógica orientada a serviços; (iv) serviços; (v) composição de serviço; e (vi) inventário de serviços. Entretanto, em virtude de sua ampla adoção, o modelo arquitetural orientado a serviço ou, simplesmente, SOA tornou-se quase que um sinônimo para o que a COS representa. Em outras palavras, esse termo é muito mais do que isso, estando relacionado diretamente com o paradigma de *design* por trás da orientação de serviços e seu relacionamento com SOA. A COS abrange aspectos como paradigma e princípios de *design*, catálogos de modelos de *design*, linguagens padrões, conceitos e tecnologias relacionadas. Diante do exposto, o objetivo deste capítulo é apresentar definições para alguns dos elementos da COS, focando principalmente nos que tiveram maior impacto no desenvolvimento deste trabalho. Sendo assim, inicialmente, na Seção 2.1, é apresentada uma definição para serviços Web. Em seguida, a Seção 2.2 apresenta conceitos e definições para SOA. Por fim, a Seção 2.3 define "Qualidade de Serviço", termo útil para o estabelecimento de um conjunto de métricas a serem utilizadas como critérios de seleção de serviços.

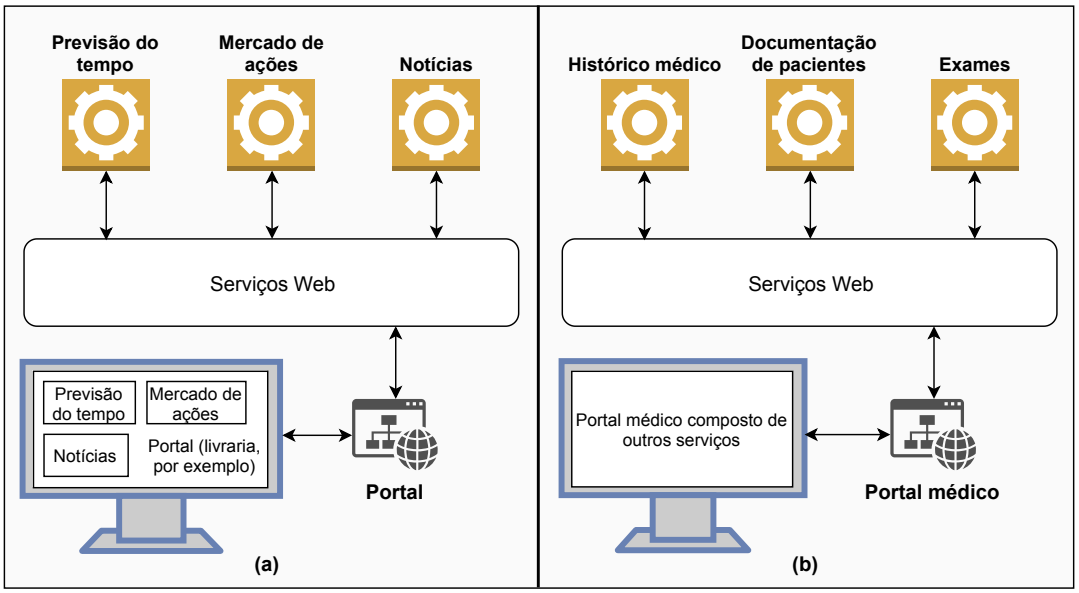
2.1 Serviços Web

De acordo com Oracle (2017), serviços Web são aplicações do tipo cliente-servidor que se comunicam através da Web por meio do protocolo HTTP (do inglês, *HyperText Transfer Protocol*). Esse tipo de aplicação provê um conjunto de padrões para que aplicações de software executadas em diferentes plataformas e/ou *frameworks* comuniquem-se e troquem informações com seus interessados. Segundo a W3C (2004b), um serviço Web consiste de uma noção abstrata que deve ser implementada através de um agente concreto. Nesse caso, o agente representa a parte de software ou hardware responsável por enviar e receber mensagens, resultando em um recurso composto por um conjunto abstrato de funcionalidades.

Em Barry (2003) é apresentado um exemplo simplificado de uma das primeiras utilizações de serviços Web, como ilustra a Figura 1. Nesse exemplo, Figura 1a, os serviços são representados por pequenos componentes tipicamente adicionados em um portal, cuja finalidade é fornecer informações como preço de ações, previsão do tempo e notícias. O cenário ilustrado mostra pequenas aplicações secundárias com a finalidade de fornecer informações sem qualquer relação com o ambiente. Em contrapartida, a Figura 1b ilustra a evolução dos serviços ao longo do

tempo, mostrando que os mesmos passaram a assumir tarefas mais voltadas para o domínio das aplicações. Indiretamente, ao atuarem em atividades mais complexas e essenciais das aplicações os serviços foram isolados em diferentes camadas e módulos, auxiliando também na distribuição do poder computacional exigido por tais aplicações através de diferentes servidores.

Figura 1 – Exemplos de diferentes utilizações de serviços Web



Fonte: adaptado de Barry (2003)

Para Erl (2007), é possível separar serviços através de três classificações distintas, a saber:

1. **Serviço de entidade:** representa um serviço focado no contexto ao qual pertence. Sendo assim, descreve entidades de negócios relevantes para a organização responsável por seu desenvolvimento e manutenção. Um serviço desse tipo deve ser capaz de descrever uma entidade (por exemplo, `Funcionário`) e manipulá-la através de um determinado conjunto de operações relacionadas a inserção, leitura, atualização e remoção de dados;

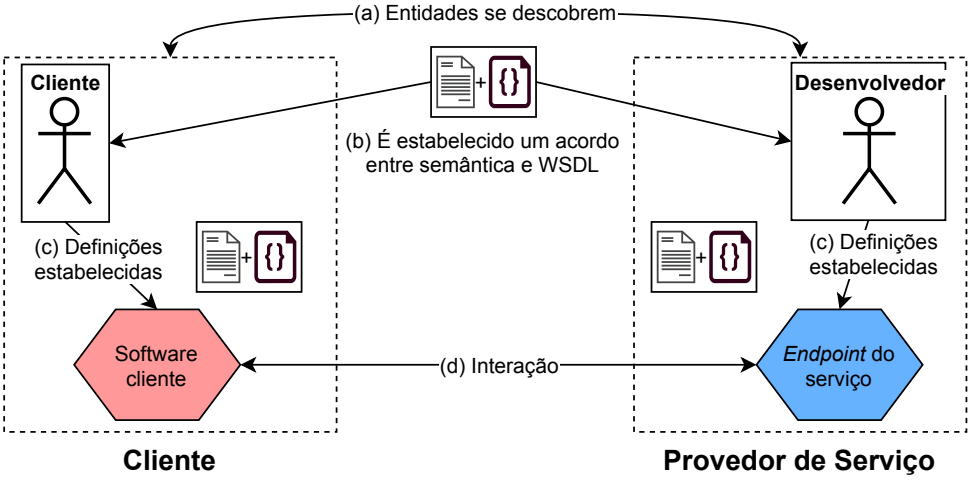
2. **Serviço-tarefa:** representa um serviço que é diretamente associado com a execução de uma tarefa ou processo específico dentro do contexto da aplicação. Embora similar ao objetivo de um serviço de entidade, o serviço-tarefa apresenta uma distinção relacionada ao escopo da aplicação. Enquanto a obtenção dos dados de um determinado funcionário pertence a um serviço de entidade, a execução de um certo conjunto de operações que envolvam um ou mais funcionários e diferentes entidades com a finalidade de atingir um determinado objetivo é realizada por um serviço-tarefa; e

3. **Serviço utilitário:** diferentemente dos tipos de serviços apresentados anteriormente, um serviço utilitário não possui foco primário na lógica de negócio da aplicação. Sua função

principal é prover funcionalidades mais voltadas para a automação de determinadas tarefas, atuando em requisitos não funcionais como operações de registro de eventos em *logs*, notificação da ocorrência de determinados eventos, tratamento de exceções, entre outras.

A comunicação entre um serviço Web e seu respectivo usuário pode ocorrer de diferentes maneiras, mas, em geral, segue alguns passos específicos, como ilustrado na Figura 2. De maneira geral, o cliente e o provedor de serviço devem "se encontrar" para que uma comunicação possa ser estabelecida (Figura 2a). Tradicionalmente, o cliente é o responsável por iniciar uma comunicação por meio de requisições enviadas ao provedor de serviço. Em seguida, é necessário garantir que ambos sejam capazes de atender às suas respectivas necessidades (Figura 2b), ou seja, que o cliente possa enviar as informações necessárias para que o serviço execute as tarefas solicitadas. Paralelamente, o serviço deve ser capaz de responder adequadamente às requisições enviadas pelo cliente. Uma possível maneira de realizar essa etapa é através da combinação de uma linguagem de descrição de serviço (do inglês *Web Service Description Language* – WSDL) e de uma descrição semântica responsável por apresentar o contexto em que o serviço atua. Uma vez que esse conjunto de definições tenha sido estabelecido (Figura 2c), uma interação entre o cliente e o provedor de serviços se torna viável (Figura 2d).

Figura 2 – Processo padrão de chamada inicial de um serviço Web

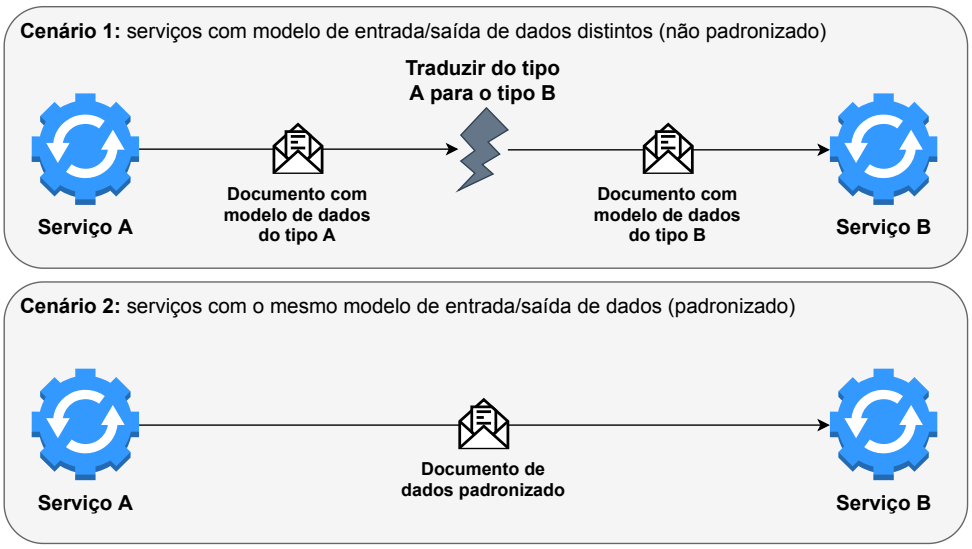


Fonte: adaptado de W3C (2004b)

Ainda sobre a comunicação entre serviços Web, a maneira como a troca de dados ocorre pode variar, como mostra a Figura 3. No primeiro cenário, temos dois serviços Web distintos, cada um com seu respectivo modelo de dados. Para que ocorra troca de informações é necessário que aconteça uma tradução à nível dos dados das entidades envolvidas, adaptando os dados para o modelo utilizado em cada um dos serviços envolvidos na comunicação. Em contrapartida, o segundo cenário apresenta uma evolução desse modelo de troca de informações, onde um determinado nível de padronização previamente acordada durante o desenvolvimento

dos serviços é adotado, permitindo trocar informações sem a necessidade de qualquer tradução prévia entre entidades. Nesse sentido, um modelo padrão de representação de dados é gerado e os serviços são desenvolvidos de forma que sejam capazes de interpretar esse modelo e realizar operações com os dados transmitidos. Uma forma comum de representar essas informações é através da adoção do padrão JSON (do inglês, *JavaScript Object Notation*).

Figura 3 – Cenários de troca de dados entre dois serviços distintos



Fonte: adaptado de Erl (2007)

2.1.1 Composição e orquestração de serviços

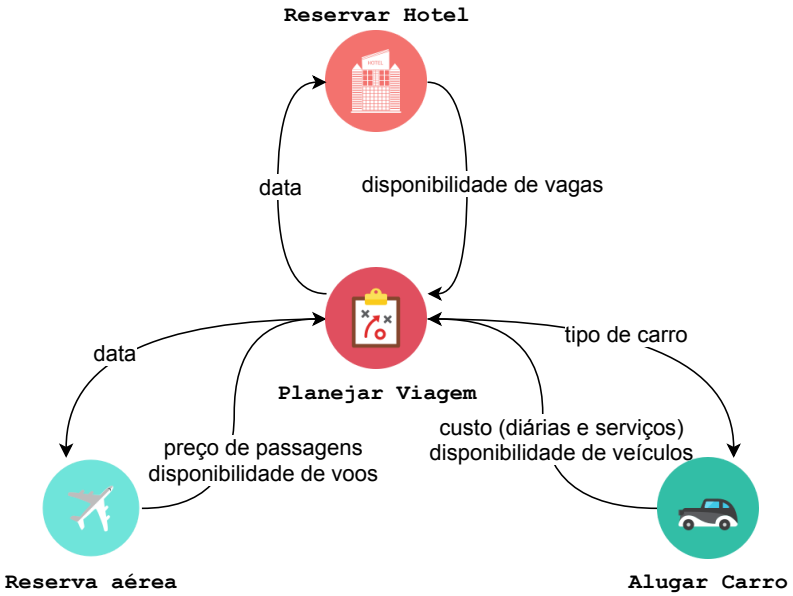
Devido a sua natureza capaz de prover transparência de linguagem através de arquivos de descrição de linguagem e de contexto semântico, um novo serviço Web pode ser concebido através da combinação de vários outros serviços Web, técnica conhecida na literatura por composição de serviços. Para Narendra e Orriens (2006), a tarefa de composição de serviços tem como principal objetivo lidar com requisições de usuários ou de sistemas que não puderam ser satisfeitas por nenhum serviço Web já existente. Segundo Limthanmaphon e Zhang (2003), é necessário combinar um conjunto de serviços Web disponíveis de maneira a obter um novo serviço que será adequado às necessidades de um interessado ou de um sistema responsável pela requisição de uma funcionalidade. Esse novo serviço gerado por uma composição tradicionalmente possui uma característica inerente a essa técnica, pois seu papel em relação ao cliente/provedor de serviço pode ser alterado em tempo de execução, dependendo do escopo em que a ação que está sendo executada. No momento em que um usuário faz uma requisição ao serviço composto, este age como um provedor de serviço. Entretanto, ao distribuir as tarefas para os demais serviços envolvidos na composição, o serviço composto assume o papel de cliente. Assim, ao enviar um conjunto de requisições para os demais serviços, para aquela relação

específica, o serviço composto deixa de exercer o papel de provedor, tornando-se nesse momento um cliente consumidor.

Chakraborty e Joshi (2001) classificaram a composição de serviços em duas categorias: (i) proativa, etapa da composição que foi feita de maneira manual ou passou por uma etapa de pré-compilação, resultando em serviços considerados estáveis e ideais; e (ii) reativa, etapa da composição que ocorreu em tempo de execução sem a participação dos seres humanos. Um serviço composto também pode exigir troca síncrona ou assíncrona de mensagens envolvendo diferentes participantes, sendo necessário que múltiplos serviços Web colaborem entre si. Dessa forma, uma sequência de passos bem definida (ou seja, uma orquestração de serviços) deve ser elaborada, a qual representará um novo serviço oriundo de uma operação composta por vários outros serviços Web (ZHANG; ZHANG; CAI, 2008). Uma orquestra representa uma boa analogia para orquestração de serviços: nesse cenário, um grupo de músicos está tocando vários instrumentos diferentes em momentos distintos, mas todos estão sendo coordenados por uma figura central, um maestro. Um serviço responsável pela orquestração de outros serviços age de maneira semelhante a um maestro de orquestra, garantindo que todas as tarefas necessárias sejam executadas, seguindo uma organização pré-determinada, para que uma operação seja cumprida (RICHARDS, 2016).

A Figura 4 ilustra a relação entre composição e orquestração de serviços através de um exemplo de planejamento de viagem composto por diferentes serviços, a saber: (i) reserva de passagem aérea; (ii) reserva de hotel; e (iii) aluguel de carro. Nesse exemplo, uma viagem requer um planejamento que envolva a compra de uma passagem aérea, a reserva de um hotel e o aluguel de um carro. Esse planejamento é representado pelo serviço `Planejar viagem`, **composto** pelos serviços `Reserva aérea`, `Reservar hotel` e `Alugar Carro`. Além de representar uma composição de serviços, o serviço composto `Planejar Viagem` também atua como orquestrador, especificando a **ordem dos passos** que serão executados e fornecendo as informações necessárias para cada um dos serviços. Esse tipo de atividade requer a presença de um orquestrador de serviço por apresentar uma dependência inerente entre o período de reserva dos serviços, afinal, para que a viagem ocorra de maneira satisfatória é necessário que o intervalo de datas das reservas seja o mesmo nos três serviços. Sendo assim, o serviço composto `Planejar Viagem` depende de informações obtidas através da comunicação com outros serviços, tais como: a disponibilidade de passagem aérea (`Reserva aérea`), o número de vagas disponíveis em hotéis (`Reservar hotel`), a disponibilidade do tipo de carro pesquisado na data desejada (`Alugar carro`) e o custo envolvido na contratação de cada um dos serviços (W3C, 2002).

Figura 4 – Exemplo de uma composição e orquestração de serviços



Fonte: Elaborada pelo autor

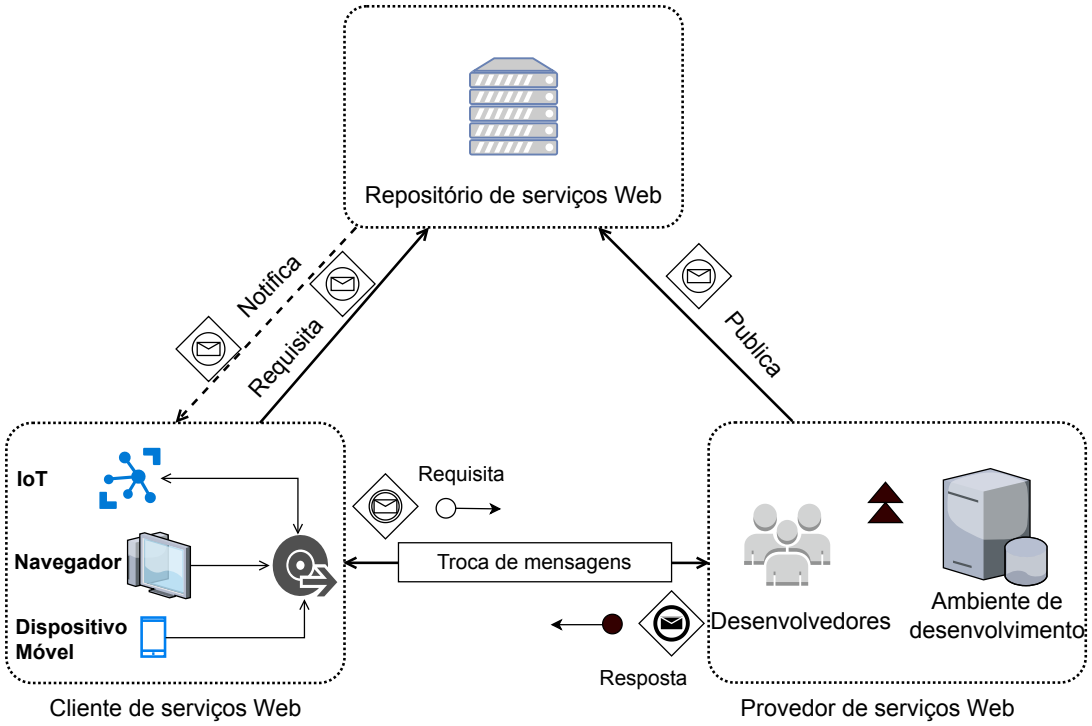
2.2 Arquitetura Orientada a Serviços

De acordo com Erl (2005) e Oasis (2012), uma arquitetura orientada a serviços é tradicionalmente composta por três elementos e por suas respectivas interações, como mostra a Figura 5. O **Provedor de serviços** representa a organização responsável pela criação e publicação de um serviço para a execução de uma determinada tarefa, o qual pode ser registrado em um **Repositório de serviços Web**. O repositório tem como finalidade facilitar a descoberta dos serviços disponibilizados pelos provedores de serviços. Para isso, armazena os *endpoints* através dos quais um determinado serviço pode ser acessado e também pode armazenar algumas informações relacionadas ao processo de comunicação, providas por documentos WSDL e/ou de descrição semântica. Tradicionalmente, o repositório é representado por um serviço de diretório denominado UDDI (do inglês, *Universal Description, Discovery and Integration*), onde é possível publicar e localizar outros serviços. Esse repositório armazena os metadados de um serviço Web, bem como seu arquivo WSDL, que contém a descrição dos serviços e as operações por ele fornecidas. Por fim, há o **Cliente de serviços Web**, que representa os consumidores de serviços, que podem ser tanto usuários quanto aplicações que irão utilizar um determinado serviço para executar suas tarefas.

Em sua forma tradicional, em SOA toda a interação entre cliente e servidor ocorre através do protocolo de comunicação SOAP. Nesse protocolo, cada serviço pode ser descrito utilizando arquivos WSDL e XSD (do inglês, *XML Schema Definition*[1]), normalmente formatados pela

[1] Um *schema* é um arquivo codificado que contém a definição da estrutura de um documento XML, no qual são definidos tipo, tamanho e regras de preenchimento dos termos que compõem o serviço.

Figura 5 – Elementos de uma Arquitetura Orientada a Serviços



Fonte: adaptado de Oasis (2012)

linguagem XML (do inglês, *eXtensible Markup Language*). A finalidade desses arquivos é expor toda a estrutura de chamadas de um determinado serviço, detalhando quais operações estão disponíveis e quais são os tipos de parâmetros de entrada e tipos de retorno esperados. Embora de fácil compreensão, esses arquivos são formados por um grande número de *tags*, as quais não serão totalmente descritas neste momento por motivos de escopo, sendo destacadas apenas as mais essenciais utilizadas na elaboração deste trabalho, a saber:

1. ***Definitions:*** contém uma descrição geral do serviço, fornecendo informações como seu nome e *namespace*, além de incluir todas as demais informações sobre o serviço. Esta é a *tag* responsável por descrever o conteúdo completo do documento;

2. ***Types:*** define os tipos de dados utilizados pelo conjunto de operações fornecidas por um serviço Web;

3. ***portType:*** apresenta detalhes de cada operação contida no serviço, ou seja, descreve quais são os parâmetros de entrada de dados e qual será a saída produzida para cada uma das operações disponibilizadas pelo serviço Web. Em outras palavras, atua analogamente a uma interface utilizada na programação orientada a objetos;

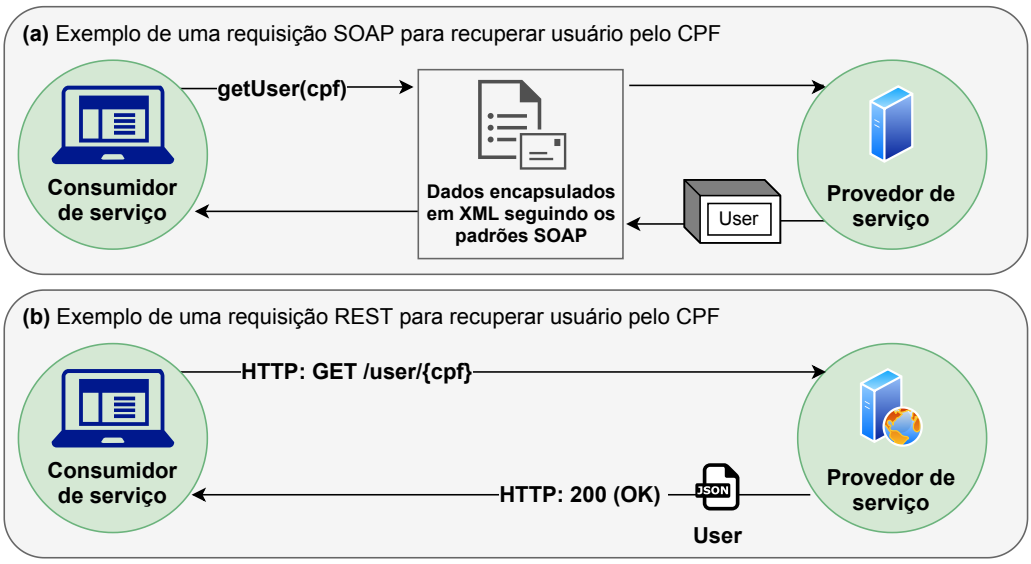
4. ***binding:*** responsável por definir qual o protocolo utilizado por cada operação descrita na *tag* ***portType*** e seus respectivos formatos de entrada e saída de dados; e

5. ***service:*** descreve a URL (do inglês, *Uniform Resource Locator*) do serviço, informando em qual *endpoint* um determinado serviço pode ser encontrado.

De acordo com Trends (2020), a utilização de serviços baseados em SOAP está decaindo com o tempo, dando lugar ao estilo arquitetural REST, o qual dá origem aos chamados *RESTful Web Services*. Esse estilo arquitetural consiste de um conjunto de princípios, regras e restrições que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas, possibilitando a comunicação entre diferentes aplicações. Os recursos da aplicação são disponibilizados diretamente através da URI (do inglês, *Uniform Resource Identifier*) do serviço e faz uso de métodos como `GET`, `POST`, `PUT`, `PATCH` e `DELETE`, os quais são providos pelo protocolo HTTP e aplicados na troca de informações entre cliente e servidor.

Embora seja possível encontrar comparações entre serviços SOAP e REST, talvez por ambos possuírem uma relação direta com SOA, é importante ressaltar que ambos são conceitualmente distintos. O primeiro trata de um protocolo de comunicação que possui um conjunto de regras bem definidas e segue uma abordagem do tipo *function-driven*, onde os dados são disponibilizados na forma de serviços e métodos (veja o método `getUser()`, ilustrado na Figura 6a). O segundo retrata um estilo arquitetural e, apesar de também possuir um conjunto de regras, proporciona ao desenvolvedor maior flexibilidade por não se tratar de um padrão a ser seguido. Além disso, esse estilo é orientado a uma abordagem do tipo *data-driven*, onde os dados são disponibilizados como recursos (veja a requisição do tipo GET localizada em uma determinada URI, ilustrada na Figura 6b) (MULLIGAN; GRACANIN, 2009; SOAPUI, 2018). A seguir, a Tabela 1 apresenta um comparativo entre algumas das características de SOAP e REST.

Figura 6 – Exemplo de requisições SOAP e REST para recuperar os dados de um determinado usuário



Fonte: Elaborada pelo autor

Tabela 1 – Comparação entre algumas das características de SOAP e REST

Característica	SOAP	REST
Padrão oficial definido	Sim	Não
Padrões comumente utilizados	HTTP e XML	HTTP, JSON, URL e XML
Método de exposição da lógica de negócio	Interfaces de serviços (*function-driven*)	Através de URLs (*data-driven*)
Linguagem para descrição de serviços	Sim (WSDL)	Não
Facilidade de integração com tecnologias Web como JavaScript	Passível de integração, porém, pode ser mais complexa	Simplificada
Tratamento nativo de erros	Sim	Não
Consumo de banda	Potencialmente elevado (XML)	Potencialmente baixo (JSON)
Curva de aprendizado	Maior	Menor

Fonte: Smartbear (2013), Malyk (2017)

Diante do exposto, uma arquitetura orientada a serviços pode ser definida como um modelo arquitetural formado por um conjunto de regras para o projeto e implementação de sistemas, cujo objetivo é integrar as aplicações, prover maior flexibilidade para mudanças e suportar serviços independentemente da plataforma e dos protocolos utilizados. Esse tipo de arquitetura é geralmente implementada através do uso de serviços Web, os quais representam as maneiras como tais regras são disponibilizadas. Sendo assim, arquiteturas orientadas a serviços especificam como deve ocorrer a comunicação entre diferentes aplicações, permitindo a criação de serviços mais complexos a partir de outros mais simples.

2.3 Qualidade de Serviço

A Associação Brasileira de Normas e Técnicas (ABNT) apresenta a seguinte definição para o termo “qualidade”:

> “A qualidade dos produtos e serviços de uma organização é determinada pela capacidade de satisfazer os clientes e pelo impacto pretendido e não pretendido nas partes interessadas pertinentes.
>
> A qualidade dos produtos e serviços inclui não apenas sua função e desempenho pretendidos, mas também seu valor percebido e o benefício para o cliente.” (ABNT, 2015)

O termo "Qualidade de Serviço" pode ser encontrado em diferentes partes da literatura e suas definições costumam variar de acordo com a área em que o termo é empregado. Originado a partir da área de telecomunicações, foi definido pela ITU (do inglês, *International Telecommunication Union*) como sendo a "Totalidade das características de um serviço de telecomunicações que define sua capacidade de satisfazer as necessidades declaradas e implícitas do usuário do serviço" (ITU (2008), tradução nossa). Com a evolução das tecnologias e das pesquisas voltadas para serviços Web, esse termo passou a ser utilizado na concepção de aplicações orientadas a serviços como um atributo não funcional.

Baseado na definição apresentada nesta seção, para Kritikos e Plexousakis (2009), QoS de um serviço Web refere-se a um conjunto não funcional de atributos aplicados na conexão entre o servidor que provê o serviço e o cliente que o consome, determinando sua capacidade de satisfazer as necessidades do cliente. Esses valores podem mudar durante seu ciclo de vida e, desde que tais mudanças não impactem nas funcionalidades críticas do serviço e se mantenham dentro de um intervalo de aceitação pré-definido, pode-se dizer que um serviço atende a um certo nível de QoS.

Segundo Deora *et al.* (2003), existem diferentes visões para o termo qualidade, originadas a partir de diferentes estudos, sendo as mais comuns as seguintes:

1. **Qualidade como funcionalidade**: considerada como sendo o conjunto de funcionalidades que o serviço pode oferecer para seus clientes. Por exemplo, considerando um cenário de planejamento de viagem envolvendo dois serviços: o primeiro serviço (`S1`) oferece aos usuários a possibilidade de compra de passagem aérea, reserva de hotel e aluguel de carro. O segundo serviço (`S2`) oferece apenas a possibilidade de compra de passagem aérea. Segundo essa visão de qualidade, pode-se dizer que o serviço `S1` possui maior nível de qualidade do que `S2`;

2. **Qualidade como conformidade**: vista como uma forma de atender determinadas especificações. Para exemplificar essa definição, o serviço `S1` da primeira definição é considerado. Supondo que esse serviço tenha que cumprir um contrato de qualidade de serviço (do inglês, *Service Level Agreement* – SLA) e isso se prove verdadeiro durante sua execução, pode-se dizer então que esse serviço apresenta um grau positivo de qualidade. O cenário ilustrado retrata a visão mais comumente avaliada e encontrada durante o mapeamento sistemático conduzido, o qual será detalhado no Capítulo 5.

3. **Qualidade como reputação**: representa uma visão do usuário em relação ao serviço, retratando o pensamento do usuário sobre esse serviço em relação a atributos como disponibilidade, resultados esperados, entre outros. Por exemplo, caso determinado serviço prometa oferecer determinada funcionalidade com um certo grau de performance e, durante

sua operação, isso se prove verdadeiro para os usuários que o utilizam, pode-se dizer que o critério de qualidade estipulado foi atendido.

A seguir, na Tabela 2, é apresentada uma lista contendo alguns atributos de QoS e suas respectivas definições. Esses atributos atendem a visão de qualidade como conformidade e, em alguns casos, qualidade como reputação. Essa lista foi compilada a partir de alguns dos atributos mais recorrentes identificados no mapeamento sistemático conduzido. As definições para cada atributo foram obtidas através do Vocabulário de Sistemas e Engenharia de software (do inglês, *Software and Systems Engineering Vocabulary* – SEVOCAB) (IEEE Computer Society, 2019) e em alguns casos diretamente do estudo responsável por mencioná-las. Para não destoar o conteúdo deste capítulo, optou-se por não mostrar nesse momento o conjunto de referências bibliográficas com a lista de estudos em que cada atributo é mencionado.

Tabela 2 – Conjunto com alguns dos atributos de qualidade de serviço mais comuns e suas definições

Atributo	Definição
Disponibilidade	Habilidade de um sistema ou componente de se manter operacional e acessível para uso de maneira que seja capaz de realizar uma determinada função quando requerido.
Latência	Intervalo de tempo decorrido durante a comunicação de uma mensagem, ou seja, tempo que uma informação leva para ser transmitida entre o requisitor e o requisitante.
Tempo de execução	Intervalo de tempo decorrido entre a submissão de uma tarefa e sua conclusão.
Tempo de resposta	Intervalo de tempo decorrido entre início de uma requisição e sua resposta, ou seja, representa uma soma entre a transmissão da mensagem (latência) e o necessário para o processamento da tarefa (tempo de execução).
Custo	Preço (valor monetário) cobrado em virtude de uma requisição a um determinado serviço.
Confiabilidade	Habilidade de um sistema ou componente de realizar as tarefas as quais se propõe dentro de um conjunto de condições pré-estabelecidas por um certo período de tempo.

Continua na próxima página

Continuando da página anterior

Atributo	Definição
Performance	Capacidade de um sistema ou componente de realizar as tarefas as quais se propõe com base em um conjunto de requisitos (por exemplo, velocidade, acurácia ou consumo de memória).
Carga (CPU/memória)	Quantidade de recursos (processamento e memória) utilizados por um determinado sistema ou componente.
Vazão	Quantidade de trabalho que pode ser realizado por um sistema ou componente dentro de um determinado período de tempo.
Taxa de falhas	Número de falhas de uma certa categoria que ocorrem em um determinado sistema ou componente durante uma certa unidade de medida (por exemplo, tempo transcorrido, quantidade de transações, entre outros).
Largura de banda	**(i)** Quantidade de tráfego mínimo necessário para o bom funcionamento de um sistema ou componente; **(ii)** Quantidade de tráfego utilizado por um sistema ou componente durante o desempenho de suas tarefas.
Correção	Grau que determina o quanto um sistema ou componente está livre de falhas em sua especificação, *design* e implementação.
Segurança	Capacidade do sistema de se proteger de ser acessado, utilizado, modificado ou destruído por usuários maliciosos.
Consumo de energia	Quantidade de energia de um dispositivo que um sistema ou componente utiliza para cumprir uma determina tarefa.

Fonte: Elaborada pelo autor

2.4 Considerações finais

Neste capítulo foram apresentados conceitos relacionados a área de COS, utilizados no desenvolvimento deste trabalho. Inicialmente foi apresentada uma definição para serviço Web e também dos termos composição e orquestração de serviços. Em seguida, foi necessário definir o que é uma arquitetura orientada a serviços, adotada neste trabalho como principal método de troca de dados entre cliente e provedor de serviços. Finalmente, foi apresentada uma definição para o termo “Qualidade de Serviço”, utilizado no contexto deste trabalho especialmente para as tarefas de busca e seleção de serviços.

3 Sistemas Autoadaptativos

Segundo Garlan e Schmerl (2002), as pesquisas realizadas na área de engenharia de software tradicionalmente consideram que a criação e modificação de módulos de sistemas ocorrem sempre em situações em que o sistema é desligado. Porém, no cenário atual, muitos sistemas requerem disponibilidade constante, de maneira que manutenções e/ou modificações devem ser realizadas com o sistema em execução. Além disso, falhas devem ser corrigidas de maneira rápida, eficiente e confiável. Não apenas isso, Williams e Carver (2010) defendem que um sistema de software inevitavelmente sofrerá mudanças durante seu ciclo de vida. Essas mudanças são motivadas principalmente pela necessidade do software de acompanhar a evolução dos requisitos dos clientes que o utiliza e também pela necessidade de se realizar adaptações no sistema para que o mesmo acompanhe as evoluções tecnológicas. Nesse sentido, é necessário elaborar técnicas e mecanismos que permitam o desenvolvimento de sistemas capazes de adaptar seu comportamento em tempo de execução de maneira confiável e eficiente. Esse recurso, disponibilizado em algumas linguagens de programação como Java, é conhecido como Reflexão Computacional, ou simplesmente Reflexão, e permite que um sistema seja capaz de executar uma determinada atividade por conta própria, sendo muito parecido com a reflexão humana, a qual possui a capacidade de ter consciência sobre suas ações (CÁMARA *et al.*, 2017). Seu principal objetivo está em obter informações a respeito das atividades do sistema com o intuito de melhorar seu desempenho, introduzir novas competências ou até mesmo ser capaz de resolver problemas a partir de um procedimento que seja mais adequado a uma situação específica, provendo a capacidade de gerenciar sua complexidade e, consequentemente, tornar-se mais flexível à modificações diante de eventuais problemas que possam surgir em tempo de execução (FORMAN; FORMAN, 2004; VINOSKI, 2005; SALEHIE; TAHVILDARI, 2009).

O objetivo deste capítulo é apresentar uma contextualização sobre sistemas autoadaptativos (do inglês, *Self-adaptive Systems* – SaS). Para isso, inicialmente a Seção 3.1 reporta algumas definições sobre SaS e suas características. A Seção 3.2 detalha o modelo MAPE-K, proposto pela IBM em 2005 e frequentemente utilizado no planejamento de todo o processo de adaptação utilizado por uma aplicação. Em seguida, na Seção 3.3, é apresentado um conjunto de propriedades tradicionalmente associadas com a área de SaS. Por fim, a Seção 3.4 reporta os principais desafios e limitações encontrados na literatura relacionados ao desenvolvimento e adoção de SaS.

3.1 Definição de sistemas autoadaptativos

É de se esperar que sistemas de software continuem evoluindo, como se pôde observar nas últimas décadas, em especial em características como tamanho e complexidade. Paralelamente à criação e adoção de novas técnicas de desenvolvimento de sistemas modernos, surge a necessidade de se elaborar mecanismos que permitam automatizar e simplificar o gerenciamento e modificação desses sistemas mesmo após sua implantação, em tempo de execução. Em 2005 foi publicado pela IBM um relatório técnico que dizia que a lei de Moore[1] não era mais considerada como o maior obstáculo na progressão da indústria da tecnologia. Os avanços proporcionados por essa lei trouxeram uma nova preocupação, agora com relação a complexidade dos produtos originados a partir da tecnologia disponível, iniciando discussões que resultaram na área de SaS. Isso foi reforçado por Esfahani, Elkhodary e Malek (2013), onde os autores argumentam que a partir dessa necessidade de automatizar e simplificar o gerenciamento e modificação de sistemas surgiram os sistemas de software autoadaptativos. Segundo Farahani *et al.* (2016), o objetivo desses sistemas é reduzir a necessidade de interferência humana, tornando-os autônomos. Em virtude disso, após a implementação desses sistemas, é necessário avaliá-los com o intuito de analisar sua eficiência e eficácia.

Embora não exista um consenso ao estabelecer uma definição para esse tipo de sistema, Cheng *et al.* (2009 apud IBM, 2005) refere-se a SaS como sendo capazes de ajustarem seu comportamento em resposta a uma percepção do ambiente e/ou de si mesmos. Brun *et al.* (2009 apud IBM, 2005) complementa essa definição ao ressaltar que o prefixo “auto” indica a capacidade do sistema em decidir sem ou com pouca interferência externa como se adaptar e se organizar de maneira que seja possível atender as mudanças de contexto e ambiente. Mahdavi-Hezavehi *et al.* (2017) definem SaS como sendo capazes de modificar sua estrutura ou comportamento por vários motivos, especialmente em situações identificáveis apenas em tempo de execução. São exemplos desses cenários: (i) mudanças no ambiente em que o sistema se encontra; (ii) falhas a nível de sistema; (iii) alterações em atributos de qualidade como performance ou confiabilidade; e (iv) a necessidade de se adaptar a novos requisitos que possam surgir. Para isso, um sistema autoadaptativo permanece em monitoramento constante, coletando dados sobre si e sobre o ambiente ao qual foi inserido, analisando-os e decidindo se uma adaptação é necessária ou não. Em casos em que se prove necessário, executar um procedimento de adaptação é uma atividade “comum” para esses sistemas. O grande desafio no desenvolvimento desse tipo de sistema está justamente em identificar o conjunto de modificações que devem ser realizadas em determinados cenários, mantendo a aplicação dentro de um certo nível da qualidade pré-estabelecido durante sua concepção.

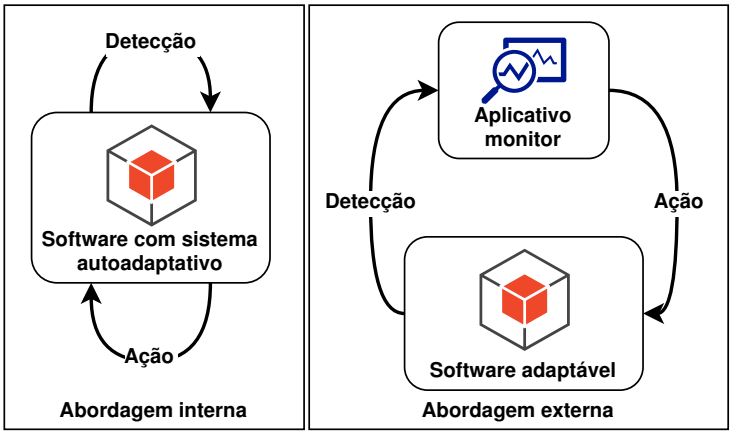
[1] Previsão feita em 1985 por Gordon Moore, cofundador da Intel, o qual dizia que a densidade de transistores nos *chips* utilizados por computadores dobra a cada 18 meses.

De acordo com Salehie e Tahvildari (2009), uma adaptação pode ser implementada através de duas abordagens, relacionadas ao mecanismo de adaptação e a lógica da aplicação. Tais abordagens, ilustradas na Figura 7, são descritas a seguir:

1. **Interna:** neste tipo de abordagem o aplicativo monitor, responsável por verificar o estado do software, está contido na aplicação, o que geralmente resulta em baixa escalabilidade e manutenibilidade do sistema de software; e

2. **Externa:** neste tipo de abordagem o sistema está organizado em duas camadas bem definidas: (i) camada do aplicativo monitor, que é responsável pelo monitoramento de um sistema e; (ii) camada de software adaptável, que representa o sistema de software que é monitorado. Portanto, por meio dessa abordagem, o sistema autoadaptativo consiste de um módulo de adaptação, responsável por monitorar o software e realizar as adaptações necessárias. Além disso, nesse tipo de organização, a existência de arquiteturas e infraestruturas confiáveis é um elemento essencial.

Figura 7 – Tipos de abordagem de adaptação



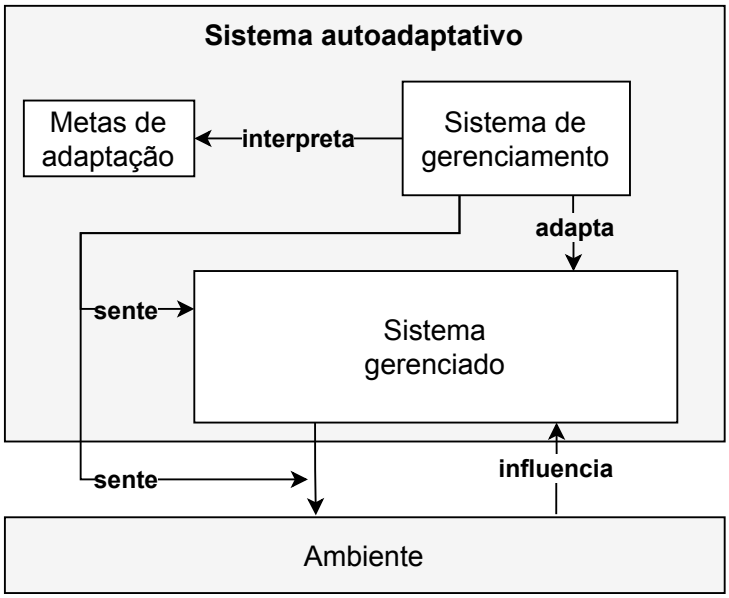
Fonte: adaptado de Salehie e Tahvildari (2009)

Weyns (2018) apresenta um modelo conceitual genérico de um sistema autoadaptativo, o qual descreve um conjunto de conceitos e suas relações, como mostra a Figura 8. Esse modelo é composto de quatro elementos básicos, a saber:

1. **Ambiente:** se refere a parte do mundo externo com a qual o sistema autoadaptativo interage e quais efeitos desse sistema serão observados e avaliados. Pode abranger tanto objetos do mundo real quanto do mundo virtual;

2. **Sistema gerenciado:** representa a aplicação que está sendo monitorada e que sofrerá as adaptações que o sistema julgar necessário executar;

Figura 8 – Modelo conceitual de um sistema autoadaptativo



Fonte: adaptado de Weyns (2018)

3. **Metas de adaptação:** tratam-se das metas que o sistema autoadaptativo objetiva cumprir, geralmente apresentando relação com alguma métrica de avaliação de qualidade (por exemplo, latência de um serviço); e

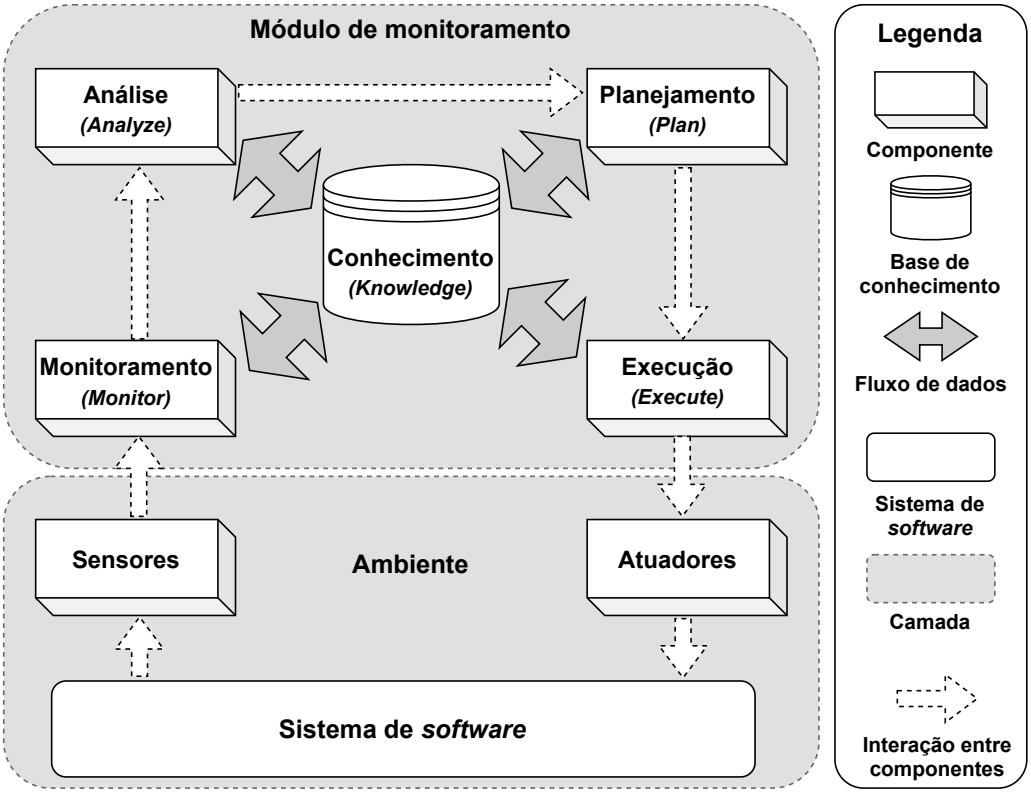
4. **Sistema de gerenciamento:** trata-se da aplicação que está responsável pela tarefa de monitoramento e adaptação. Esse elemento do sistema contém a lógica de adaptação que será utilizada tanto no monitoramento quanto na tomada de decisão e execução de tarefas, objetivando manter as metas de adaptação sempre atendidas.

3.2 O modelo MAPE-K

Para lidar com a tarefa da autoadaptação, uma das maneiras mais utilizadas é a adoção do modelo MAPE-K, proposto em 2005 pela IBM (2005). Esse modelo se propõe a automatizar a tarefa de gerenciamento de um ou mais componentes de um sistema, seguindo o fluxo ilustrado pela Figura 9. Um conjunto de **sensores** coleta informações do sistema de software e/ou do ambiente de execução ao qual está inserido, enviando-as para o componente de monitoramento. Em seguida, o **monitor** provê mecanismos para coletar, unir, filtrar e reportar as informações recebidas, transmitindo-as para o componente de **análise**, o qual é responsável por estabelecer relações entre os dados recebidos com o intuito de prever situações que possam ocorrer futuramente. Uma vez identificados os possíveis cenários futuros, essas informações são transmitidas para o componente de **planejamento**, cujo propósito é especificar quais ações devem ser tomadas para atingir ou manter um conjunto de objetivos determinados em alguma fase do ciclo de

vida do sistema (ou seja, *design* ou *runtime*). Por fim, esse planejamento é transmitido para o componente de **execução**, o qual, por meio dos **atuadores**, irá prover os meios para cumprir o planejamento elaborado, executando-o no sistema de software. Todas essas etapas acontecem dentro de um laço, executado sem interrupção durante todo o ciclo de vida do sistema.

Figura 9 – Modelo de referência MAPE-K



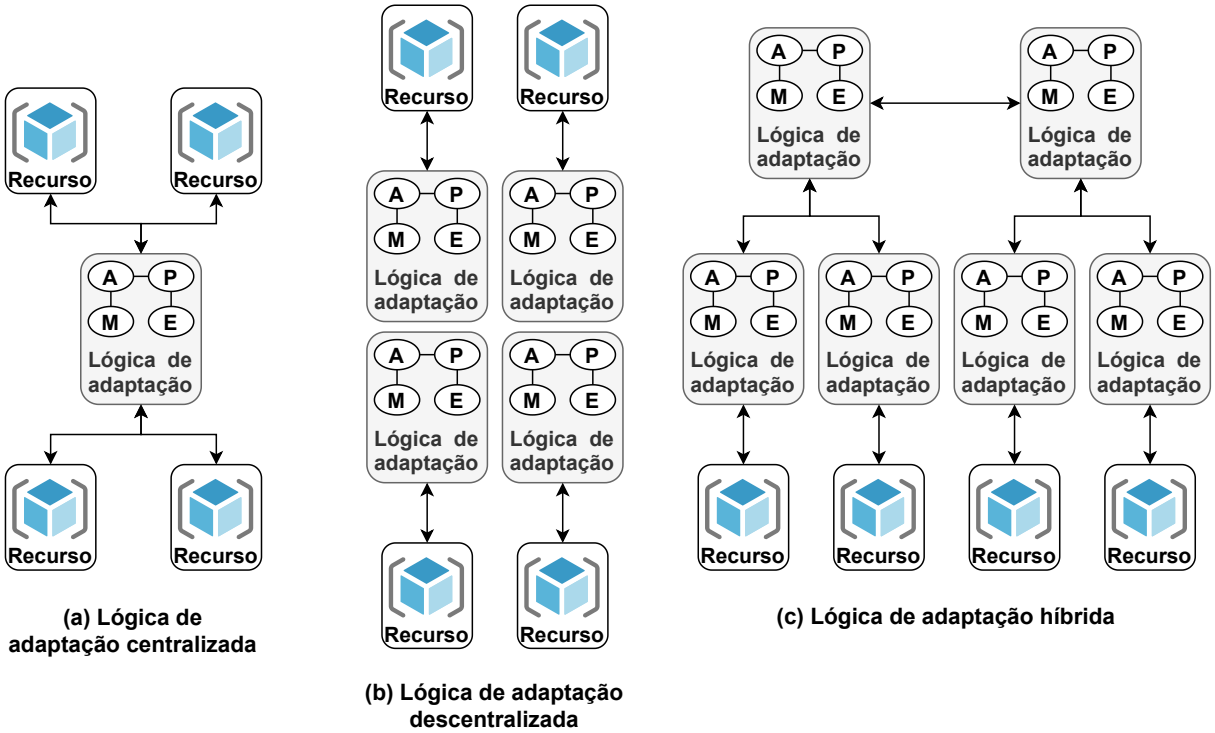
Fonte: adaptado de Mahdavi-Hezavehi *et al.* (2017)

O modelo apresentado na Figura 9 também dispõe de uma **base de conhecimento**, que visa compartilhar todo seu conhecimento sobre o sistema com todos os componentes do módulo de monitoramento. Esse conhecimento pode ser previamente especificado e informado durante a etapa de elaboração do sistema, descrevendo aspectos como configurações de execução e parâmetros de métricas de QoS. Em outro cenário, esse conhecimento pode ser obtido durante o ciclo de vida da aplicação, reportando medições obtidas para determinadas métricas de QoS, conjunto de *logs* gerados, entre outros.

Krupitzer *et al.* (2015) argumenta que os principais questionamentos sobre a lógica de adaptação e recursos estão relacionados a estrutura e/ou organização dos componentes, como ilustra a Figura 10. Esse autor surge três possíveis abordagens para organizar a lógica de adaptação, a saber:

1. **[Figura 10a] Centralizada:** nesta abordagem a lógica de adaptação é totalmente centralizada, sendo responsável por monitorar o ambiente e gerenciar todos os recursos da

Figura 10 – Possíveis abordagens para a lógica de adaptação MAPE-K



Fonte: adaptado de Krupitzer *et al.* (2015)

aplicação. Dessa forma, obtém-se a maior eficiência possível nas tarefas de monitoramento e gerenciamento, já que toda a informação se concentra em apenas uma instância da lógica de adaptação. Entretanto, sistemas de grande porte podem não se beneficiar desse tipo de abordagem, uma vez que o tamanho do sistema irá influenciar diretamente na quantidade de informação coletada e, consequentemente, pode exigir um alto poder computacional de processamento;

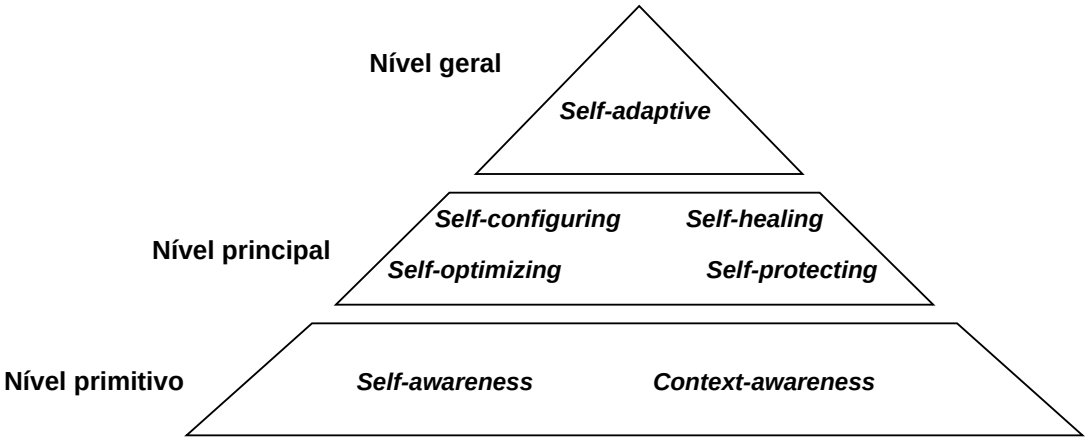
2. **[Figura 10b] Descentralizada:** nesta abordagem a lógica de adaptação é totalmente descentralizada, garantindo que cada um dos recursos da aplicação tenha sua própria lógica de aplicação, sendo todos os recursos do sistema monitorados e gerenciados individualmente. De maneira complementar, os diferentes subsistemas originados a partir dessa abordagem podem estabelecer um canal de comunicação com o intuito de atingir um objetivo mais geral; e

3. **[Figura 10c] Híbrida:** esta abordagem é a combinação das duas anteriores. Sendo assim, cada um dos recursos da aplicação possui sua própria lógica de aplicação para as tarefas de monitoramento e gerenciamento. Essas tarefas são coordenadas por uma ou mais lógicas de aplicação hierarquicamente superiores, as quais são responsáveis por atuar de maneira similar a orquestradores, auxiliando no processo de tomada de decisão.

3.3 Características e propriedades de sistemas autoadaptativos

Para Kramer e Magee (2007), SaS apresentam as seguintes características: escalabilidade, suporte para composição dinâmica, flexibilidade e robustez na ocorrência de mudanças. Esses sistemas são capazes de desempenhar tarefas de autoconfiguração, autoadaptação, autocura, automonitoramento, autoajuste, entre outras. Essa mesma ideia pode ser aplicada na arquitetura de software desses sistemas, resultando em uma importante área de pesquisa conhecida como arquiteturas de software dinâmicas (KRAMER; MAGEE, 2007; OQUENDO, 2008; SALEHIE; TAHVILDARI, 2009). O termo autoadaptativo é muitas vezes utilizado de maneira genérica para se referir a um tipo de sistema capaz de realizar algum tipo de adaptação em tempo de execução. Trata-se, portanto, de um termo mais abrangente que engloba várias outras propriedades relacionadas ao tipo de adaptação envolvida, também referidas como propriedades *self-**. A Figura 11 mostra uma hierarquia em pirâmide que organiza as propriedades *self-** em três diferentes níveis, de modo que seu topo representa propriedades mais abrangentes e sua base retrata propriedades mais primitivas. A seguir são apresentados detalhes de cada um dos níveis envolvidos nessa divisão hierárquica.

Figura 11 – Visão hierárquica de algumas das propriedades *self-** mais comuns



Fonte: adaptado de Salehie e Tahvildari (2009)

Nível geral. Este nível representa a propriedade de autoadaptação mais abrangente (*self-adaptive*), a qual é utilizada para indicar que o sistema é capaz de executar algum grau de adaptação sem especificar seu tipo;

Nível principal. Este nível representa o conjunto de propriedades possivelmente envolvidas quando se menciona um sistema autoadaptativo. Para Salehie e Tahvildari (2009), esse conjunto de propriedades costuma apresentar uma certa relação de similaridade com

mecanismos de adaptação de sistemas biológicos. Nesse sentido, pode-se citar como exemplo o corpo humano e sua capacidade de perceber e reagir em relação a algum tipo de mudanças de contexto, tais como: temperatura, som, cheiro, entre outras. Além disso, pode-se citar também as mudanças que refletem diretamente em seu estado de saúde, como um ferimento ou até mesmo uma fratura. As propriedades desse nível podem ser compostas a partir de outras propriedades mais primitivas, resultando em uma mais específica para a realização de uma determinada tarefa. A seguir é apresentada uma breve descrição para cada propriedade desse nível:

1. ***Self-configuring*:** refere-se à capacidade de **autoconfiguração** automática do sistema, permitindo alterar sua configuração automaticamente, reagindo dinamicamente a mudanças e atuando através da instalação, atualização, integração e composição/-decomposição de componentes de software;
2. ***Self-healing*:** refere-se à capacidade de **autocura** do sistema, sendo relacionada com sua aptidão em descobrir, diagnosticar e reagir a cenários que envolvam algum tipo de interrupção. Essa propriedade está também relacionada com sua capacidade em prever possíveis problemas futuros e reagir com a finalidade de evitar sua ocorrência. As características que a descrevem geralmente estão associadas diretamente com duas outras propriedades *self-**: (i) *self-diagnosing*, capacidade do sistema em **diagnosticar** erros e falhas; e (ii) *self-repairing*, habilidade de executar **reparos** e recuperar-se de um cenário de falha;
3. ***Self-optimizing*:** termo também referido na literatura por *self-tuning* ou *self-adjusting*, refere-se à capacidade de gerenciar recursos e de que maneira são alocados. Consequentemente, exerce um impacto direto na performance do sistema, tendo como objetivo garantir que os requisitos dos usuários sejam cumpridos. Atributos de QoS que envolvam métricas quantitativas (por exemplo, tempo de resposta e carga de trabalho) são exemplos de preocupações dessa propriedade;
4. ***Self-protecting*:** termo relacionado à capacidade do sistema em detectar possíveis falhas de segurança e recuperar-se de seus efeitos. Atua com o principal objetivo de defender o sistema de ataques maliciosos, preocupando-se em antecipar possíveis problemas e tomar ações para evitar sua ocorrência ou mitigar seus efeitos.

Nível primitivo. Este nível representa propriedades mais primitivas, geralmente associadas com a noção de percepção do sistema em relação ao ambiente em que está inserido (*context-awareness*) ou a si mesmo (*self-awareness*). Essa ideia de percepção se baseia nas pesquisas da área de computação autoconsciente. De acordo com Cámara *et al.* (2017), a construção de sistemas para computação autoconsciente tem sido possível graças aos esforços em diferentes áreas, como por exemplo: (i) inteligência artificial; (ii) computação

autônoma; (iii) sistemas autoadaptativos ou auto-organizáveis; e (iv) computação cognitiva. Para os autores, a visão de inteligência artificial que mais se aproxima da proposta de sistemas autoconscientes é a adotada em Russell e Norvig (2009). Nessa visão, o conceito de inteligência artificial refere-se à capacidade de projetar e criar sistemas de agentes racionais. Esses agentes obedecem a um conjunto de quatro propriedades, a saber: (i) autonomia, indicando sua capacidade de operar sem a necessidade de intervenção humana; (ii) habilidade social, indicando sua capacidade de interagir com outros agentes ou humanos através de uma linguagem de comunicação de agentes; (iii) reatividade, a qual se refere a capacidade desses agentes de perceber e responder a mudanças ocorridas no ambiente que os envolve; e (iv) proatividade, a qual indica que os agentes não são dotados apenas da capacidade de responder a eventos, mas sim de traçar objetivos que desejam atingir. De acordo com Narendra e Orriens (2017), um sistema de computação autoconsciente deve ser capaz de:

1. **Aprender modelos** continuamente através da coleta de **conhecimento** sobre si e o seu ambiente, determinando sua estrutura, estado, quais as possíveis ações que podem ser tomadas e seu comportamento em tempo de execução; e
2. **"Raciocinar"** através desses modelos (ou seja, prever, analisar, considerar e planejar) de maneira que **atuem** com base em seu conhecimento atual e **raciocinem** (ou seja, explorem, expliquem, reportem, sugiram, adaptem-se ou impactem o ambiente em que estão) de acordo com um conjunto de **regras**, as quais também estão sujeitas a alterações.

Baseado nas perspectivas apresentadas por Cámara *et al.* (2017) e Narendra e Orriens (2017), pode-se concluir que a computação autoconsciente se refere a um tipo de sistema de agentes ou multiagentes autônomos, dotados de capacidades de interação, reação e proativos. Essas capacidades são satisfeitas através da adoção de abordagens de aprendizado de modelos, os quais são utilizados com a finalidade de atingir os objetivos desses sistemas.

De acordo com Salehie e Tahvildari (2009), ao aproveitar-se das características de sistemas de computação autoconsciente e de outras propriedades *self-**, um novo tipo de sistema, denominado como autoadaptativo, pode ser criado. Esse sistema deve ser capaz de observar as mudanças ocorridas em si mesmo e/ou em seu ambiente de execução para que possa reagir adequadamente, conforme o conjunto de propriedades implementado. Para viabilizar a capacidade de autoadaptação, o modelo 5W1H[2] tem sido utilizado para determinar os requisitos essenciais desse tipo de sistema por meio de um conjunto de seis perguntas, a saber:

[2] Seis perguntas: O que (*What*), Onde (*Where*), Quem (*Who*), Quando (*When*), Porque (*Why*) e Como (*How*)

1. **O que** (*What*). O objetivo deste tipo de pergunta é identificar quais partes do sistema são alvo de adaptação;

2. **Onde** (*Where*). O objetivo deste tipo de pergunta é identificar quais partes do sistema estarão diretamente envolvidas com a adaptação, além do nível de grandeza da mesma;

3. **Quem** (*Who*). O objetivo deste tipo de pergunta é identificar quem será o agente responsável pela modificação. Por tratar-se de um método de evolução autônoma, espera-se que o nível de interação humana seja o menor possível;

4. **Quando** (*When*). A partir desse questionamento é possível definir em que situações uma adaptação deve ser realizada, além dos cenários que possibilitam tal adaptação. Em outras palavras, o objetivo deste tipo de questão é definir se uma modificação pode ser realizada a qualquer momento ou se existem limitações que devem ser respeitadas antes de um evento de adaptação;

5. **Porque** (*Why*). O objetivo deste tipo de pergunta é entender a motivação por trás de uma adaptação. Esse questionamento pode resultar na descrição das necessidades de se conduzir um evento de adaptação em uma aplicação; e

6. **Como** (*How*). O objetivo deste tipo de pergunta é identificar os detalhes sobre quais partes do sistema serão afetadas durante um evento de adaptação, além das ações que devem ser realizadas para garantir que a aplicação continue em funcionamento.

Segundo Salehie e Tahvildari (2009), para a concepção de SaS é necessário que as questões do modelo 5W1H sejam respondidas em duas etapas: projeto e operação. Na primeira, o projetista da aplicação deve direcionar seus esforços para o projeto da estrutura de adaptação. Já na segunda, durante o ciclo de vida da aplicação, momento em que os procedimentos de adaptação planejados são continuamente testados e executados quando necessário.

3.4 Desafios e limitações envolvidos na adoção de sistemas autoadaptativos

Embora a adoção de SaS traga muitos benefícios, a implementação desse tipo de sistema não é uma tarefa trivial. Na literatura é possível encontrar diversas iniciativas que abordam o desenvolvimento de SaS em diferentes domínios de sistemas, a saber: sistemas de informação, sistemas orientados a serviços, sistemas críticos, sistemas móveis, entre outros. Essas iniciativas podem ser agrupadas através da proposição de abordagens, metodologias, métodos, *frameworks*, recursos de infraestrutura e revisões formais da literatura. De modo geral, tais iniciativas também

apresentam os assuntos em aberto relativos ao tema da pesquisa abordado, além de nortear a comunidade científica para importantes desafios, limitações e tendências (MUÑOZ-FERNÁNDEZ *et al.*, 2018; WEYNS *et al.*, 2012; CHEN; BAHSOON; YAO, 2018; SALEHIE; TAHVILDARI, 2009).

Uma das questões de pesquisa utilizadas no mapeamento sistemático conduzido por Weyns *et al.* (2012) investiga quais são os benefícios mais apontados por estudos nessas áreas e também seus *trade-offs* (ou seja, perdas e ganhos). Segundo os autores, dos 30 estudos envolvidos, aproximadamente 94% relatam características relacionadas com atributos de qualidade de serviço. Os três atributos mais recorrentes foram: (i) flexibilidade; (ii) confiabilidade; e (iii) eficiência/performance. Outros atributos como, por exemplo, disponibilidade, precisão, usabilidade e segurança também foram mencionados, mas em menor número, representando juntos uma pequena porcentagem dos estudos. Os outros 6% dos estudos reportam características como complexidade do software, esforço envolvido em seu desenvolvimento e custo orçamental. Já com relação aos *trade-offs*, poucos estudos se dispõem a apresentá-los ou propor uma discussão aprofundada sobre o assunto, sendo possível encontrar em alguns dos estudos preocupações relacionadas principalmente a performance ou aos atributos fora do escopo de qualidade, como por exemplo, as preocupações relacionados ao custo de desenvolvimento desse tipo de aplicação.

Chen, Bahsoon e Yao (2018) e Muñoz-Fernández *et al.* (2018) reportaram alguns dos problemas em aberto e os desafios envolvidos na pesquisa e desenvolvimento de SaS. A seguir, alguns dos pontos destacados pelos autores são apresentados:

Representações explícitas de conhecimento são necessárias em SaS. Embora a maior parte dos estudos faça referência a necessidade de se estabelecer algum nível de representação do conhecimento, é difícil definir exatamente como esse conhecimento é representado, principalmente em domínios de aplicação que envolvam tarefas mais complexas. A maior parte dos estudos se limita a discutir sistemas dotados de uma capacidade básica de autoconsciência, mas nem sempre apresentam uma discussão mais profunda sobre o tema. O desafio envolvido nesse ponto está em estabelecer um método que possibilite distinguir os diferentes níveis de conhecimento que podem ser adotados e como podem ser integrados nesse tipo de aplicação (CHEN; BAHSOON; YAO, 2018);

A adoção de múltiplos laços de controle pode beneficiar os SaS. Segundo Chen, Bahsoon e Yao (2018), a maior parte dos estudos selecionados descreve a utilização de um único laço de controle, gerando um problema de alto acoplamento entre diferentes partes da arquitetura. Uma solução para esse problema é adoção de múltiplos laços de controle para diferentes aspectos da arquitetura, obtendo diferentes níveis de adaptação para os diferentes tipos de adaptação envolvidos. Entretanto, determinar uma quantidade de níveis de adaptação e estabelecer como serão estabelecidos pode provar-se uma atividade desafia-

dora. Os autores sugeriram estabelecer uma diferenciação entre cada um desses níveis a partir do tipo de conhecimento necessário para cada laço de controle;

É necessário medir o impacto que as etapas de modelagem e adaptação causarão no nível de QoS da aplicação. Os resultados apresentados por Chen, Bahsoon e Yao (2018) mostram que o processo de decisão envolvido na etapa de adaptação pode causar impactos nos atributos de qualidade da aplicação. O planejamento de uma adaptação, desconsiderando fatores que determinam o nível de qualidade da aplicação, pode resultar em decisões incorretas referentes a aspectos como escalabilidade da aplicação;

Necessidade de se formalizar uma notação para especificar requisitos. A falta de uma notação padrão para a especificação de requisitos de software em arquiteturas de SaS pode se tornar um problema, pois pode gerar ambiguidade ao permitir definir o mesmo conceito ou requisito de diferentes maneiras. Apesar de ser possível contornar essa limitação através do uso de comentários e anotações que possam ser compreendidas por outras pessoas, nem sempre é possível descrever esses requisitos com total exatidão. Por exemplo, em um sistema de carro autônomo, cuja complexidade é elevada, uma parte desses comentários e/ou anotações pode ser mal interpretada ou até mesmo ignorada, podendo gerar efeitos indesejados durante o ciclo de vida da aplicação (MUÑOZ-FERNÁNDEZ *et al.*, 2018); e

Tamanho dos modelos de adaptação e das bases de conhecimento. É necessário buscar estratégias que permitam reduzir a complexidade relacionada à avaliação dos modelos envolvidos em uma tarefa de adaptação, assim como os dados disponibilizados pela base de conhecimento. Esse aspecto é uma preocupação dos desenvolvedores devido ao custo computacional envolvido durante a realização de uma busca por esses modelos. O resultado deve ser obtido o mais rápido possível, preferencialmente sem erros, especialmente em domínios de aplicação que requerem decisões rápidas como parte de seu requisito. Por exemplo, um sistema de carro autônomo representa um exemplo de aplicação desse domínio, já que em casos de falha, o sistema deve se recuperar da maneira mais rápida possível, minimizando a possibilidade de que um acidente ocorra (MUÑOZ-FERNÁNDEZ *et al.*, 2018).

3.5 Considerações finais

Neste capítulo foram apresentados alguns dos conceitos de SaS, área de pesquisa de grande importância para os sistemas de software atuais, especialmente em virtude de seu crescimento em tamanho e complexidade. Inicialmente, na Seção 3.1, foi apresentada uma definição mais geral para esse tipo de sistema e algumas de suas características, focando principalmente em apresentar uma maneira de se atingir o proposto por essa área. Para isso, foi apresentado

o modelo de referência MAPE-K, comumente adotado em SaS para executar todas as etapas da adaptação, desde o monitoramento do sistema até a execução de uma determinada tarefa, reagindo às mudanças detectadas no ambiente ou na própria aplicação. Na Seção 3.3 foram apresentadas algumas das propriedades de autoadaptação mais comuns, descrevendo seus objetivos e em que se diferenciam. Por fim, na Seção 3.4 foram apresentadas algumas das preocupações envolvendo os desafios e limitações relacionados ao desenvolvimento de SaS, os quais foram obtidos por meio de análise de revisões da literatura direcionadas para esse tipo de arquitetura.

4 Ontologias e Web Semântica

Um dos grandes desafios ao manipular um conjunto de dados está no entendimento do tipo de representação desses dados e suas relações. Esse tipo de informação, conhecida por metadado, é responsável por representar algum tipo de conhecimento sobre um determinado conjunto de dados. Em um banco de dados tradicional, essa informação está representada implicitamente através das definições de tabelas e suas respectivas relações, descritas através de chaves estrangeiras ou até mesmo nas *queries* de consulta (SEGARAN; EVANS; TAYLOR, 2009). A própria documentação de um sistema de software pode ser citada como uma fonte de informação, mas em algumas ocasiões a mesma não está bem descrita, relatando informações incompletas ou conflitantes. Em alguns casos, uma documentação sequer é disponibilizada, muitas vezes porque a equipe responsável pela aplicação não levou em consideração a elaboração de uma documentação adequada durante as etapas de desenvolvimento. Nesse cenário, a única opção para os desenvolvedores interessados em entender sua lógica é analisar o código fonte do sistema de software para compreender suas estruturas, comportamentos e relações. Esse tipo de atividade pode ser categorizada em uma escala de complexidade que varia entre simples e complexa, dependendo do domínio, tamanho do sistema de software, entre outros fatores. Além disso, dependendo das necessidades e restrições dos interessados (por exemplo, tempo e custo), esse tipo de tarefa se torna inviável.

No contexto desse trabalho, localizar um serviço capaz de substituir um outro, em estado de degradação de QoS ou que tenha apresentado falha, pode ser considerada uma tarefa de grande importância. Como as operações disponibilizadas por serviços Web lidam com tarefas vinculadas a um contexto semântico, a adoção de uma representação semântica de dados pode ser benéfica. Em linhas gerais, essas tarefas descrevem a realização de uma determinada ação e produzem uma saída a partir de um determinado conjunto de dados de entrada. Sendo assim, o objetivo deste capítulo é apresentar uma introdução sobre ontologias e sua utilização na Web semântica. Inicialmente, na Seção 4.1, são apresentados alguns conceitos e definições relacionados a dados semânticos e as ontologias que os descrevem. Adicionalmente, são apresentados o *framework* RDF (do inglês, *Resource Description Framework*) e a linguagem SPARQL (do inglês, *SPARQL Protocol and RDF Query Language*), utilizados para representar e manipular dados semânticos, respectivamente. Por fim, na Seção 4.2 são apresentados alguns dos esforços encontrados na literatura relacionados ao desenvolvimento de ontologias para a representação de serviços, seguidos por uma breve definição da ontologia OWL-S (do inglês, *Web Ontology Language for Service*).

4.1 Representação e manipulação de dados semânticos

O termo ontologia, inicialmente introduzido pela filosofia como uma maneira de estudar a natureza por trás de tudo que existe, foi adotado na ciência da computação como uma alternativa para descrever conceitos do mundo real. De acordo com Sikos (2015), organizações, projetos de pesquisa, eventos históricos e pessoas são exemplos de entidades do mundo real que podem ser descritas por meio de um modelo de dados passível de interpretação contextual por computadores. O desenvolvimento de uma ontologia provê um vocabulário através do qual é possível representar um conhecimento sobre um determinado domínio. A partir desse vocabulário é possível especificar quais entidades podem ser representadas, como estas devem ser agrupadas e quais são os tipos de relacionamentos que as envolve (SEGARAN; EVANS; TAYLOR, 2009). Esse vocabulário representa um contrato social entre entidades provedoras e consumidoras de dados. Observe que, diferentemente de um modelo tradicional de representação de dados, uma ontologia tenta preencher a lacuna do contexto semântico. Em outras palavras, modelos tradicionais de representação de informação são capazes de armazenar dados de um determinado domínio, mas falham ao apresentar o contexto ao qual pertencem. Por exemplo, em um contexto que envolva informação semântica, a utilização de uma ontologia garante uma descrição do modelo utilizado para a representação dos dados.

O desenvolvimento de uma nova ontologia considera um conjunto de componentes, os quais são implementados através de linguagens próprias, como, por exemplo, a linguagem OWL. A seguir, é apresentada uma breve descrição de alguns dos componentes mais frequentes no desenvolvimento de uma ontologia:

1. **Classes:** representadas através de grupos abstratos, conjuntos ou coleções de objetos e seus tipos. Geralmente descrevem grupos ou outras classes, cujos membros compartilham propriedades em comum de maneira hierárquica. Por exemplo, no contexto de uma ontologia, uma empresa pode ser representada através de uma classe e possuir um conjunto de subclasses capazes de representar seus departamentos e funcionários;

2. **Indivíduos:** são instâncias ou objetos das classes que representam o domínio. Por exemplo, no contexto de uma empresa, cada um de seus funcionários é um indivíduo;

3. **Atributos:** tratam-se das propriedades, características e parâmetros utilizados para descrever indivíduos e classes. Por exemplo, nome e data de nascimento são atributos que poderiam ser utilizados na descrição de um indivíduo da classe `Pessoa`;

4. **Relações:** descrevem como classes e indivíduos se relacionam com outras classes e/ou indivíduos. Por exemplo, um indivíduo da classe `Empresa` pode ter uma relação com vários indivíduos da classe `Funcionario`;

5. **Restrições:** formaliza limitações e/ou intervalos de valores válidos para classes e indivíduos. Por exemplo, uma possível restrição pode ser aplicada ao atributo data de nascimento, o qual deve aceitar apenas valores numéricos em um determinado formato. Uma possível restrição referente a data de nascimento exemplificada é o formato `dd/mm/aaaa`, onde `dd` corresponde ao dia, `mm` ao mês, `aaaa` ao ano e a quantidade de letras nesse formato determina quantos números são necessários nesse campo;

6. **Regras:** estruturas condicionais (`se-então-senão`) utilizadas para definir inferência lógica[1]. Por exemplo, `se` um processo possui todas as informações necessárias para sua execução, `então` pode ser executado, `senão` as informações que faltam devem ser informadas; e

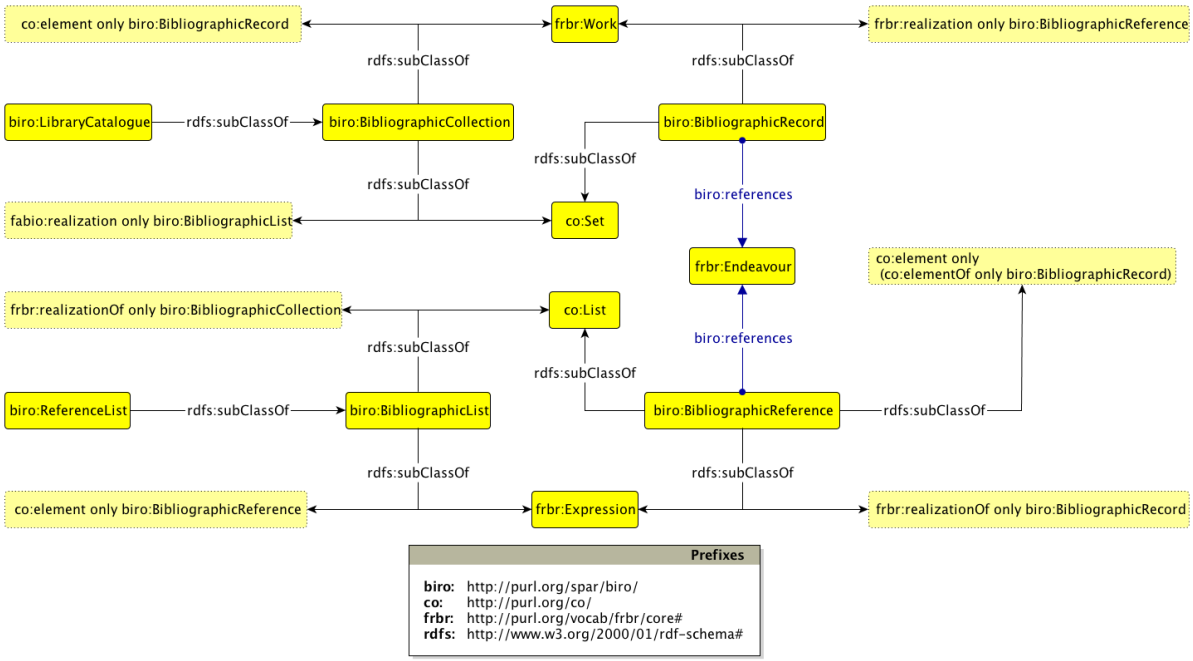
7. **Axiomas:** conjunto de asserções lógicas que, juntamente com as regras, compõem a estrutura geral da ontologia que está sendo definida. Diferentemente da representação tradicional da informação, em que se descreve apenas o conhecimento pré-determinado, ontologias também trabalham com conhecimento posteriormente obtido através da realização de inferências sobre todo conteúdo previamente conhecido. Por exemplo, é possível estabelecer uma transitividade lógica que determina que, `se` todo serviço baseado em SOAP possui um documento WSDL `e` um determinado serviço possui um documento WSDL, `então` esse serviço é baseado em SOAP.

A Figura 12 apresenta uma versão resumida da ontologia BiRO (do inglês, *Bibliographic Reference Ontology*) (IORIO *et al.*, 2014), cujo propósito é definir registros e referências bibliográficas, compilando-os, respectivamente, em coleções e listas. Analisando essa figura, observa-se a classe `Work` e suas subclasses `BibliographicCollection` e `BibliographicRecord`, as quais descrevem coleções e listas, e referências bibliográficas, respectivamente. Na parte inferior da figura é apresentada uma lista de prefixos (retângulo cinza), os quais precedem cada um dos objetos (ou seja, classes, relacionamentos e propriedades) para facilitar sua representação e leitura. Assim como a Internet, que se utiliza de um URL para associar um endereço remoto e uma página, as ontologias, no contexto da Web semântica, utilizam-se de um URI para associar um endereço remoto e um recurso, tradicionalmente uma informação. Por fim, vale destacar que a classe `Work` apresenta o prefixo `frbr`, cuja localização aponta para http://purl.org/vocab/frbr/core#, especificando que essa classe pode ser unicamente identificada através da URI http://purl.org/vocab/frbr/core#Work.

Uma vez apresentada a definição para ontologia e como a mesma pode ser desenvolvida através da utilização de uma linguagem como a OWL (do inglês, *Web Ontology Language*),

[1] Inferência no contexto de ontologias refere-se ao ato de descobrir relações entre entidades das quais inicialmente não se podia afirmar nada.

Figura 12 – Versão resumida da ontologia BiRO



Fonte: Iorio *et al.* (2014)

é necessário definir um método de representação para mostrar como expressar os dados[2] e compartilhar com diferentes pessoas e computadores. O *framework* RDF tem sido utilizado na representação de ontologias por meio de grafos, seguindo uma recomendação da W3C (SEGARAN; EVANS; TAYLOR, 2009).

Segundo a W3C (2014a), o objetivo de um *framework* RDF é representar informações na Web através de um modelo de dados baseado em grafos. Uma das notações mais comuns para representar grafos RDF é a notação Turtle (W3C, 2014b), a qual dispõe de uma sintaxe estruturada por um conjunto de triplas. Essas triplas permitem representar grafos RDF de uma maneira compacta, com a possibilidade de utilizar abreviações para padrões e tipos de dados comuns. Basicamente, uma tripla é formada por três elementos, a saber: sujeito, predicado e objeto. Todos são representados por uma IRI (do inglês, *Internationalized Resource Identifier*), uma generalização das URIs que permite uma maior representatividade de caracteres. Em alguns casos, um sujeito também pode ser representado através de um nó em branco do grafo ao qual pertence e um objeto por um valor literal (por exemplo, um número ou uma *string*) ou até mesmo por um nó em branco do grafo. Por fim, o predicado define a relação entre o sujeito e o objeto.

A Listagem 1 mostra um exemplo de um grafo RDF utilizando notação Turtle. As linhas de 1 a 4 são utilizadas para a definição dos prefixos que são utilizados no restante do grafo. Em seguida, na linha 6 é estabelecida uma relação entre os indivíduos “alice” e “bob”. Nessa relação,

[2] Tradicionalmente, os dados são representados através de grafos.

a propriedade `schema:knows` é utilizada para descrever uma relação social em que as pessoas em ambos os lados (sujeito e objeto) se conhecem. O mesmo tipo de relação é estabelecido entre "bob" e "carol" (linha 8), "carol" e "alice" (linha 13) e "carol" e "bob" (linha 14). As demais linhas utilizam outras propriedades do vocabulário *schema* para descrever informações pessoais de "bob" e "carol", como nome (`schema:name`), data de nascimento (`schema:birthDate`) e local de nascimento (`schema:birthPlace`). Vale observar que o sujeito e o predicado das triplas são sempre representados através de uma IRI, mas um objeto pode ser definido de diferentes maneiras com relação ao contexto em que o mesmo se aplica. Em situações em que se está estabelecendo uma relação com um outro objeto acessível através de uma IRI, sua representação inevitavelmente será a IRI para o objeto. Já em situações em que se trata de um dado atômico, um valor literal será utilizado para representá-lo. Caso esse valor possua um determinado tipo pré-definido, como uma data, seu formato é especificado também pela URI que o representa (por exemplo, `xsd:date`). A Figura 13 ilustra um grafo para a Listagem 1.

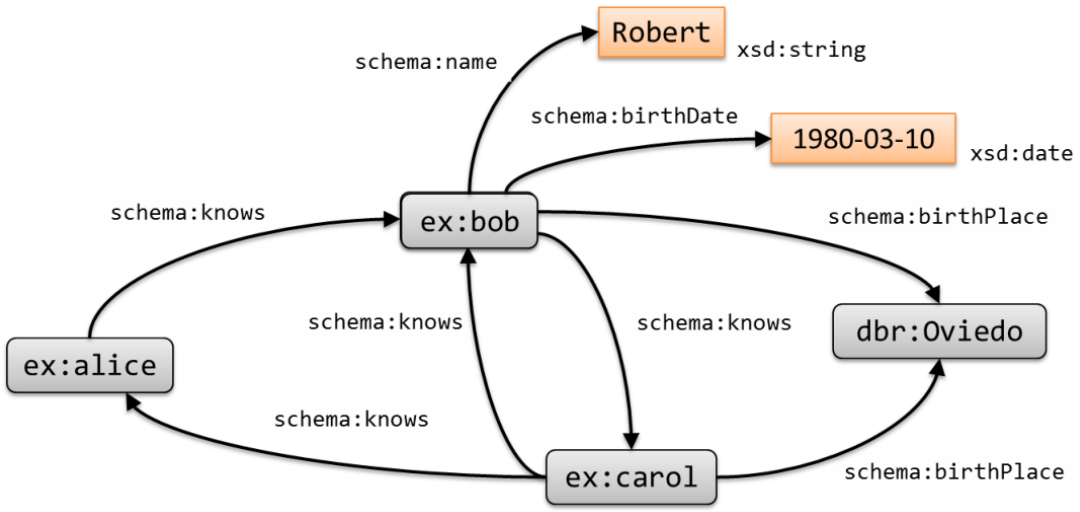
Listagem 1 – Arquivo de descrição de um grafo RDF em notação Turtle

```
1  prefix ex:   <http://example.org/>
2  prefix schema: <http://schema.org/>
3  prefix dbr:  <http://dbpedia.org/resource/>
4  prefix xsd:  <http://www.w3.org/2001/XMLSchema#>
5
6  ex:alice schema:knows  ex:bob .
7
8  ex:bob  schema:knows    ex:carol .
9  ex:bob  schema:name     "Robert" .
10 ex:bob  schema:birthDate "1980-03-10"^^xsd:date .
11 ex:bob  schema:birthPlace dbr:Oviedo .
12
13 ex:carol schema:knows  ex:alice .
14 ex:carol schema:knows  ex:bob .
15 ex:carol schema:birthPlace dbr:Oviedo .
```

Fonte: (GAYO *et al.*, 2017)

Uma vez representada a informação, é necessário definir uma maneira de manipulá-la. Em um banco de dados relacional, a maneira mais comum de se modificar uma informação é por meio de uma linguagem de consulta especificamente voltada para esse propósito, objetivo geralmente alcançado com a utilização da linguagem SQL (do inglês, *Structured Query Language*). A manipulação de dados em um grafo RDF é realizada por meio da linguagem SPARQL, criada especialmente para esse domínio de aplicação (FIONDA; PIRRÒ, 2018). Essa linguagem também segue o modelo de triplas, porém, diferentemente de SQL, uma consulta pode ser subdividida

Figura 13 – Representação em grafo de um arquivo RDF



Fonte: (GAYO *et al.*, 2017)

em quatro diferentes tipos, cada qual com seu propósito específico, a saber:

1. **`SELECT [DISTINCT]:`** resulta em uma tabela baseada nas variáveis[3] definidas em sua declaração. É semelhante à consulta SELECT da linguagem SQL;

2. **`ASK:`** retorna um valor booleano sobre uma determinada consulta, sendo útil para cenários em que se deseja consultar a veracidade de uma determinada informação sem a recuperação de dados. Verificar a veracidade de uma informação fornecida sobre uma pessoa é um exemplo para esse tipo de consulta;

3. **`CONSTRUCT:`** resulta em um novo grafo RDF construído a partir de um conjunto de triplas especificadas nesse tipo de consulta, sendo útil para gerar novos conjuntos de dados a partir de outros dados existentes;

4. **`DESCRIBE:`** retorna um grafo RDF contendo os dados que descrevem a informação que está sendo pesquisada.

Adicionalmente, assim como na linguagem SQL, SPARQL também dispõe de um conjunto de modificadores com o propósito de filtrar as informações que estão sendo consultadas. Nesse sentido, `ORDER BY [ASC|DESC]`, `LIMIT` e o modificador `FILTER` para o processamento de expressões regulares são exemplos de filtros em SPARQL. Além disso essa linguagem também possui alguns operadores reservados para a manipulação de dados em grafos, tais como `INSERT` e `DELETE`. Um exemplo de uma consulta SPARQL é apresentado na Listagem 2 para mostrar o uso dos comando supracitados. Essa consulta retornará uma tabela contendo duas colunas,

[3] Em SPARQL uma variável é definida por uma *string* precedida pelo caractere "?", por exemplo, `?nome`.

Listagem 2 – Consulta SPARQL simples

```
1 prefix : <http://example.org/>
2 prefix schema: <http://schema.org/>
3 prefix dbr: <http://dbpedia.org/resource/>
4
5 SELECT ?x ?y
6 WHERE {
7     ?x schema:birthPlace dbr:Oviedo .
8     ?x schema:knows ?y
9 }
```

Fonte: (GAYO *et al.*, 2017)

representadas pelas variáveis `?x` e `?y`, detalhando as pessoas que nasceram em Oviedo (`?x`) e conhecem (`?y`).

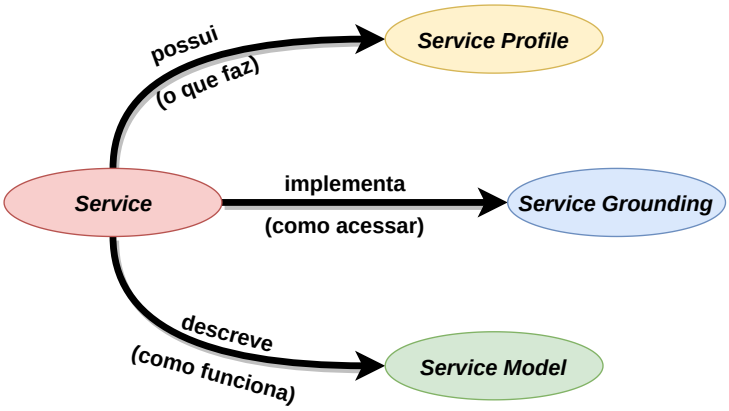
4.2 Ontologias de descrição de serviços Web

De acordo com Wu, Deng e Wu (2015b), o surgimento de tecnologias e padrões para o desenvolvimento de serviços Web facilitou sua adoção em diferentes áreas, tais como: comercial, financeiro e turístico. A crescente publicação de serviços requer mecanismos que facilitem a descoberta de serviços Web de maneira rápida, eficiente e precisa. A tarefa de localizar serviços Web envolve diferentes fatores intrinsecamente ligados com as necessidades dos clientes que utilizarão as aplicações que consomem tais serviços. Essas necessidades englobam aspectos funcionais, como tarefas que são realizadas por um determinado serviço Web e sua interface de comunicação, e não funcionais, como métricas de QoS. Segundo Bennaceur e Nuseibeh (2017), a utilização de uma base de conhecimento aliada a uma ontologia fornece uma maneira mais rica e precisa de representar serviços Web. A partir dessa representação é possível encontrar serviços com maior facilidade em um repositório, além de compor novos com base no conhecimento sobre as tarefas que podem ser executadas e como devem ser configuradas em uma composição de serviço. Outra característica que também pode ser destacada é a possibilidade de redução de tempo gasto na tarefa de composição, uma vez que esse tipo de representação dispensa a necessidade de se analisar individualmente os arquivos WSDL de cada serviço para determinar quais operações podem ser executadas e como uma requisição deve ser enviada.

Os esforços envolvidos na tarefa de representação semântica de serviços Web resultaram no desenvolvimento de diferentes ontologias, cada qual com suas respectivas características. Em Wu, Deng e Wu (2015a) são descritas algumas ontologias para serviços Web: (i) OWL-S; (ii) WSMO (do inglês, *Web Service Modeling Ontology*); e (iii) SWSO (do inglês, *Semantic Web Service Ontology*). As ontologias (ii) e (iii) se baseiam na OWL-S (W3C, 2004a), a qual divide um serviço em três diferentes níveis, como ilustrado na Figura 14. Cada nível de representação

da ontologia OWL-S procura responder três questões relacionadas a um serviço Web, a saber:

Figura 14 – Níveis de representação de um serviço na ontologia OWL-S e seus significados



Fonte: adaptado de W3C (2004a)

O que o serviço proporciona para potenciais clientes? A resposta para essa pergunta especifica quais informações o ***Service Profile*** fornece aos clientes. Em síntese, essas informações descrevem, por exemplo, as habilidades, as limitações e as operações que um serviço Web pode desempenhar. Além disso, um serviço pode ser descrito de maneira mais abrangente através do seguinte conjunto de propriedades:

- ***serviceName*:** armazena o nome do serviço Web que está sendo descrito, podendo ser utilizado como uma forma de identificá-lo;
- ***textDescription*:** apresenta uma descrição resumida das tarefas que o serviço é capaz de executar, mostrando de maneira mais abrangente qual é seu propósito e quais são suas restrições; e
- ***contactInformation*:** fornece meios de contatar a equipe responsável pelo desenvolvimento e/ou manutenção de um serviço, tais como: endereço, e-mail e telefone.

Para descrever como uma requisição deve ser realizada e qual tipo de resultado será produzido por cada operação do serviço, o seguinte conjunto de propriedades é utilizado:

- ***hasParameter*:** relaciona um parâmetro, representado através da classe `Parameter`, com sua respectiva operação. Embora seja referenciado, não é instanciado, pois parâmetros possuem um conjunto de entradas e saídas descritos separadamente. Sendo assim, seu propósito nessa ontologia é manter a integridade do contexto semântico do domínio que está sendo representado, nesse caso, o serviço;
- ***hasInput*:** aponta para um conjunto de entradas como nome de variável e seu respectivo tipo, ambos definidos em um *service model*;

- ***hasOutput*:** aponta para um conjunto de saídas como nome de variável e seu respectivo tipo, ambos definidos em um *service model*;
- ***hasPrecondition*:** especifica uma ou mais condições que devem ter ocorrido previamente à chamada do serviço; e
- ***hasResult*:** especifica qual será o resultado produzido pelo serviço em função das condições de entrada fornecidas.

Como o serviço é utilizado? A resposta para essa pergunta é o que o ***Service Model*** tenta providenciar. O objetivo dessa questão é descrever as operações disponibilizadas pelo serviço Web e como as mesmas devem ser utilizadas. Para isso, os parâmetros de entrada esperados pelo serviço e seu tipo de retorno devem estar evidentes aos clientes desses serviços.

Como uma pessoa interage com o serviço? Para responder esta questão, um ***Service Grounding*** tem como propósito apresentar detalhes sobre como a comunicação entre um serviço e seus clientes deve ocorrer, detalhando características como protocolos de comunicação adotados e número de portas que devem ser acessadas. Além disso, também especifica os formatos de mensagem adotados, instruindo, por exemplo, como os dados devem ser serializados. Isso ocorre através da criação de um vínculo entre a descrição semântica, representada pelo *service grounding*, e a descrição técnica, representada pela propriedade *binding* especificada no arquivo WSDL do serviço, de cada uma das operações atômicas descritas.

4.3 Considerações finais

Neste capítulo foram apresentados os principais conceitos relacionados a elaboração de dados semânticos. A adoção desse tipo de representação no desenvolvimento de serviços Web pode se provar útil no contexto de seleção e substituição de serviços em situações que envolvam a necessidade de se realizar alguma adaptação em virtude da ocorrência de uma variação no ambiente em que um sistema de software está sendo executado. Na Seção 4.1 foi apresentada uma definição para ontologia, cuja adoção fornece um vocabulário através do qual é possível representar os dados de um determinado domínio e seus relacionamentos. Adicionalmente, nessa seção foram introduzidos o *framework* RDF, uma das maneiras de se representar dados semânticos através de grafos compostos de triplas e a linguagem SPARQL, a qual provê uma maneira de se manipular dados em um contexto semântico. Por fim, na Seção 4.2, alguns dos esforços implícitos no desenvolvimento de ontologias para descrição de serviços Web também foram reportados. Também foram expostos conceitos básicos de OWL-S, uma das ontologias mais encontradas nessa área de descrição de serviços. O objetivo de apresentar essa ontologia foi apenas introduzir seus principais conceitos, utilizados neste trabalho. Entretanto, é possível en-

contrar uma descrição mais aprofundada sobre OWL-S diretamente em sua documentação (W3C, 2004a). Assim, este capítulo encerra o conjunto de definições que compõem a fundamentação teórica para o desenvolvimento deste projeto de mestrado acadêmico.

5 Trabalhos relacionados

Com o passar do tempo foi observado que, conforme uma área de pesquisa evolui, há geralmente um aumento no número de relatórios e resultados disponibilizados para acompanhar tal evolução. Durante o estudo de uma nova área de conhecimento, pesquisadores geralmente realizam algum tipo de revisão da literatura, quase sempre de maneira informal, para identificar publicações relacionadas ao tema da pesquisa. Esse tipo de revisão não utiliza uma abordagem sistemática e pode resultar em uma visão parcial sobre o tema de interesse. Assim, o uso de mecanismos para fornecer uma visão mais justa sobre uma área ou tópico de interesse torna-se importante (PETERSEN *et al.*, 2008). Desse ponto de vista, a revisão sistemática tem sido adotada na Engenharia de Software Baseada em Evidências (do inglês, Evidence-Based Software Engineering – EBSE) como uma maneira de realizar pesquisas na literatura através de um conjunto de passos bem definidos (KITCHENHAM; DYBA; JORGENSEN, 2004).

A realização de uma revisão formal da literatura permite uma avaliação abrangente e sistemática de um tema de pesquisa através da adoção de uma estratégia de busca bem definida visando mitigar ou, no mínimo, minimizar influências que o pesquisador possa sofrer (KITCHENHAM; CHARTERS, 2007). Em outras palavras, a adoção de técnicas de revisão da literatura permite sistematizar um conhecimento, possibilitando resumir, avaliar e interpretar a relevância de todas as evidências relacionadas a uma questão específica, área temática ou fenômeno de interesse. Uma evidência individual[1] que contribui para um estudo secundário[2] é chamada de estudo primário. Sendo assim, neste trabalho foram utilizadas técnicas de estudo secundário para explorar a área de pesquisa alvo neste projeto de mestrado. Portanto, a condução de um mapeamento sistemático tem como objetivo estabelecer uma visão justa e, ao mesmo tempo, ampla com relação ao desenvolvimento de *frameworks* para *Self-Apps*.

O objetivo deste capítulo é apresentar os resultados da revisão da literatura que compõe o conjunto de trabalhos relacionados. Com o intuito de ampliar a cobertura de resultados, essa revisão foi dividida em duas atividades de busca: (i) condução de uma revisão formal da literatura por meio de um "Estudo de Mapeamento Sistemático" (do inglês, *Systematic Mapping Study* – SMS); e (ii) pesquisa complementar utilizando os mesmos termos e sinônimos do SMS em duas bases de teses e dissertações nacionais (Biblioteca Digital de Teses e Dissertações (BDTD) [3] e Catálogo de Teses e Dissertações da CAPES) [4]. Para apresentar os resultados dessa revisão, este capítulo foi organizado da seguinte maneira: na Seção 5.1 são reportados detalhes de

[1] Um estudo de caso ou um estudo experimental relatado em uma publicação e/ou artigo
[2] Revisão sistemática, mapeamento sistemático, *surveys*, ou uma combinação desses estudos
[3] Disponível em: <http://bdtd.ibict.br/vufind/>. Acesso em: 03/05/2020
[4] Disponível em: <http://catalogodeteses.capes.gov.br/catalogo-teses/>. Acesso em: 03/05/2020

como foi conduzida a pesquisa na literatura composta por publicações, teses e dissertações, e um compilado das informações obtidas através da execução do mapeamento sistemático. Por fim, na Seção 5.2 são apresentados estudos complementares, obtidos a partir de uma pesquisa adicional realizada em bases nacionais de teses e dissertações.

5.1 Mapeamento sistemático da literatura

O objetivo desta seção é apresentar a estratégia de pesquisa adotada neste trabalho e os resultados obtidos através do mapeamento sistemático da literatura conduzido. O mapeamento é uma técnica, proposta pela EBSE (KITCHENHAM; DYBA; JORGENSEN, 2004), que permite estabelecer um panorama geral sobre um tema de pesquisa (KITCHENHAM; CHARTERS, 2007; PETERSEN *et al.*, 2008; PETERSEN; VAKKALANKA; KUZNIARZ, 2015). A aplicação dessa técnica de revisão permitiu verificar em diferentes bases de publicação a existência de estudos primários relacionados ao tema de pesquisa deste trabalho (*frameworks* para *Self-Apps*). Para isso, um protocolo de pesquisa foi elaborado (ver Apêndice A), onde são apresentados os detalhes do mapeamento, a saber: Questões de Pesquisa (QP), conjunto de bases utilizadas para coleta de estudos, *string* de busca e demais procedimentos adotados durante a condução do mapeamento. É importante ressaltar que a *string* de busca utilizada é composta por três termos, a saber: "*Framework*", "*Web Service*" e "*Self-adaptive*". A partir destes foi definida uma lista de sinônimos baseada em um processo de calibragem incremental, determinando os diferentes graus de relevância de cada sinônimo e removendo aqueles cuja inclusão não trouxeram nenhum estudo adicional.

A Tabela 3 mostra o número bruto de estudos para cada base de publicação em dois períodos. Cronologicamente, os números apresentados em 2019 representam a primeira execução do mapeamento e os de 2020 mostram sua atualização. A primeira execução (Janeiro/2019) teve como finalidade obter um panorama geral do estado da arte do tema investigado. Esse conjunto de dados permitiu estabelecer um comparativo em relação as abordagens adotadas por outros autores e a aplicação de tais abordagens em partes já desenvolvidas nesse trabalho. Isso permitiu reforçar a viabilidade das técnicas aplicadas, além de estimular a discussão sobre melhorias no trabalho que estava em andamento. Após o período de um ano foi realizada uma nova execução da pesquisa focada na análise de estudos, executada com um filtro de data delimitando o período de Janeiro a Dezembro de 2019, visando complementar os dados já coletados.

Em relação aos resultados, o número de estudos obtidos na base *Springer Link* foi elevado por existirem apenas filtros por área de pesquisa e idioma. Sendo assim, a busca foi realizada sem restrições de campos e, para auxiliar o processo de seleção de estudos, uma ferramenta foi desenvolvida por nosso grupo de pesquisa para filtrar e compatibilizar o processo de busca aplicado nas demais bases.

Tabela 3 – Número de estudos obtidos por base de pesquisa

Base pesquisada	Número de estudos (2019)	Número de estudos (2020)
ACM Digital Library	375	289
IEEE Xplore	485	92
ScienceDirect	107	22
Scopus	904	84
Springer Link	11208	1233
Web of Science	573	35
Total	**13646**	**1755**

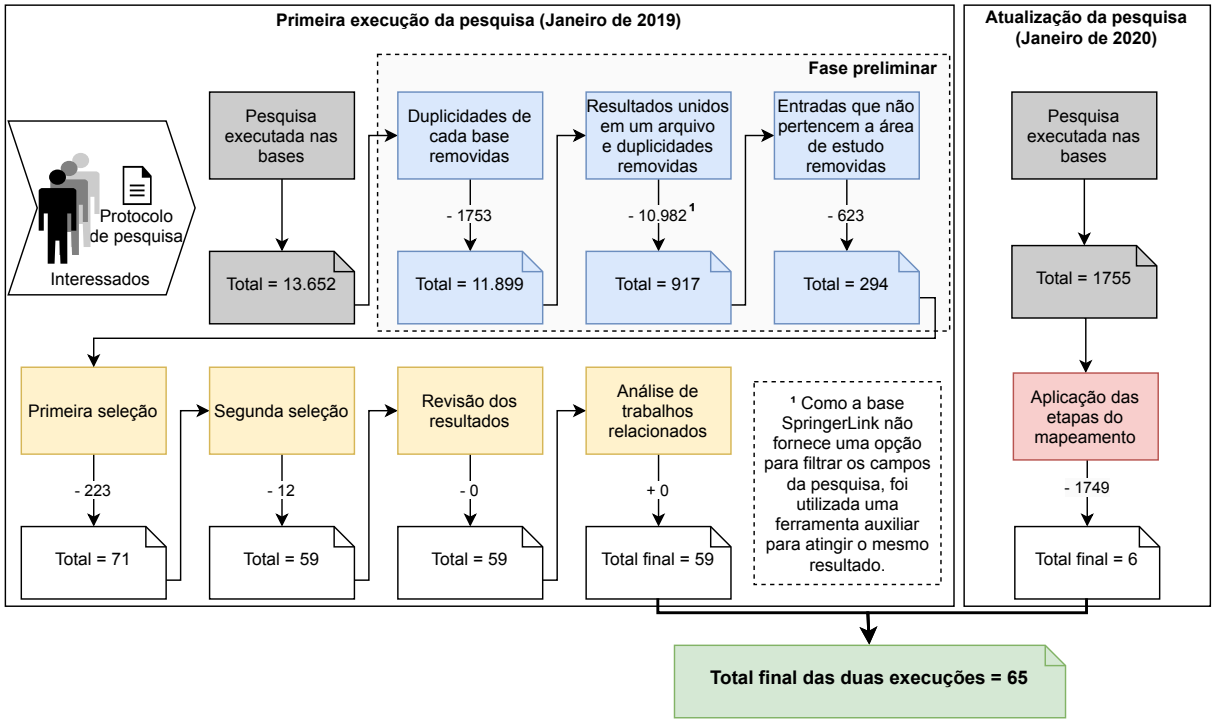
A Figura 15 apresenta em detalhes os números obtidos em cada etapa da execução da pesquisa. As etapas destacadas em cinza indicam os resultados da execução da *string* de busca nas bases de pesquisa mostradas na Tabela 3 e, para cada uma das execuções, o total de resultados recuperados. Como diferentes bases podem indexar o mesmo estudo, antes de iniciar as etapas do mapeamento foi necessário preparar os resultados por meio de uma fase de pré-processamento, indicada pelas etapas destacadas em azul. Nessas etapas foram realizadas tarefas como: remoção de duplicidades individuais de cada base, união dos resultados de todas as bases em um único arquivo, removendo duplicidades entre as bases. Também foram removidos estudos pertencentes à outras áreas e que, embora utilizem os termos da *string* de busca, não retratam a área de pesquisa avaliada. Por fim, as etapas destacadas em amarelo especificam os processos do mapeamento, a saber:

- **Primeira seleção:** momento da pesquisa em que foram aplicados os critérios de seleção definidos no Apêndice A. Tais critérios foram aplicados através da análise do título, resumo e palavras-chave dos estudos. A ferramenta JabRef[5] foi utilizada para auxiliar no processo, pois possibilita customizar diferentes níveis de visões para arquivos de bibliografias. Tal característica tornou possível especificar um modo de visualização que destacasse apenas os campos analisados: título, resumo e palavras-chave. Adicionalmente, sua aplicação viabiliza a inserção dos critérios de seleção diretamente na bibliografia, concentrando todos os dados necessários em um só lugar e, consequentemente, tornando o processo de revisão mais ágil;

- **Segunda seleção:** o conjunto de estudos resultante da primeira seleção foi submetido à uma segunda seleção, momento em que o texto de cada um dos estudos foi lido na íntegra e os critérios de seleção foram novamente aplicados. Adicionalmente, para facilitar o processo de síntese de dados, durante a realização dessa etapa também foram extraídas as informações necessárias para responder as QPs de nosso mapeamento;

[5] Disponível em: <http://jabref.org/>. Acesso em: 03/05/2020

Figura 15 – Etapas da condução do mapeamento sistemático



Fonte: Elaborada pelo autor

- **Revisão dos resultados:** com o intuito de aumentar o grau de confiabilidade do mapeamento, todos os dados extraídos dos estudos selecionados após a segunda seleção foram revisados. Em situações que houvessem dúvidas sobre determinada característica e/ou funcionalidade de um dos *frameworks* avaliados, certas partes de cada estudo foram lidas novamente. Caso tal leitura não fosse capaz de sanar essas dúvidas, o texto era lido na íntegra novamente e, se necessário, discutido entre os participantes do mapeamento. Durante essa etapa foi possível extrair dados adicionais, inicialmente não observados durante as etapas anteriores; e

- **Análise dos trabalhos relacionados:** nesta etapa foram verificadas as referências utilizadas pelos estudos selecionados. A partir dessa verificação, foi possível analisar se não existia nenhum trabalho relacionado que se encaixava dentro dos critérios de inclusão estabelecidos pelo mapeamento e que não tenha sido encontrado durante a execução da busca nas bases de pesquisa.

É importante ressaltar que todo o processo descrito também foi aplicado durante a segunda execução da pesquisa. Entretanto, optou-se por não exibir as etapas conduzidas de maneira individual para essa atualização da pesquisa, sintetizando-as em uma única etapa que representa a totalidade dos passos descritos (destacada em vermelho – Figura 15). A seguir são apresentadas as respostas para cada QP de nosso mapeamento. Essas respostas foram sintetizadas

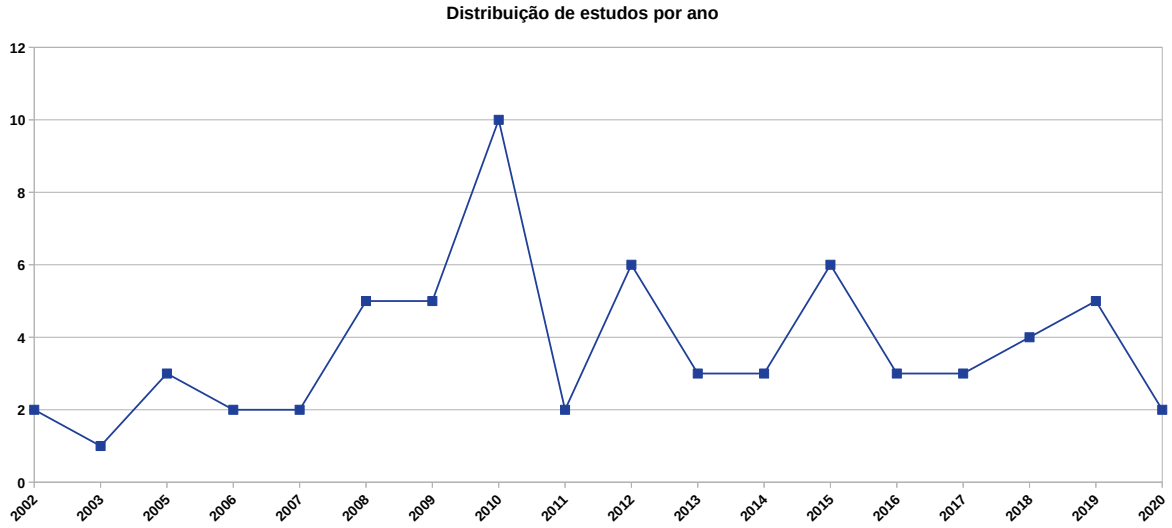
a partir de um conjunto de 65 estudos, cuja lista com título, autores e ano de publicação é apresentada na Tabela 12 (ver Apêndice B).

QP1: Quais são as características demográficas de estudos sobre *frameworks* para *Self-Apps*?

O objetivo desta questão é obter um panorama sobre o período de publicação dos estudos voltados para o tema dessa dissertação, seus principais veículos de publicação, sua aplicabilidade/origem na indústria ou na academia e sua distribuição entre os grupos de pesquisa envolvidos. Os dados foram extraídos e sintetizados em quatro diferentes subquestões, a saber:

QP1.1: *Quando os estudos foram publicados?* A Figura 16 mostra a distribuição de estudos por ano. A partir dessa distribuição é possível observar que o mapeamento identificou estudos entre os anos de 2002 e 2020, com maior concentração em 2010, 2012 e 2015. Em 2019 observa-se uma aproximação quanto ao número de estudos publicados em relação aos dois últimos supracitados. Analisando a distribuição de estudos dessa figura pode-se observar que o tema de pesquisa deste mapeamento ainda desperta interesse, reforçando a evidência de que esse ainda é um tema passível de exploração nos próximos anos.

Figura 16 – Distribuição de estudos por ano (intervalo de 2002 a 2020)

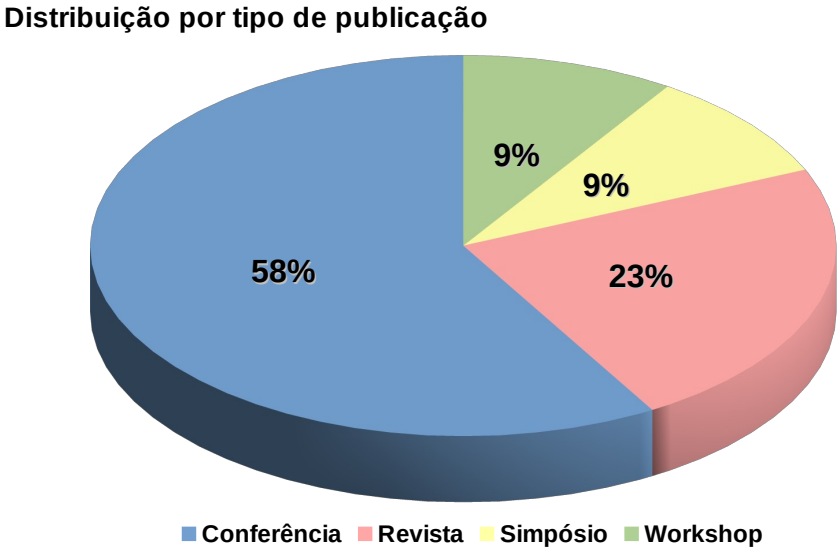


Fonte: Elaborada pelo autor

QP1.2: *Onde os estudos foram publicados?* Para responder à essa pergunta foram estabelecidas quatro possíveis origens para os estudos: conferências, revistas, simpósios e *workshops*, as quais foram coletadas de cada estudo durante a extração de dados. A Figura 17 apresenta a distribuição desses estudos conforme o tipo de publicação. A parte superior da figura

ilustra um gráfico de pizza que mostra a distribuição em porcentagem e a parte inferior identifica os estudos de acordo com seus respectivos tipos.

Figura 17 – Distribuição de estudos por tipo de publicação



Tipo	Estudos	Total
Conferência	S1, S3, S4, S5, S6, S7, S10, S11, S12, S13, S15, S16, S19, S20, S23, S25, S26, S27, S28, S29, S31, S32, S33, S34, S36, S38, S39, S40, S41, S42, S44, S46, S53, S55, S56, S59, S60, S65	38
Revista	S2, S9, S17, S18, S22, S24, S45, S47, S48, S51, S52, S57, S61, S62, S64	15
Simpósio	S21, S30, S37, S43, S54, S63	6
Workshop	S8, S14, S35, S49, S50, S58	6

Fonte: Elaborada pelo autor

QP1.3: *Como as publicações se distribuem entre academia e indústria?* Para responder à essa pergunta foram identificados em quais cenários os *frameworks* haviam sido elaborados e, principalmente, para quais situações se destinavam. Para isso, três categorias foram utilizadas na classificação dos estudos de nosso mapeamento, as quais são detalhadas a seguir:

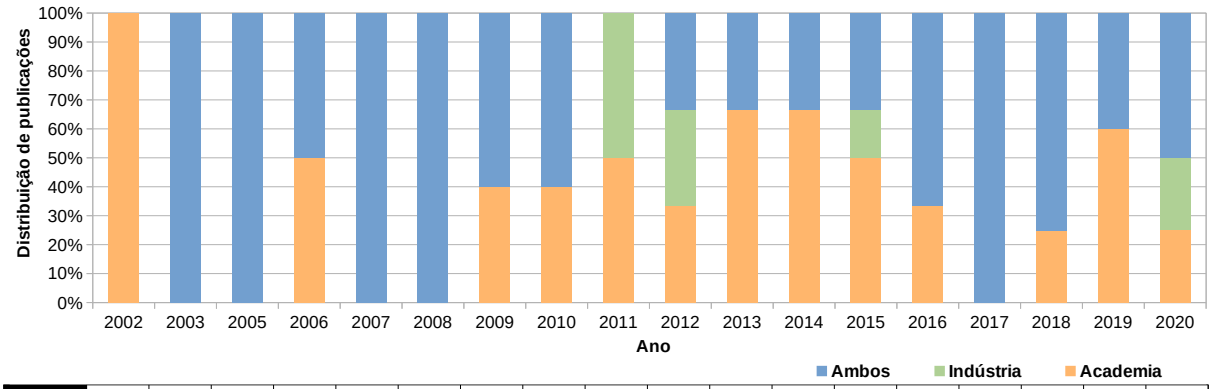
1. **Academia:** um estudo se enquadra nesta categoria quando o assunto é abordado com maior foco nos aspectos teóricos e experimentais. Nessa categoria os estudos geralmente apresentam detalhes mais específicos sobre os algoritmos que foram utilizados e quais técnicas foram aplicadas. Sua avaliação é conduzida através da realização de experimentos, simulações, comparações com outros estudos e exemplos de estudos de caso;
2. **Indústria:** nesta categoria um *framework* é desenvolvido a partir de um conjunto de requisitos bem determinado e se baseia em um protótipo funcional. Aqui, o principal objetivo está em demonstrar a aplicabilidade do *framework* em uma necessidade ime-

diata, podendo dispensar discussões sobre algoritmos e técnicas que foram utilizadas; e

3. **Ambos:** esta categoria engloba estudos que apresentam *frameworks* desenvolvidos considerando tanto a fundamentação teórica da academia quanto a aplicabilidade imediata exigida pela indústria na solução de um problema.

A Figura 18 mostra a distribuição anual dos estudos por categoria de publicação. Como pode-se observar, a maioria dos estudos foi enquadrada como pertencente à ambas categorias, o que pode ser considerado um fator positivo por evidenciar uma certa integração entre academia e indústria na evolução da área de conhecimento deste trabalho. Além disso, pôde-se constatar que os estudos desta categoria visam resolver problemas atuais, tendo embasamento teórico como sustentação de suas soluções e avaliação a partir de implementações e situações reais.

Figura 18 – Distribuição de estudos entre academia e indústria



	2002	2003	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
Academia	2	0	0	1	0	0	2	4	1	2	2	2	3	1	0	1	3	1
Indústria	0	0	0	0	0	0	0	0	1	2	0	0	1	0	0	0	0	1
Ambos	0	1	3	1	2	5	3	6	0	2	1	1	2	2	3	3	2	2

Fonte: Elaborada pelo autor

QP1.4: *Como as publicações se distribuem geograficamente?* Identificar a distribuição geográfica de um conjunto de estudos pode ser considerado como um elemento importante durante a condução de um mapeamento, uma vez que a partir disso é possível identificar as localidades e grupos de pesquisa associados com o tema que está sendo investigado. Para responder à essa questão foram extraídas as afiliações de cada autor dos estudos selecionados. Em seguida, essas informações foram sintetizadas na ferramenta *Projection Explorer* (PEx)[6](JOIA *et al.*, 2011). A Figura 19 apresenta esses dados, relacionando estudos, universidades e países. Ao analisar essas relações nota-se que: (i) Itália e China, com 10 e 9 estudos, respectivamente; (ii) Estados Unidos e França, ambos com 7 estudos

[6] Disponível em: <http://vicg.icmc.usp.br/vicg/tool/1/projection-explorer-pex>. Acesso em: 03/05/2020

cada; e (iii) Tunísia, com 5 estudos, são os países de maior incidência neste mapeamento. Juntos, esses países representam 58% do total de estudos.

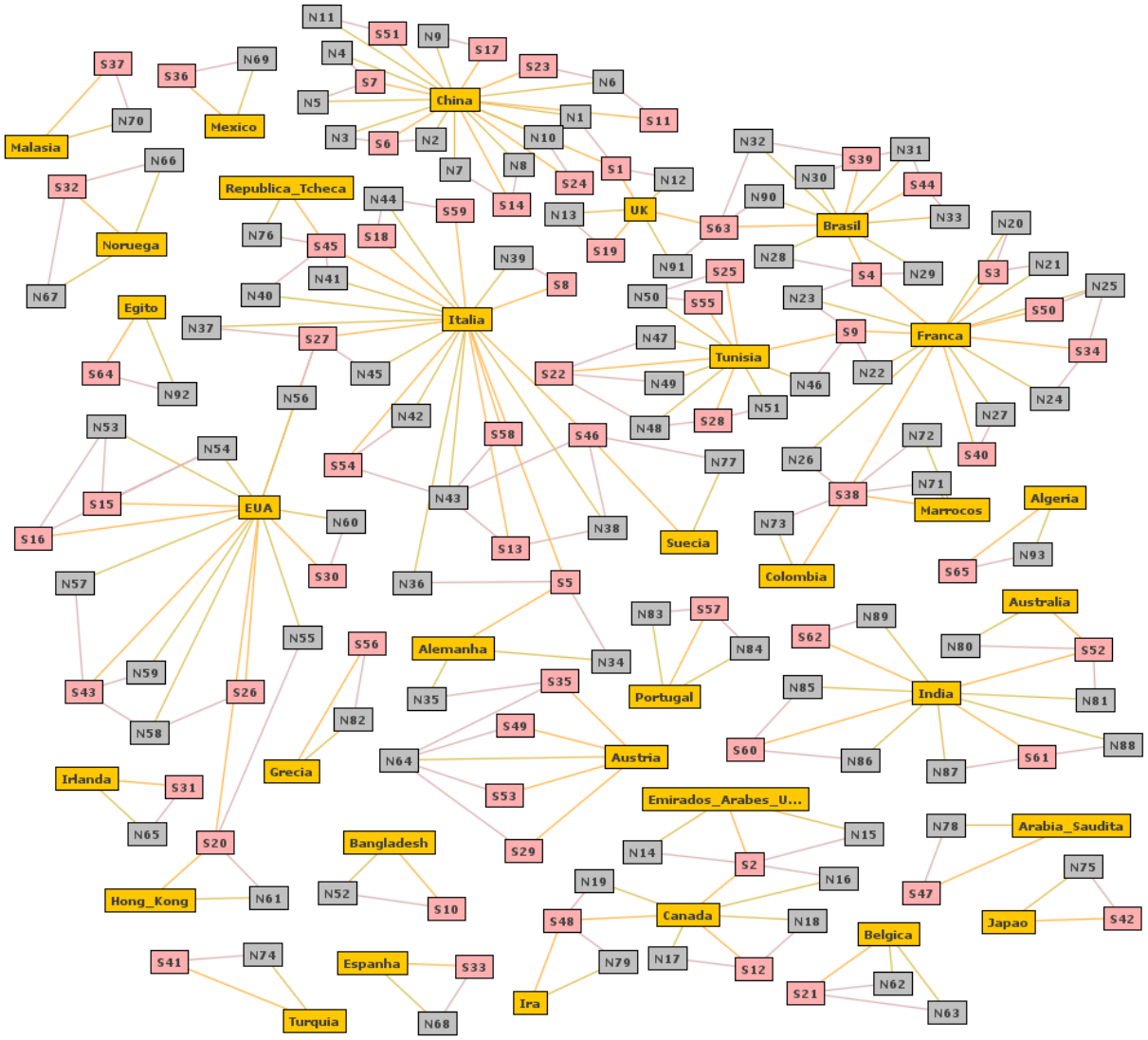
QP2: Quais domínios de aplicação se beneficiam da existência de *frameworks* para *Self-Apps*?

Após a análise dos 65 estudos selecionados neste mapeamento, 39 estudos apresentaram um domínio de aplicação e, portanto, foram utilizados para responder essa questão. De maneira geral, tais dados puderam ser obtidos através da análise da seção de avaliação do *framework*, em particular para estudos que adotavam o desenvolvimento de aplicações ou de estudos de caso. Desses 39 estudos, apenas dois domínios de aplicação foram extraídos, sendo cada um com seu respectivo conjunto de subdomínios, como ilustrado na Figura 20. O nome para cada domínio/subdomínio e sua respectiva descrição foram baseados nas nomenclaturas e definições apresentadas pelos autores dos estudos analisados. A seguir, uma breve descrição de cada domínio e subdomínio é apresentada.

Sistemas da Informação (SI): este domínio representa aplicações que lidam através da coleta e distribuição de dados e/ou informações de/para seus usuários. Nesse tipo de domínio a ocorrência de falhas não é algo que apresente qualquer tipo de risco a integridade de seus usuários, resultando apenas em inconveniências que possam ser resolvidas posteriormente. Do conjunto de 39 estudos, 31 foram classificados como pertencentes a esse domínio e seus respectivos subdomínios, os quais são detalhados a seguir:

1. **Comércio eletrônico.** As aplicações desenvolvidas para este subdomínio têm como objetivo auxiliar o usuário na compra e venda de produtos através da Internet, como por exemplo, aplicativos para entrega de alimentos. Apenas um estudo pertence a esse subdomínio;
2. **Compartilhamento de mídia.** Refere-se a aplicações cujo propósito é o envio de fotos, vídeos e áudio para um servidor remoto que possa ser acessado a partir de qualquer local, como por exemplo, um serviço de transcodificação de mídia. Apenas um estudo pertence a esse subdomínio;
3. **Educação.** A proposta deste tipo de aplicação é facilitar o acesso aos meios de educação, tornando-os mais eficientes. Em outras palavras, se resume a qualquer aplicação voltada para fins de ensino, como por exemplo, um serviço de biblioteca digital. Apenas um estudo pertence a esse subdomínio;
4. **Computação móvel.** Aplicações relacionadas a esse subdomínio envolvem sistemas de computação pervasiva, *smartphones* e dispositivos IoT. Por exemplo, um aplicativo

Figura 19 – Distribuição de estudos por localidade

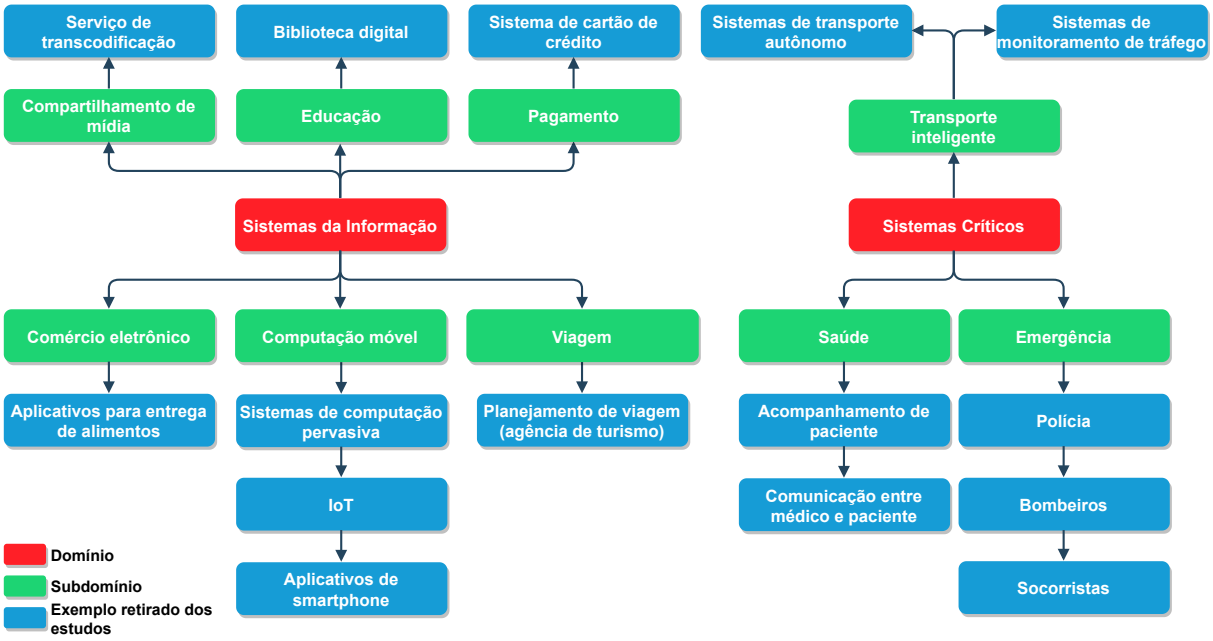


Legenda:

N1 National University of Defense Technologies	N33 Universidade Federal Fluminense	N63 Ghent University
N2 Nanjing Xiaozhuang University	N34 University of Duisburg-Essen	N64 Vienna University of Technology
N3 Hohai University	N35 University of Stuttgart	N65 University College Dublin
N4 Shandong University of Science & Technology	N36 FBK-Irst	N66 University of Oslo
N5 TaiZhou Vocational & Technical College	N37 Università del Sannio	N67 SINTEF ICT
N6 Tianjin University	N38 Università di Roma Tor Vergata	N68 Universidad de Málaga
N7 Beijing Institute of Technology	N39 Università degli Studi di Napoli	N69 Universidad Autónoma de Yucatán
N8 Linyi University	N40 Università degli Studi di Milano	N70 Universiti Teknologi PETRONAS
N9 Zhejiang University	N41 Università degli Studi di Bergamo	N71 Université Mohammed V-Rabat
N10 Peking University	N42 Università degli Studi dell'Aquila	N72 École Nationale Supérieure des Mines de Rabat
N11 Northeastern University	N43 Politecnico di Milano	N73 Universidad EAFIT
N12 University of Derby	N44 Consiglio Nazionale delle Ricerche	N74 Middle East Technical University
N13 Liverpool John Moores University	N45 Seconda Università di Napoli	N75 National Institute of Informatics
N14 Khalifa University	N46 University of Sfax	N76 Charles University in Prague
N15 Etisalat Britsh Telecom Innovation Center	N47 Sousse National School of Engineers	N77 Blekinge Institute of Technology
N16 Université du Québec à Montréal	N48 Jendouba University	N78 King Saud University
N17 University of Ontario	N49 Institut Supérieur de Gestion de Tunis	N79 Iran University of Science and Technology
N18 University of Ottawa	N50 Institut supérieur d'informatique et de multimédia de Sfax	N80 University of Melbourne
N19 Auton Solutions		N81 Thapar Institute of Engineering and Technology
N20 Université de Rennes 1	N51 Faculty of Sciences of Tunis	N82 Athens University of Economics and Business
N21 Technople Brest-Iroise IT/Telecom Bretagne	N52 University of Dhaka	N83 Instituto Politécnico do Porto
N22 Laboratoire d'analyse et d'architecture des systèmes	N53 Arizona State University	N84 Universidade de Coimbra
N23 Université de Toulouse	N54 Miami University	N85 SSN College of Engineering
N24 Université Lyon 1	N55 University of Wisconsin	N86 Anna University
N25 Institut national des sciences appliquées de Lyon	N56 Ohio State University	N87 Devi Ahilya University
N26 Université Paris Panthéon-Sorbonne	N57 University of Michigan-Flint	N88 Prestige Institute of Engineering
N27 Université de Lorraine	N58 Wayne State University	N89 Cisco Systems India
N28 Universidade Estadual Paulista	N59 University of Michigan-Dearborn	N90 Instituto Federal de Pernambuco
N29 Universidade Estadual de Campinas	N60 Carnegie Mellon University	N91 The Open University
N30 Universidade Federal do Maranhão	N61 The Hong Kong University of Science and Technology	N92 Mansoura University
N31 PUC-Rio	N62 University of Antwerp	N93 University of Tlemcen Tlemcen
N32 Universidade de São Paulo		

Fonte: Elaborada pelo autor

Figura 20 – Gráfico organizacional dos domínios e subdomínios encontrados



Fonte: Elaborada pelo autor

de *Internet Banking* ou um sistema de monitoramento de previsão do tempo. Cinco estudos pertencem a esse subdomínio;

5. **Viagem.** Envolve aplicações voltadas para o planejamento de viagens, como por exemplo, uma agência de turismo responsável por reservar passagens aéreas, cruzeiros e/ou ferroviárias. Apenas um estudo pertence a esse subdomínio; e
6. **Pagamento.** Aplicações pertencentes a este subdomínio englobam sistemas de pagamento responsáveis por enviar dinheiro para outras instituições e/ou pessoas, como por exemplo, um sistema de pagamento de cartão de crédito. Apenas um estudo pertence a esse subdomínio.

É importante destacar que a soma de estudos dos subdomínios apresentados não corresponde ao total reportado para o domínio de sistemas da informação. Durante a etapa de análise, 21 estudos apresentaram algum tipo de avaliação, no entanto nenhum domínio ou subdomínio de aplicação estava evidente. Embora sua real natureza não tenha sido descrita, essas avaliações reportavam o desenvolvimento de sistemas de uso comum, os quais foram classificados como pertencentes ao domínio de sistemas da informação, resultando no total de 31 estudos classificados nesse domínio de aplicação.

Sistemas Críticos (SC): este domínio é composto por aplicações em que a ocorrência de uma ou mais falhas pode resultar em impactos severos, seja em fatores como performance ou segurança de seus usuários, ou integridade do ambiente de execução. Foram identificados oito estudos nesse domínio e seus respectivos subdomínios, os quais são descritos a seguir:

1. **Saúde.** Aplicações desenvolvidas para este subdomínio estão diretamente relacionadas com tarefas como acompanhamento em tempo real de pacientes e também no estabelecimento de canais diretos de comunicação entre médico e paciente. Seis estudos pertencem a esse subdomínio;

2. **Emergência.** Este tipo de aplicação possibilita a seus usuários detectar a ocorrência de uma situação de emergência e comunicá-la imediatamente à organização responsável, como, por exemplo, polícia, bombeiros e socorristas. Apenas um estudo pertence a esse subdomínio; e

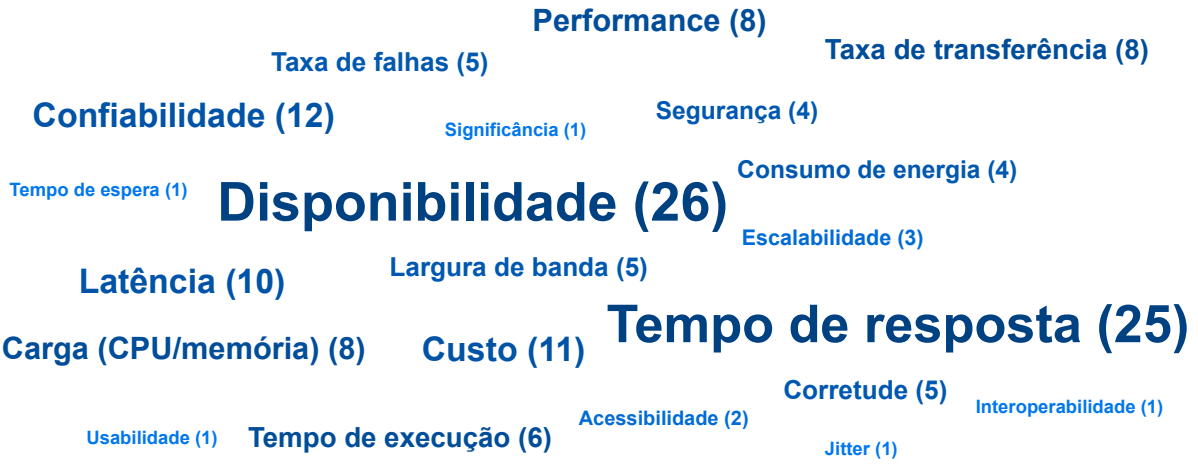
3. **Transporte inteligente.** Este tipo de aplicação tem como objetivo coordenar a movimentação de veículos autônomos, tais quais carros, trens e aviões. Dois estudos pertencem a esse subdomínio.

Em relação aos subdomínios supracitados, observa-se que a soma final dos estudos resulta em 9 estudos e difere do total de 8 estudos para o domínio de sistemas críticos. Essa divergência se deu porque o estudo S10 apresenta exemplos de aplicações em dois subdomínios distintos: saúde e serviços de emergência.

QP3: Quais são os atributos de qualidade de serviço propostos para *Self-Apps*?

Durante a extração de dados dos estudos selecionados no mapeamento, foram coletados os atributos de QoS mencionados em cada uma das publicações. A seguir, Figura 21, é ilustrada a distribuição dos atributos identificados neste mapeamento e seu respectivo número de ocorrência. Pode-se observar por essa figura que disponibilidade (26 estudos) e tempo de resposta (25 estudos) foram os atributos de QoS mais recorrentes durante a concepção e avaliação de *Self-Apps*. Em relação aos nomes dos atributos de QoS presentes nessa figura, vale destacar quem nem todos estavam evidentes nos estudos deste mapeamento e alguns tiveram que ser interpretados pela equipe executora. Além disso, diferentes nomes para o mesmo atributo de QoS também foram encontrados. Visando evitar ou minimizar a inserção de algum tipo de viés, os nomes foram preservados da maneira como foram encontrados nos estudos. Durante a fase de síntese de dados, os nomes dos atributos foram agrupados de acordo com sua proposta e definição formal na literatura. Em um primeiro momento, esse agrupamento visou reduzir o risco da existência de atributos com nomes distintos e de mesmo contexto/significado. Por fim, a padronização dos nomes teve como base dois fatores: (i) pesquisa por estudos secundários, localizando a variante mais adotada na literatura; e (ii) número de ocorrências, verificando o atributo com maior número de ocorrências dentre os estudos selecionados.

Figura 21 – Atributos de QoS encontrados e seus respectivos números de ocorrência



Fonte: Elaborada pelo autor

QP4: Quais são as evidências que motivaram a adoção dos *frameworks* existentes?

O objetivo ao estabelecer esta questão de pesquisa foi encontrar evidências que indicassem o nível de maturidade de *frameworks* para *Self-Apps* e, principalmente, quais foram os processos de avaliação adotados. Do total de estudos selecionados para a etapa de extração de dados, 14 não apresentaram evidências quanto aos processos de avaliação. A partir dos demais estudos foi possível elaborar a Tabela 4, a qual correlaciona os métodos de avaliação identificados e seus respectivos estudos. A seguir, uma breve descrição de cada método é apresentada:

1. **Comparação com outros algoritmos:** tipo de avaliação geralmente focado em determinar a performance do *framework*. Para atingir esse objetivo deve-se medir o tempo necessário para realizar a adaptação e compará-lo com outros algoritmos já existentes;

2. **Desenvolvimento de estudos de caso:** este método de avaliação visa analisar o *framework* em cenários próximos ao mundo real, permitindo realizar análises mais completas do ambiente de execução, tais como: comportamentos esperados e inesperados, particularidades sobre um domínio de aplicação, entre outros;

3. **Desenvolvimento de protótipos:** este método de avaliação baseia-se no desenvolvimento de um protótipo funcional de uma aplicação, sendo útil para demonstrar o funcionamento do *framework* proposto e suas possíveis limitações. Além disso, pode atuar de maneira complementar ao desenvolvimento de um estudo de caso, permitindo obter uma documentação teórica do sistema e um protótipo baseado na documentação produzida;

4. **Discussão de um ou mais cenários de adaptação:** neste método de avaliação procura-se

identificar possíveis cenários de adaptação que seriam enfrentados pela aplicação e como o *framework* seria aplicado nessas situações;

5. **Pesquisa com usuários:** este método de avaliação foca principalmente nos usuários da aplicação, especificamente em seu grau de satisfação. Trata-se de uma avaliação mais genérica que visa determinar se o usuário está satisfeito com os resultados da adaptação. Por exemplo, em uma situação que demanda a substituição de serviço, avalia se tal modificação preservou a capacidade de execução da aplicação, se o tempo de resposta permanece satisfatório, entre outros fatores;

6. **Condução de experimentos:** um protocolo de execução deve ser elaborado para este tipo avaliação, o qual deve permitir avaliar uma ou mais características do *framework*. Para isso, um conjunto de dados (do inglês, *datasets*) deve ser considerado para criar os cenários que se pretende simular. Em seguida, os resultados são analisados para que possam ser extraídos parâmetros quanto aos comportamentos esperados e inesperados. Um aspecto importante desse tipo de avaliação é sua capacidade de replicação, fornecendo análises comparativas mais precisas em trabalhos futuros;

7. **Simulações:** este tipo de avaliação visa estabelecer simulações para cenários de adaptação que se pretende avaliar acompanhadas, de maneira complementar, por discussões. Essas simulações geralmente são compostas por testes dotados de alguma métrica definida para comparação e avaliação dos resultados de um determinado número de execuções; e

8. **Teste de aplicações previamente desenvolvidas:** este tipo de avaliação, em sua maioria, consiste em criar *benchmarks* de aplicações já existentes ou desenvolvidas exclusivamente para analisar algum aspecto de performance de um *framework*.

Tabela 4 – Métodos de avaliação de *frameworks* para *Self-Apps*

Métodos	Estudos	Total
Comparação com outros algoritmos	S24	1
Desenvolvimento de protótipos	S2, S4, S19, S24, S27, S30, S39, S42, S63	9
Discussão de um ou mais cenários de adaptação	S3, S23, S40, S42, S44, S54, S60	7
Pesquisa com usuários	S60	1
Condução de experimentos	S1, S7, S8, S9, S11, S13, S14, S18, S29, S39, S41, S43, S48, S51, S52, S57, S64, S65	18
Simulações	S20, S21, S47, S59, S62	5
Teste de aplicações previamente desenvolvidas	S2, S4, S15, S16, S17, S46, S61	7

QP5: Quais estratégias de busca de serviços estão sendo aplicadas em *frameworks* para *Self-Apps*?

A utilização de uma estratégia de busca de serviços pode ser considerada um elemento vital para o processo de adaptação requerido por *frameworks* que atuam no contexto de *Self-Apps*. Uma vez identificada a necessidade de se realizar uma adaptação de serviço, seja por falha de execução ou degradação de QoS, um serviço similar deve ser localizado e substituído sem a percepção de seus usuários. O conjunto de estudos desse mapeamento revelou as seguintes estratégias de seleção:

1. **Algoritmos de otimização por inteligência de enxame:** para este tipo de estratégia é necessário entender a definição de inteligência de enxame. A seguir, são apresentadas duas definições para o termo:

   > "Nós utilizamos o termo 'enxame' de uma maneira geral para descrever uma coleção de agentes interagindo de maneira levemente estruturada. Um enxame de abelhas é um exemplo clássico desse tipo de estrutura, a qual pode ser estendida para outros sistemas que possuam uma arquitetura similar. Uma colônia de formigas, por exemplo, pode ser considerada um enxame em que os agentes são formigas, um bando de aves representa um enxame cujos agentes são pássaros (...)" (Eberhart, Shi e Kennedy (2001), tradução nossa).

   > "Inteligência de enxame descreve qualquer tentativa de elaboração de algoritmos ou dispositivos de solução de problemas inspirados a partir do comportamento coletivo de sociedades de insetos ou de qualquer outro animal" (Bonabeau, Dorigo e Theraulaz (1999), tradução nossa).

   Diante do exposto, algoritmos de otimização por inteligência de enxame, assim como os de otimização por colônia de formigas[7] e por partículas[8], são inspirados na movimentação de um corpo em um determinado espaço de pesquisa;

2. **Algoritmos genéticos:** este tipo de algoritmo emprega os conceitos básicos da teoria da evolução Darwiniana e os aplica na otimização de um problema. Dada uma população, composta pelos os indivíduos que estão sendo otimizados, nesse caso geralmente retratados por serviços Web, todos os indivíduos são representados através de uma sequência de *bits*. O problema a ser resolvido é definido e as soluções em potencial são avaliadas por uma função-objetivo. Assim como na teoria Darwiniana, espera-se que a população evolua com o passar das iterações, de modo que se obtenha um ou mais indivíduos mais adequados às

[7] Do inglês, *Ant-Colony Optimization algorithm* – ACO
[8] Do inglês, *Particle Swarm Optimization* – PSO

características avaliadas pela função-objetivo. No caso do tema abordado neste trabalho, o resultado seria composto por uma lista dos melhores serviços Web dentro do repositório;

3. ***Minimal-conflict Hill-Climbing*:** este tipo de algoritmo consiste de um método de busca local que usa uma estratégia de evolução em um estado no espaço de pesquisa. Durante cada iteração, uma pequena perturbação na vizinhança do melhor estado é realizada, resultando em um novo candidato. Caso este seja melhor do que o atual, é selecionado como o novo estado. Caso contrário, um novo ponto, que representa um estado, é escolhido dentro da vizinhança. O algoritmo termina sua execução quando ao menos um dos seguintes critérios é atendido:

   a) Não é possível obter mais nenhuma evolução;

   b) O algoritmo atingiu um determinado número máximo de iterações; ou

   c) O objetivo foi atingido.

4. **Web semântica/ontologias:** estudos que adotam esta estratégia aproveitam-se do conceito de ontologias e Web semântica, previamente descritos no Capítulo 4. O objetivo é pesquisar por serviços através de anotações semânticas, buscando aqueles que mais se enquadram dentro das necessidades da aplicação; e

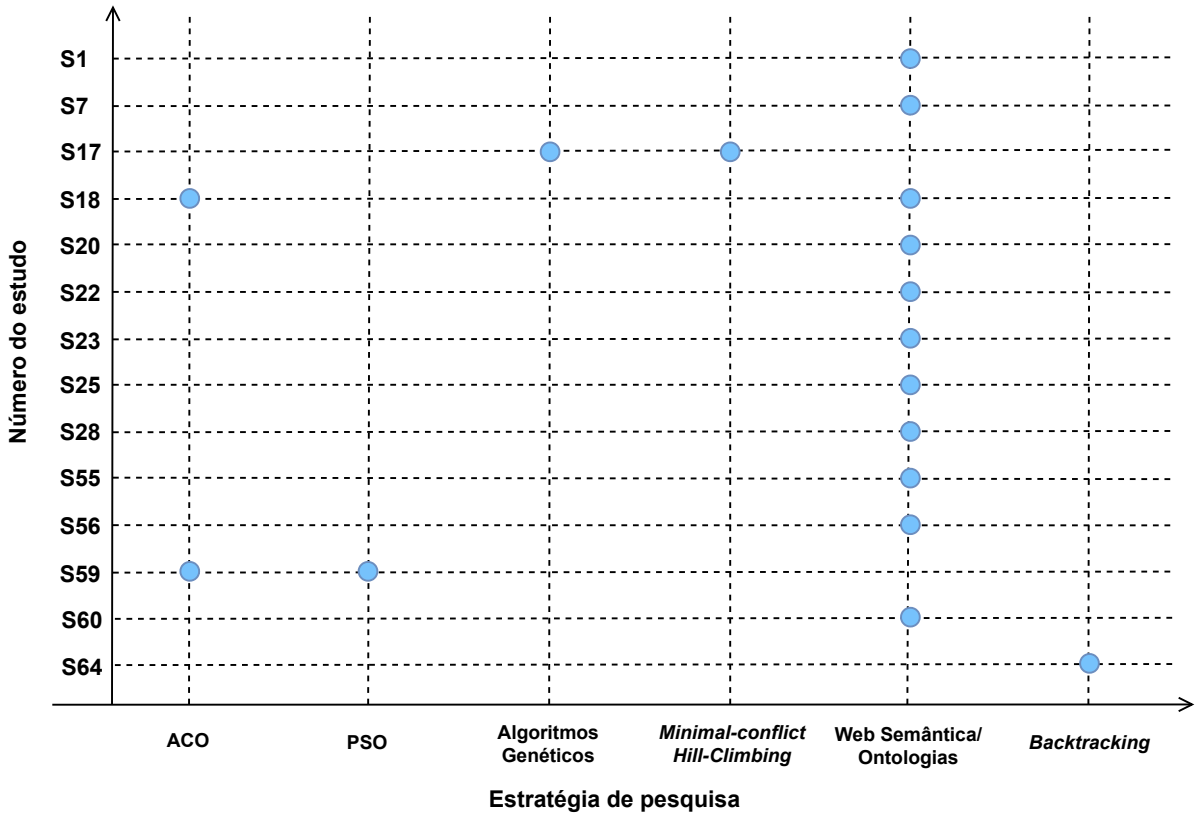
5. **Otimização baseada em *backtracking*:** este tipo de algoritmo atua de maneira recursiva na solução de um problema, dividindo-o em múltiplos subproblemas. Para cada nível de subproblemas, uma busca recursiva em nível pela solução mais apropriada é executada. Caso o algoritmo atinja um estado em que não seja possível gerar novos subproblemas, o nível anterior é assumido como atual e uma outra possível solução é selecionada para que os testes possam ser realizados. O algoritmo de busca por profundidade é um exemplo de algoritmo de *backtracking*.

A Figura 22 mostra as estratégias de busca encontradas no mapeamento. No eixo-X da figura estão dispostos os algoritmos encontrados e no eixo-Y os estudos que utilizam esses algoritmos. A partir da análise da figura pode-se observar que a maior parte dos estudos adotou uma abordagem que implementa uma busca semântica. Isso pode ser justificado a partir da natureza descritiva de serviços semânticos a nível de contexto, facilitando a localização de serviços similares e tornando a pesquisa potencialmente mais rápida e precisa.

QP6: Quais são as estratégias de gerenciamento que estão sendo utilizadas por *frameworks* para *Self-Apps*?

De maneira similar às estratégias de pesquisa adotadas na QP5, é importante identificar qual a abrangência dos *frameworks* selecionados pelo mapeamento com relação a etapa de

Figura 22 – Algoritmos de busca adotados na pesquisa de serviços substitutos



Fonte: Elaborada pelo autor

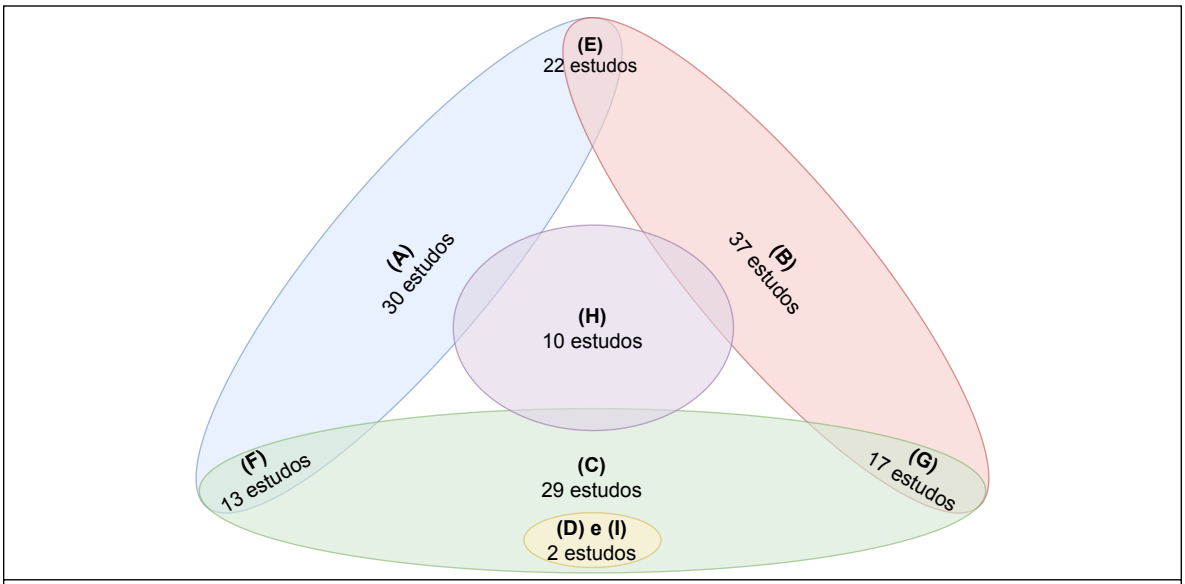
adaptação. A Figura 23 ilustra a relação entre os estudos deste mapeamento e as estratégias de gerenciamento de adaptação. Do conjunto de estudos selecionados, 11 não apresentaram evidências de estratégias de adaptação. Os demais foram agrupados em quatro categorias distintas, a saber:

1. **Composição de serviços:** representa a utilização de um ou mais serviços com a finalidade de compor um novo, o qual deve estar apto a executar uma determinada tarefa;

2. **Descoberta de serviços:** representa a pesquisa por novos serviços em repositório e a coleta de suas respectivas informações. Detalhes sobre como uma conexão deve ser estabelecida e quais métodos estão disponíveis são exemplos das informações recuperadas a partir de uma pesquisa por novos serviços;

3. **Orquestração de serviços:** representa a integração de serviços Web de modo a executar uma tarefa composta por um conjunto de operações que sigam uma determinada ordem. Em resumo, uma orquestração possui um processo mestre, o qual é responsável por coordenar a sequência de chamadas aos serviços que devem ser executados, além de atuar como intermediador na troca de informações entre os serviços envolvidos; e

Figura 23 – Estratégias de gerenciamento de serviços



(A) Descoberta de serviço: S1, S5, S7, S8, S13, S18, S19, S22, S23, S24, S25, S28, S29, S30, S31, S33, S34, S36, S37, S40, S41, S44, S52, S55, S56, S59, S60, S62, S63, S64

(B) Seleção e substituição de serviço: S1, S4, S5, S7, S8, S10, S11, S12, S13, S14, S15, S17, S18, S22, S23, S28, S31, S33, S34, S36, S37, S38, S39, S40, S44, S46, S49, S51, S54, S55, S56, S59, S60, S61, S62, S63, S65

(C) Composição de serviço: S5, S6, S12, S13, S14, S17, S18, S19, S22, S23, S24, S25, S26, S27, S32, S33, S35, S40, S43, S44, S47, S49, S51, S54, S55, S58, S61, S63, S65

(D) Orquestração de serviço: S58, S61

(E) Descoberta de serviço ∩ Seleção e substituição de serviço: S1, S5, S7, S8, S13, S18, S22, S23, S28, S31, S33, S34, S36, S37, S40, S44, S55, S56, S59, S60, S62, S63

(F) Descoberta de serviço ∩ Composição de serviço: S5, S13, S18, S19, S22, S23, S24, S25, S33, S40, S44, S55, S63

(G) Seleção e substituição de serviço ∩ Composição de serviço: S5, S13, S14, S17, S18, S22, S23, S33, S40, S44, S49, S51, S55, S55, S61, S63, S65

(H) Descoberta de serviço ∩ Service Selection/Replacement ∩ Service Composition: S5, S13, S18, S22, S23, S33, S40, S44, S55, S63

(I) Composição de serviço ∩ Orquestração de serviço: S58, S61

Fonte: Elaborada pelo autor

4. **Seleção e substituição de serviços:** representa a seleção e substituição de serviços em tempo de execução. Os serviços são avaliados de acordo com os interesses da aplicação e substituídos por outro similar na ocorrência de falhas ou degradação de QoS. Essa operação deve ser realizada em tempo de execução e sem a percepção dos usuários da aplicação.

QP7: Como *Self-Apps* estão sendo projetados?

O objetivo desta questão foi compreender quais técnicas estão sendo utilizadas para projetar *frameworks* para *Self-Apps*, focando especificamente no monitoramento da aplicação. Como resultado, um conjunto composto por quatro categorias, disponível na Tabela 5, foi elaborado. A seguir é apresentada uma breve descrição dessas categorias, elaborada a partir das definições encontradas nos estudos deste mapeamento.

MAPE-K: como descrito anteriormente na Seção 3.2, trata-se de um laço de controle responsável por: (i) coletar informações de maneira contínua sobre a aplicação que está

Tabela 5 – Técnicas de monitoramento adotadas em *frameworks* para *Self-Apps*

Técnicas	Estudos	Total
MAPE-K	S29, S35, S38, S45, S46, S48, S50, S52, S53	9
Agentes inteligentes	S7, S22, S31, S44, S57	5
Raciocínio baseado em casos	S37, S44	2
Raciocínio baseado em regras	S44	1

sendo monitorada; (ii) analisar seu estado com base em tais informações; (iii) planejar a melhor ação a ser tomada em função de alguma anomalia identificada; e (iv) executar o plano de ação para corrigir tal anomalia. Para isso, essas fases utilizam-se de uma base de conhecimento, que geralmente é formada por um conhecimento inicial fornecido por um especialista e pelo conhecimento adquirido ao longo de sua execução;

Agentes inteligentes: este método pode ser implementado de diferentes maneiras como, por exemplo, baseado em reação a estímulos do sistema ou configurados para cumprir um ou mais objetivos. Tem como principal característica a definição de agentes de software capazes de observar o ambiente em que atuam e, a partir dos dados coletados, tomar decisões e agir;

Raciocínio baseado em casos: este método funciona de maneira similar ao aprendizado de um cérebro humano. Na ocorrência de uma situação de falha, o raciocínio baseado em casos (do inglês, *Case-Based Reasoning* – CBR) tenta identificar em uma base de conhecimento cenários passados iguais ao atual. Em um cenário positivo, a solução encontrada pode ser aplicada ao problema que está sendo analisado. Caso contrário, uma pesquisa por casos similares é realizada, cujo objetivo é construir uma nova solução, a qual será posteriormente inserida na base de conhecimento se apresentar resultado satisfatório; e

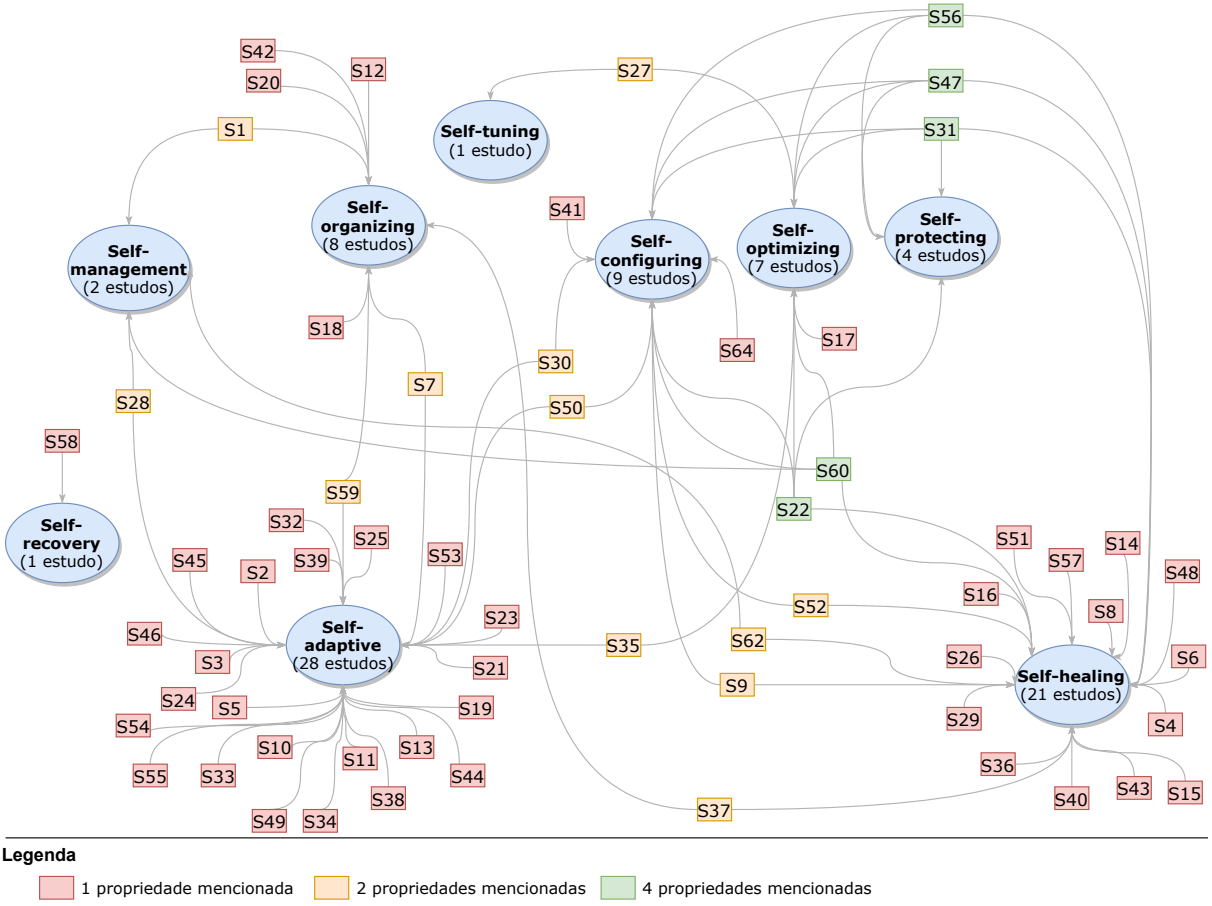
Raciocínio baseado em regras: em uma situação de falha, o raciocínio baseado em regras (do inglês, *Rule-Based Reasoning* – RBR) tenta recuperar instruções de adaptação a partir de uma base de conhecimento pré-estabelecida e, a partir desse conhecimento obtido, segue um conjunto de passos e regras que determinam previamente como a adaptação deve ocorrer.

QP8: Quais propriedades *Self-** estão sendo utilizadas por *frameworks* para *Self-Apps*?

O objetivo desta questão foi identificar quais as propriedades de autoadaptação (do inglês, *self-adaptive properties*) utilizadas por *frameworks* para *Self-Apps*. A Figura 24 mostra a relação entre as propriedades *self-** encontradas nos estudos deste mapeamento. Analisando as

relações dessa figura nota-se que a maior parte dos estudos utiliza apenas uma propriedade, o que pode ser justificado possivelmente pela tarefa árdua e onerosa de implementar um sistema autoadaptativo capaz de operar com mais de uma propriedade (CHENG *et al.*, 2009; SALEHIE; TAHVILDARI, 2009). Por já terem sido apresentados anteriormente no Capítulo 3, detalhes sobre as propriedades reportadas nessa figura não serão apresentados nessa QP.

Figura 24 – Relação entre estudos e propriedades *Self-** identificadas



Fonte: Elaborada pelo autor

5.2 Resultados complementares de bases nacionais

Visando complementar o mapeamento sistemático reportado na Seção 5.1, uma pesquisa complementar em duas bases nacionais de teses e dissertações foi conduzida, a saber: (i) BDTD; e (ii) Catálogo de Teses e Dissertações da CAPES. A primeira base dispõe de uma pesquisa avançada e, portanto, foi possível aplicar uma combinação com grande parte dos termos da *string* de busca apresentada na Tabela 9 (ver Apêndice A). Entretanto, na segunda base encontrou-se apenas a possibilidade de realizar uma pesquisa simples. Portanto, apenas os termos *framework*, *service* e *self-adaptive*, considerados principais, foram inseridos na pesquisa. Como o número de resultados obtido foi expressivamente menor do que o encontrado no mapeamento, os estudos

resultantes da pesquisa nessa base foram manualmente selecionados. Adicionalmente, por se tratarem de bases nacionais, é importante ressaltar que foi necessário adequar alguns termos da *string* de busca para uma versão em português. Tal adequação teve como objetivo garantir que estudos que utilizassem a versão em português dos termos pesquisados não fossem ignorados. Por exemplo, o termo "*self-adaptive*" foi pesquisado em sua forma original e também em sua versão traduzida, "autoadaptativo". Dessa forma, foram obtidos 25 resultados, os quais foram analisados individualmente através de seu resumo e, quando necessário, sua introdução e conclusão. Desse conjunto de dissertações, sete foram selecionadas como tendo uma relação direta com o tema abordado neste trabalho. Diferentemente do que foi apresentado na Seção 5.1, as respostas para o conjunto de questões de pesquisa serão apresentadas individualmente por estudo. Finalmente, como todos os resultados tratam-se de dissertações de programas de mestrado acadêmico, todos apresentam natureza acadêmica (QP1.3). A seguir são apresentadas descrições para seis estudos[9].

Guerra (2007) desenvolveu um modelo de descoberta de serviços com base em Web semântica. A finalidade desse modelo é auxiliar os usuários de uma aplicação a realizar um determinado conjunto de tarefas específicas. O trabalho foca principalmente em computação ubíqua, de maneira que os serviços utilizados pelas aplicações que compõem o ambiente sejam descobertos automaticamente com base na informação contextual do dispositivo que está sendo utilizado e no conjunto de tarefas que o usuário planeja executar. Como forma de avaliação, o autor adota e implementa um estudo de caso de uma biblioteca inteligente, permitindo executar tarefas como, por exemplo: (i) obter uma lista de livros cadastrados como bibliografia de uma disciplina; e (ii) listar quais disciplinas determinado estudante está matriculado.

Já Provensi (2009) teve como foco o desenvolvimento de aplicações móveis e suas necessidades de adaptação, baseando-se nas variações em contexto, tais como: localização do usuário, nível de bateria do dispositivo e conectividade. Além dessas variações, o ambiente em que o usuário se encontra também foi levado em consideração. Por exemplo, o nível de luminosidade do ambiente pode influenciar na vida útil de bateria do dispositivo. A partir disso, o autor discutiu possíveis cenários de adaptação que englobam a descoberta de novos serviços e sua utilização. Também considera adaptações a nível de comunicação como, por exemplo, qual protocolo está sendo utilizado para a troca de dados e se há um outro mais seguro que possa ser adotado. Para validar seu *framework*, o autor propôs um estudo de caso de um quadro branco compartilhado e desenvolveu um protótipo funcional, avaliando o impacto causado em uma aplicação voltada para um cenário de uso real.

Um *framework* para auxiliar no desenvolvimento de aplicações móveis foi proposto por Guimarães (2012), que visa apoiar o desenvolvimento de mecanismos que lidam com

[9] O estudo de Neto (2010) foi recuperado durante o mapeamento da literatura (S44) e compôs o conjunto de dados apresentados na Seção 5.1. Portanto, optou-se por omiti-lo nessa seção, sendo esse o motivo pela discussão de apenas seis dos sete estudos obtidos.

disponibilidade e interoperabilidade em conexões entre os aplicativos e os serviços Web. Seu *framework* se apoiou no conceito de agentes e adotou o modelo MAPE-K para atuar na tarefa de adaptação, de maneira que o dispositivo móvel em que o sistema está sendo executado é composto por agentes, os quais realizam a comunicação com os serviços Web. Como parte da avaliação foram detalhados dois estudos de caso.

Em Oliveira (2014) é possível observar um interesse maior voltado para a composição de serviços, mais especificamente na seleção de qual conjunto de serviços fará parte de uma composição. A autora utilizou um *framework* existente, denominado *OpenKnowledge*, e implementou um mecanismo cujo objetivo é atuar na seleção de serviços Web em ambientes coreografados considerando determinados atributos de QoS, como, por exemplo, estimativas de atraso, taxa de perda, uso de CPU, entre outros. Para isso, foram implementados algoritmos seletores, os quais recebem dados de QoS de um conjunto de serviços e estabelecem uma pontuação para cada um deles com base nesses dados recebidos. O serviço que apresentar melhor pontuação será o escolhido. Para avaliar o trabalho proposto foram conduzidos experimentos em diferentes cenários, considerando estimativa de atraso, perda de pacotes e utilização de CPU como parâmetros para a seleção.

Mendonça (2015) também abordou o assunto de composição de serviços, focando em especial na elaboração de coreografias de execução de serviços. Segundo o autor, é importante monitorar parâmetros de QoS em coreografias devido à natureza distribuída do fluxo de informações que passam por esses serviços e de seus dados de controle. Como em uma coreografia não existe um processo mestre responsável por acompanhar a execução das tarefas, cada serviço opera de maneira independente, ou seja, não há um ponto central de falha. Baseado nisso, o autor relata o desenvolvimento de uma camada intermediária (do inglês, *middleware*) com base no modelo MAPE-K. O objetivo desse *middleware* é gerenciar as coreografias de serviço por meio de uma atividade de monitoramento de QoS, onde os serviços que apresentem falha ou degradação de QoS podem ser substituídos. O trabalho foi avaliado por meio de um experimento utilizando composição de serviços simples para validar o desempenho da solução proposta pelo autor.

Por fim, Junior (2015) descreveu a arquitetura de um *framework* chamado SASeS (do inglês, *Self-Adaptive Service-based Systems*), cujo objetivo é facilitar a criação de aplicações autoadaptativas baseadas em serviços. O autor baseou-se em padrões de serviços Web e de projeto que suportem a execução de tarefas comuns no desenvolvimento de um sistema. Segundo o autor, a partir do *framework* SASeS é possível criar um ou mais componentes de adaptação, os quais selecionam e informam a lista de serviços que devem ser utilizadas pela aplicação, a qual pode ser modificada em tempo de execução de acordo com os requisitos do software. Para isso, propôs uma abordagem baseada no modelo MAPE-K para monitorar um sistema de software e efetuar as adaptações necessárias em tempo de execução. Para validar sua proposta, propôs

dois estudos de caso: (i) um sistema de descoberta de serviços de um aeroporto, cujo objetivo é verificar os sistemas disponíveis no ambiente e apresentar ao usuário apenas os que estiverem disponíveis; e (ii) um serviço de notícias, responsável por oferecer conteúdo multimídia à seus clientes.

5.3 Considerações finais

Neste capítulo foram apresentados os resultados obtidos a partir de pesquisas na literatura por trabalhos relacionados ao tema "*frameworks para Self-Apps*". A partir da análise do mapeamento sistemático da literatura e da pesquisa complementar em bases nacionais de teses e dissertações foi possível observar um interesse nessa área de pesquisa por diversos pesquisadores, reforçado tanto pela existência de estudos entre os anos de 2002 até o presente momento (2020) quanto pela distribuição de estudos entre vários países e universidades. O número de questões de pesquisa elaboradas e a possibilidade de se obter respostas variadas para cada uma das questões é um indicador de que esse tema possui grande abrangência, contemplando diferentes áreas da computação, tais como: engenharia de software, inteligência artificial, computação inspirada pela natureza, redes, entre outras. As diferentes respostas para algumas das QPs apresentadas neste capítulo (ver Apêndice A), como, por exemplo, atributos de QoS mais utilizados e as estratégias de pesquisa e de gerenciamento de serviços mostram que, embora existam caminhos mais usualmente adotados, não há nada consolidado como um padrão, existindo ainda questões em aberto nessa área que apresentam potencial para evoluir através da condução de novas pesquisas.

Finalmente, a condução de um mapeamento em diferentes momentos permitiu obter uma visão compreensiva do estado da arte atual da área de pesquisa deste projeto de mestrado, além de permitir acompanhar sua evolução durante o desenvolvimento do mesmo. As respostas apresentadas para cada QP nos permite confirmar a viabilidade de algumas das abordagens adotadas no desenvolvimento do *framework* DynaMS proposto neste trabalho, além de refletir sobre as possíveis evoluções e contribuições resultantes do mesmo.

6 Apresentação do *framework* DynaMS

O objetivo deste capítulo é apresentar os detalhes do *framework* DynaMS, proposto neste projeto de mestrado acadêmico. Esse *framework* reúne experiências anteriores de nosso grupo de pesquisa (AFFONSO; NAKAGAWA, 2013; PASSINI; AFFONSO, 2018b; PASSINI; AFFONSO, 2018a; AFFONSO; PASSINI; NAKAGAWA, 2019) e os resultados de uma revisão da literatura, a qual foi apresentada em detalhes no Capítulo 5. Em resumo, o *framework* DynaMS visa apoiar o desenvolvimento de aplicações que requerem características de autoadaptação e sejam baseadas em serviços simples e/ou compostos. Para isso, esse *framework* viabiliza os seguintes recursos: (i) técnicas de implantação dinâmica de serviços Web; (ii) métricas de QoS para avaliação de tais serviços; e (iii) busca semântica baseada em critérios de similaridade por parâmetros técnicos, dados semânticos, e a combinação das duas anteriores. Além disso, vale destacar que o *framework* DynaMS foi projetado para atuar em duas fases: *design* e *runtime*. Na fase de *design*, os engenheiros de software fazem o projeto da aplicação, selecionando os serviços que irão compor a mesma de acordo com interesses referentes aos atributos de qualidade do domínio de atuação. Para viabilizar o projeto da aplicação, o *framework* DynaMS disponibiliza uma estrutura dinâmica que permite criar os serviços compostos das aplicações, os quais são organizados em uma lista de serviços preferenciais. Em seguida, para cada serviço preferencial, uma outra lista de serviços alternativos, similares aos preferenciais, deve/pode ser elaborada. Na fase de *runtime*, a aplicação é monitorada e os parâmetros de QoS são constantemente analisados. Nesse sentido, em caso de falha ou degradação de QoS, um serviço preferencial é substituído por um alternativo, evitando que a aplicação se torne indisponível ou de execução inviável. Para mostrar o projeto do *framework*, este capítulo foi organizado da seguinte maneira: na Seção 6.1 é apresentado o levantamento de requisitos realizado, o qual teve como fonte de informação os trabalhos anteriores de nosso grupo de pesquisa supracitados e o mapeamento da literatura apresentado no Capítulo 5. Em seguida, detalhes de projeto do *framework* são reportados na Seção 6.2. Finalmente, na Seção 6.3 são detalhadas as duas soluções de infraestrutura auxiliares utilizadas pelo *framework* DynaMS, a saber: (i) sistema de monitoramento; e (ii) repositório de dados semânticos sobre serviços Web.

6.1 Levantamento de requisitos

Antes de apresentar os requisitos do *framework* DynaMS, duas considerações devem ser ressaltadas. A primeira está relacionada ao entendimento do domínio de atuação que esse *framework* visa apoiar, ou seja, aplicações baseadas em serviços que requerem características de autoadaptação. A segunda está relacionada ao modo de captura de requisitos para software

autoadaptativo, uma vez que o desenvolvimento desse tipo de software não é uma tarefa trivial quando comparado a um software tradicional (SALEHIE; TAHVILDARI, 2009). Diante desse cenário e visando contemplar as considerações supracitadas, Psaier e Dustdar (2011a) e Angarita *et al.* (2016) propuseram um modelo de estados para SaS que implementa a propriedade *self-healing*. Esse modelo foi adaptado para o contexto deste trabalho, passando a atender também a evolução do sistema. Em resumo, a evolução dos sistemas de software autoadaptativos pode ser considerada uma questão especial, pois tais sistemas visam capturar mudanças no ambiente de execução ou em si mesmo durante seu ciclo de vida. Em linhas gerais, o ambiente operacional de um sistema de software autoadaptável pode sofrer variações ao longo do tempo. Assim, o sucesso da autoadaptação depende da distinção entre estados do sistema, conforme ilustrado na Figura 25. Esse conjunto de estados permite delimitar por qual tipo de situação um sistema está passando e quais seus possíveis estados futuros. Para isso, o serviço é constantemente monitorado e seu estado de execução atual deve ser identificado. A partir dessa figura também é possível observar o conjunto de propriedades de autoadaptação vinculadas ao modelo de estados que indicam necessidade e viabilidade de adaptação. Por exemplo, situações que envolvam a degradação do serviço estão associadas com propriedades como *self-repair*, *self-healing* e *self-recovery*. Por outro lado, situações que envolvam evolução do serviço estão relacionadas com propriedades como *self-organization, self-optimization* e *self-tuning*. A seguir é apresentada uma descrição para cada um dos estados de adaptação:

1. **Normal:** representa o sistema em um estado de estabilidade, indicando que nenhuma falha foi detectada e a aplicação está executando de acordo com os parâmetros de QoS esperados;

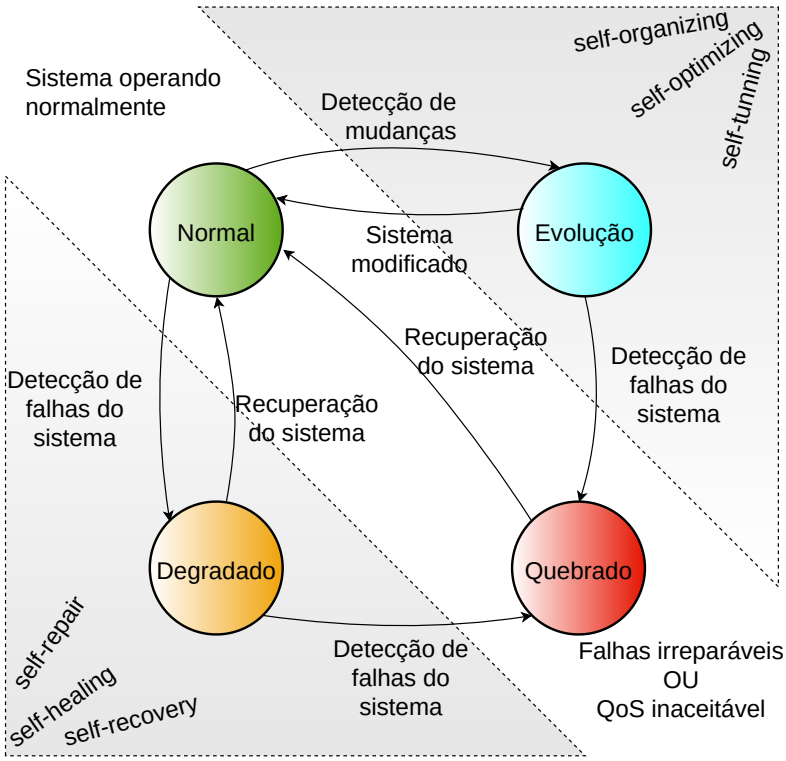
2. **Degradado:** representa a presença de alguma inconsistência ou variação em um ou mais níveis de qualidade de serviço da aplicação;

3. **Quebrado:** representa a ocorrência de uma falha mais séria que não pode ser resolvida de maneira automática pelo módulo de adaptação, sendo necessário que ocorra intervenção humana para que a aplicação volte ao estado normal de execução; e

4. **Evolução:** representa as mudanças efetuadas na aplicação para atender aos requisitos do usuário ou do ambiente de execução. É importante ressaltar que caso uma operação de adaptação não seja concluída com sucesso a aplicação pode ir diretamente para o estado "Quebrado".

Para auxiliar na captura de requisitos para SaS, Salehie e Tahvildari (2009) reportam o modelo 5W1H como uma alternativa factível que tem sido utilizada nesse tipo de sistema. Esse modelo permite especificar exatamente qual o tipo de adaptação que se pretende atingir com

Figura 25 – Possíveis estados de execução de serviços autoadaptativos



Fonte: Adaptado de Ghosh *et al.* (2007 apud PSAIER; DUSTDAR, 2011b)

o sistema, além de detalhes sobre como obtê-la. Portanto, com o intuito de expor o nível de adaptação proposto e seu plano de execução, as questões que compõem o modelo 5W1H e suas respectivas respostas são apresentadas a seguir:

***[What?]* O que será adaptado?** A adaptação ocorrerá na composição do serviço. Quando um (ou mais) serviço(s), que faz(em) parte de uma composição, apresentar(em) falha(s) ou degradação de QoS, este(s) serão substituído(s) por outro(s) similar(es);

***[Where?]* Onde a adaptação ocorrerá?** A adaptação deve ocorrer na composição de serviço, de maneira que serviços que fazem parte da composição e apresentem algum tipo de falha ou degradação de QoS sejam substituídos;

***[Who?]* Quem realizará a adaptação?** O processo de adaptação deverá ser executado por um sistema de adaptação, o qual receberá dados coletados por um sistema de monitoramento. Ao identificar um cenário de adaptação, tal sistema utilizará o repositório de serviços e será auxiliado pelo sistema de monitoramento para realizar uma análise dos parâmetros de QoS dos serviços candidatos a substituição, fornecendo uma maneira de selecionar aquele que melhor se enquadra nas necessidades do serviço;

[When?] **Quando a adaptação será realizada?** Uma adaptação deve ser realizada sempre que for detectado um cenário de falha ou degradação de QoS de um ou mais serviços;

[Why?] **Porque a adaptação será realizada?** O objetivo principal da adaptação é garantir que a aplicação não seja interrompida ou apresente instabilidades pela degradação de QoS; e

[How?] **Como a adaptação será realizada?** Um novo serviço compatível deverá ser selecionado e as requisições devem ser encaminhadas para tal serviço, sem a necessidade de interrupção da aplicação principal.

De acordo com Salehie e Tahvildari (2009), as questões do modelo 5W1H devem ser respondidas em duas fases: *design* e *runtime*. A primeira visa apoiar o desenvolvimento de um software autoadaptativo novo (do zero) ou reengenharia de um legado. Os interessados, representados por engenheiros de software e projetistas, extraem os requisitos com base nas questões do modelo, estabelecendo mecanismos e alternativas a serem usadas na fase operacional. A segunda visa gerenciar as preocupações do ambiente de execução para responder adequadamente a alterações no ambiente/contexto de um aplicativo de software. Em resumo, é nessa fase que a adaptação ocorre, evidenciando as decisões de projeto como abordagem e tipo de adaptação escolhido na fase de *design*.

Com base no modelo de estado para serviços (Figura 25) e no modelo 5W1H expostos nesta seção, os requisitos para o projeto do *framework* DynaMS foram elaborados. Nesse sentido, vale enfatizar que tais requisitos foram baseados em experiências anteriores de nosso grupo de pesquisa (AFFONSO; NAKAGAWA, 2013; PASSINI; AFFONSO, 2018b; PASSINI; AFFONSO, 2018a; AFFONSO; PASSINI; NAKAGAWA, 2019) e nos resultados do mapeamento sistemático apresentado no Capítulo 5. A seguir, os requisitos identificados são apresentados:

1. O *framework* deve possibilitar a utilização de serviços SOAP e RESTful. O intuito desse requisito é garantir sua compatibilidade com os dois modelos de comunicação mais comumente adotados no desenvolvimento de serviços Web, garantindo flexibilidade e versatilidade ao *framework*;

2. O *framework* deve suportar o estabelecimento de uma composição de serviços, garantindo a possibilidade de criar novos serviços mais complexos a partir da utilização de serviços mais simples. Nesse processo de composição, em algumas situações pode ser necessário também que o serviço resultante da composição seja executado a partir de uma sequência de passos bem definida. Portanto, é necessário que o *framework* disponha também da capacidade de estabelecer uma orquestração de serviços, garantindo que tal sequência de passos seja obedecida, quando necessário;

3. O *framework* deve suportar a busca de serviços com base em suas especificações técnicas, semânticas e híbrida, sendo esta última uma combinação das duas anteriores. Esta estratégia de busca foi a que apresentou maior recorrência no mapeamento sistemático conduzido (Questão de Pesquisa 5 – QP5, Seção 5.1), evidenciando ser uma abordagem comumente utilizada e bem aceita pela comunidade de pesquisadores dessa área;

4. O *framework* deve permitir a customização de pesos para diferentes atributos de QoS. Não é possível definir um cenário único de requisitos de implantação a ser adotado pelas aplicações desenvolvidas com base no *framework* proposto. Nesse sentido, é necessário garantir que os desenvolvedores disponham da capacidade de especificar quais atributos consideram mais importantes para a boa execução de sua aplicação. Tal escolha impactará diretamente na classificação de serviços candidatos à substituição e, consequentemente, quais serviços serão recomendados; e

5. O *framework* deve permitir adaptação em tempo de execução, monitorando os serviços constantemente e agindo rapidamente em um cenário que envolva a necessidade de substituição de serviço. Espera-se que a adaptação ocorra rapidamente, sem a necessidade de intervenção humana e que seja preferencialmente imperceptível ao usuário.

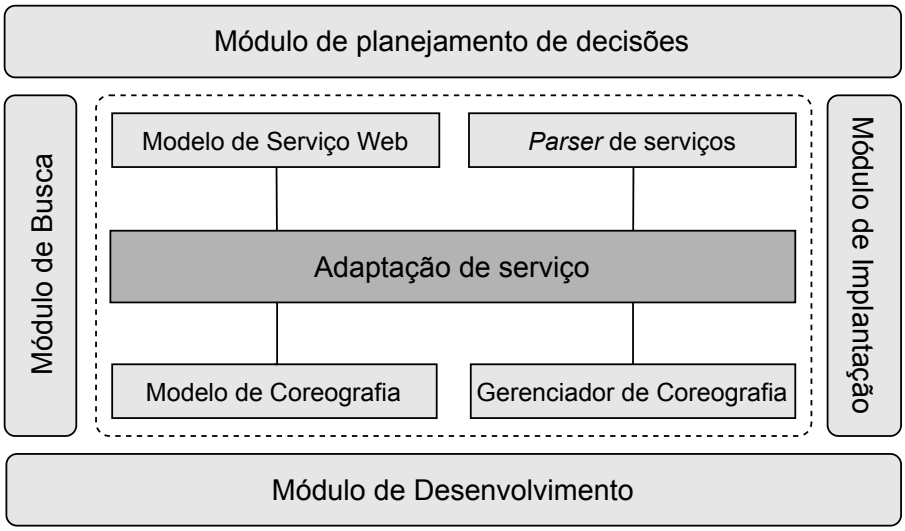
6.2 Projeto do *framework*

A Figura 26 mostra uma visão geral do *framework* DynaMS, proposto neste projeto de mestrado acadêmico. Em linhas gerais, esse *framework* foi organizado em um núcleo de adaptação (linha tracejada) e quatro módulos adicionais, a saber: (i) planejamento de decisões; (ii) implantação; (iii) busca; e (iv) desenvolvimento. A seguir, Seções 6.2.1 a 6.2.5, são apresentados os detalhes de cada componente desse *framework*.

6.2.1 Núcleo de adaptação

O núcleo de adaptação pode ser considerado como a principal estrutura do *framework* DynaMS, sendo responsável por gerenciar os detalhes referentes a execução de um serviço. É a partir dele que a representação de um serviço Web, a composição de serviços e suas respectivas coreografias podem ser configuradas. De maneira geral, essa estrutura está também organizada através de um conjunto de módulos, cada qual com sua respectiva função. A Figura 39 (ver Apêndice C) mostra a versão completa do diagrama de classes do *framework*. Nesse diagrama são apresentadas as classes que compõem o metamodelo utilizado para representar serviços, coreografias, *parser* de serviços e o cliente utilizado para a chamada desses serviços. O diagrama também detalha como esse conjunto de classes se relaciona. Para facilitar a apresentação de cada um dos componentes da estrutura por trás do núcleo de adaptação, partes desse diagrama

Figura 26 – Visão geral do sistema de reconhecimento e adaptação e de seus módulos

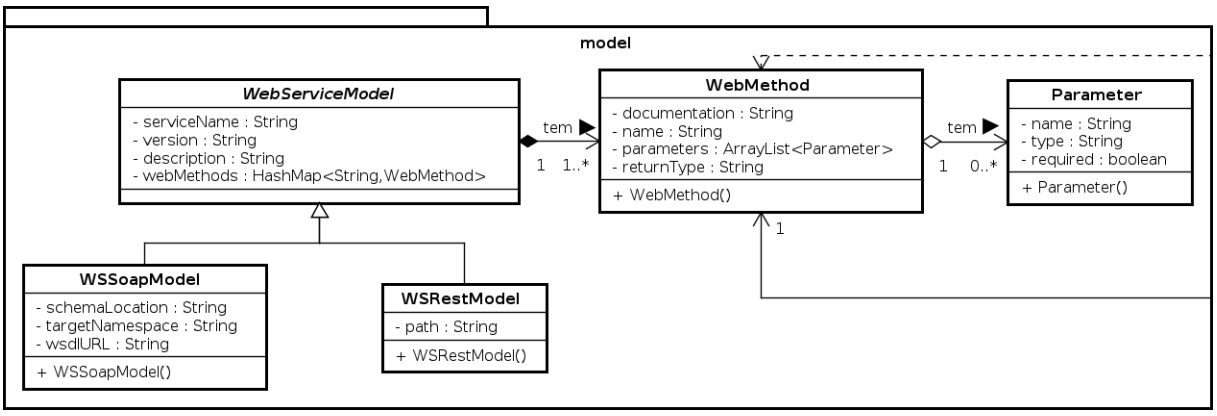


Fonte: Elaborada pelo autor

de classes são ilustradas a seguir durante a apresentação de cada um dos componentes dessa estrutura, a saber:

Modelo de serviço Web. Representa o metamodelo para serviços Web do tipo SOAP e REST no contexto do *framework* DynaMS. Como ilustrado na Figura 27, tal metamodelo é composto por cinco classes distintas, a saber:

Figura 27 – Metamodelo de serviço Web.



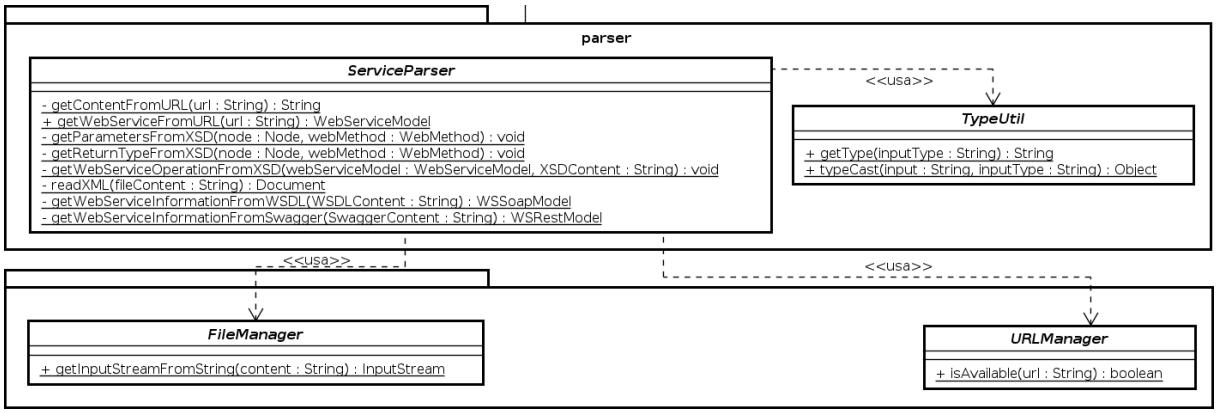
Fonte: Elaborada pelo autor.

1. `Parameter`: unidade mais simples da representação de um serviço Web, armazena as informações (atributos `name` e `type`) de um parâmetro. Adicionalmente, especifica através de um valor booleano (atributo `required`) se seu preenchimento é necessário para a execução do método em que o respectivo parâmetro se encontra;

2. `WebMethod`: representa um método disponibilizado por um serviço Web. É responsável por armazenar uma documentação descritiva sobre o objetivo daquele serviço (atributo `documentation`), seu nome (atributo `name`), o conjunto de parâmetros (atributo `parameters`) necessários para invocá-lo e qual será o tipo de informação retornada (atributo `returnType`); e
3. `WebServiceModel`: representa um nível de hierarquia de classes superior, contendo informações comuns (atributos `serviceName`, `version`, `description` e `WebMethods`) aos tipos de serviços suportados pelo *framework* DynaMS. As classes de nível de hierarquia inferior (`WSSoapModel` e `WSRestModel`) armazenam informações complementares e exclusivas aos serviços que representam.

Parser de serviços. No contexto de autoadaptação proposto neste trabalho, uma boa documentação das capacidades de cada serviço é um elemento fundamental para que a atividade de descoberta de serviços em repositórios possa ser executada. É a partir dessa documentação que *endpoints* de acesso podem ser obtidos, assim como os dados que precisam ser fornecidos para que a API (do inglês, *Application Programming Interface*) possa funcionar adequadamente. Ilustrado na Figura 28, o *parser* de serviços tem como objetivo coletar essas informações, transformando um serviço em um objeto do modelo de serviço Web utilizado pelo *framework* DynaMS. Tal transformação torna possível a posterior utilização do respectivo serviço através do cliente de comunicação implementado pelo *framework*, facilitando o processo de substituição de serviço em tempo de execução. Para realizar essa transformação é necessário estabelecer processos específicos para os diferentes tipos de serviços envolvidos. Serviços baseados em SOAP possuem um documento WSDL criado por padrão durante sua implantação, o qual é disponibilizado através de uma URL ou um arquivo. Como discutido anteriormente na Seção 2.2, esse tipo de documento é responsável por armazenar informações pertinentes a um serviço e suas respectivas operações. Para a

Figura 28 – Classes que compõem o *parser* de serviço Web.



Fonte: Elaborada pelo autor.

extração de informações desse documento foi utilizada a API DOM (do inglês, *Document Object Model*), cujo propósito é ler um documento XML, formato padrão utilizado por arquivos WSDL. Essa leitura produz como resultado o conteúdo obtido do arquivo estruturado em uma árvore, onde cada nó é composto pelas *tags* do documento XML lido. Do *parser* de serviços destaca-se o método `getWebServiceInformationFromWSDL(...)` da classe `ServiceParser` (Figura 28). Seu objetivo é percorrer a árvore gerada pela API, pesquisando pelo conjunto de *tags* do documento XML lido e extrair os dados que irão compor o modelo de serviço Web. A seguir, Algoritmo 1, é apresentado um pseudocódigo que demonstra o comportamento padrão do *parser* WSDL. Em síntese, esse algoritmo recebe uma URL ou um arquivo WSDL como entrada e retorna uma instância da classe `WebService` (`parserManager.model.WebService`).

Algoritmo 1 Algoritmo do reconhecedor de serviços SOAP

```
1:  função ParseWSDL(WSDL)                ▷ WSDL é um arquivo WSDL ou uma URL
2:      Ler conteúdo do arquivo WSDL
3:      Estruturar conteúdo lido em uma árvore
4:      se WSDL ⊃ schema então
5:          Ler schema
6:          Estruturar conteúdo lido em uma árvore
7:          Percorrer a árvore gerado procurando por informações do serviço
8:          ModeloWSDL ← InformaçõesDoServiço
9:      senão
10:         Pesquisar por operações de serviço no arquivo WSDL
11:         ModeloWSDL ← InformaçõesDoServiço
12:     fim se
13: retorna ModeloWSDL   ▷ Dados do arquivo WSDL representados na classe WebService
14: fim função
```

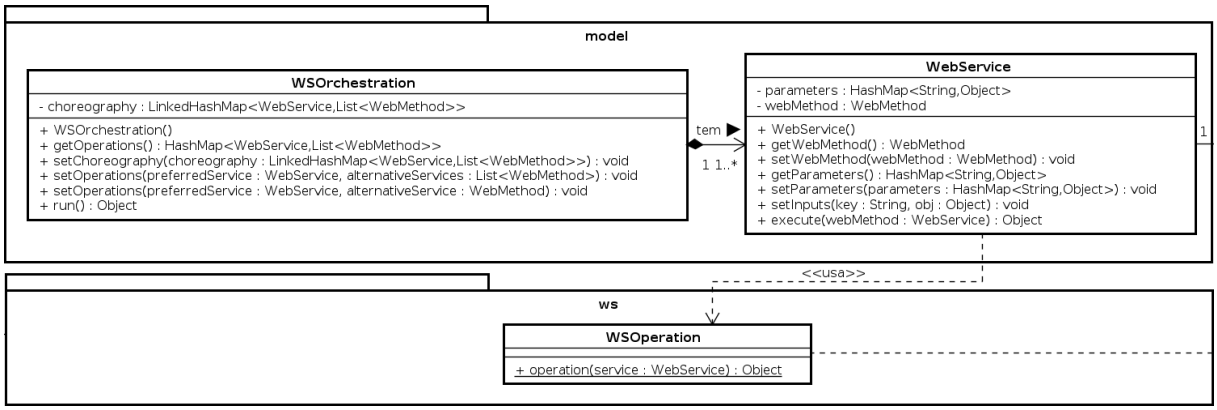
Diferentemente do que é observado em SOAP, serviços RESTful não possuem um documento nativo para descrever suas operações. Entretanto, é possível encontrar implementações de ferramentas que se propõem a documentá-los. Nesse sentido, durante o desenvolvimento deste trabalho, optou-se por adotar o Swagger[1], um projeto *open source* composto por algumas ferramentas que auxiliam o desenvolvimento de APIs REST. Tal escolha foi motivada principalmente pela possibilidade de utilizar a ferramenta como um meio de gerar a documentação automaticamente através de anotações no código fonte, o que pode otimizar consideravelmente o processo de desenvolvimento. Adicionalmente, a facilidade de disponibilizar a documentação de uma API simultaneamente através de um *endpoint* e de uma interface Web, possibilitando testar a comunicação com os serviços desenvolvidos diretamente através de uma interface visual, foram dois outros pontos que justificaram a escolha dessa ferramenta. O *endpoint* responsável por

[1] Disponível em <https://swagger.io/>

fornecer a documentação produz um arquivo JSON, o qual pode ser facilmente obtido e transformado no modelo de serviços utilizado pelo *framework* DynaMS. Similarmente ao *parser* de arquivos WSDL apresentado na Figura 28, foi desenvolvido um método `getWebServiceInformationFromSwagger(...)`, cujo objetivo é acessar o *endpoint* de uma determinada API e extrair as informações relevantes para a criação de instâncias do modelo de serviço Web. Embora a coleta das informações de um serviço RESTful ocorra por meio da leitura de um arquivo JSON, seu funcionamento é análogo ao do pseudocódigo mostrado no Algoritmo 1, cujo objetivo é realizar a leitura da documentação de um serviço padronizado em JSON, extrair as informações necessárias e transformá-las em instâncias dos objetos representados pelo modelo de serviço Web.

Modelo de coreografia. O modelo de coreografia ilustrado na Figura 29 representa uma tarefa a ser executada. Para isso, esse modelo armazena quais serviços estabelecem uma composição, além da ordem de execução dos mesmos.

Figura 29 – Classes que compõem o modelo e o cliente de uma composição de serviços.



Fonte: Elaborada pelo autor.

A classe `WSOrchestration` é o ponto de partida para a criação de uma nova coreografia. Essa classe é composta apenas pelo atributo `choreography`, que armazena um conjunto de objetos da classe `WebService`. O atributo `choreography` deve, para cada operação, armazenar uma tupla contendo a operação de um serviço preferencial e seu conjunto de parâmetros de entrada. Objetos da classe `WebService` armazenam o conjunto de parâmetros de entrada necessários para a execução da tarefa e os dados do método que será executado. Também permite armazenar uma lista alternativa com diferentes instâncias de cada operação, localizadas em serviços distintos. Essas listas poderão ser utilizadas imediatamente em um cenário que necessite de adaptação caso o serviço definido como preferencial apresente falha ou degradação de QoS. Além disso, vale destacar que a lista de serviços alternativos pode ser manipulada em duas etapas distintas, a saber:

1. **Fase de *design*:** momento em que os serviços similares aos preferenciais são identificados, via pesquisa no repositório de serviços, e inseridos nessa lista. Para isso, cada serviço é analisado manualmente pelo desenvolvedor quanto aos parâmetros de entrada, tipo de retorno e propósito de execução.
2. **Fase de *runtime*:** momento em que a aplicação já foi implantada no ambiente de execução. A realização dessa pesquisa tem como objetivo garantir que, em um cenário de falha ou degradação de QoS, seja possível efetuar a substituição de serviços sem a percepção dos clientes da aplicação. Para isso, ao verificar a lista de serviços alternativos, duas possibilidades podem ocorrer: (i) a lista está vazia e uma pesquisa automática, levando em consideração os critérios de similaridade, deve ser realizada para tentar identificar ao menos um serviço compatível; e (ii) a lista possui um ou mais serviços similares e o primeiro passa a assumir o papel do serviço preferencial.

Finalmente, é necessário manter a ordem de execução conforme a sequência de inserção dos serviços na coreografia, pois podem existir situações em que o conjunto de parâmetros de entrada depende do resultado produzido por um serviço executado anteriormente. Para atender a esse conjunto de requisitos optou-se por utilizar uma lista encadeada de `HashMap`, a qual preserva a ordem de inserção e, ao mesmo tempo, possui tempo de acesso de complexidade `O(1)` para atividades de inserção e busca. A informação referente ao serviço principal a ser executado define a chave da `Hash`, a qual armazena os dados de uma execução em cenário normal de operação e é complementada por uma lista adicional (valores da `Hash`). Tal lista contém um conjunto de objetos da classe `WebMethod`, representando uma lista de serviços já analisados durante a concepção da coreografia ou adicionados posteriormente em tempo de execução e considerados aptos para, em um cenário que exija uma adaptação, cumprir a mesma tarefa executada pelo serviço definido como preferencial.

Gerenciador de coreografia. Este componente, implementado através do método `run()` da classe `WSOrchestration` (Figura 29), é responsável por atuar como um "orquestrador" em uma coreografia. Esse método garante que todos os serviços envolvidos sejam chamados em sua respectiva ordem e recebam todas as informações necessárias para uma execução correta do conjunto de passos determinado no modelo de coreografia. Ao término da execução, o resultado é retornado para o responsável pela requisição, ou seja, usuário ou algum subsistema da aplicação.

Adaptação de serviço. Trata-se da união de todas as atividades desempenhadas pelo *framework*, resultando no processo de adaptação proposto. Este componente é responsável por coordenar as atividades realizadas pelos módulos supracitados, agindo de maneira similar à de um "orquestrador". A Figura 30 mostra o conjunto de passos para um cenário de uma

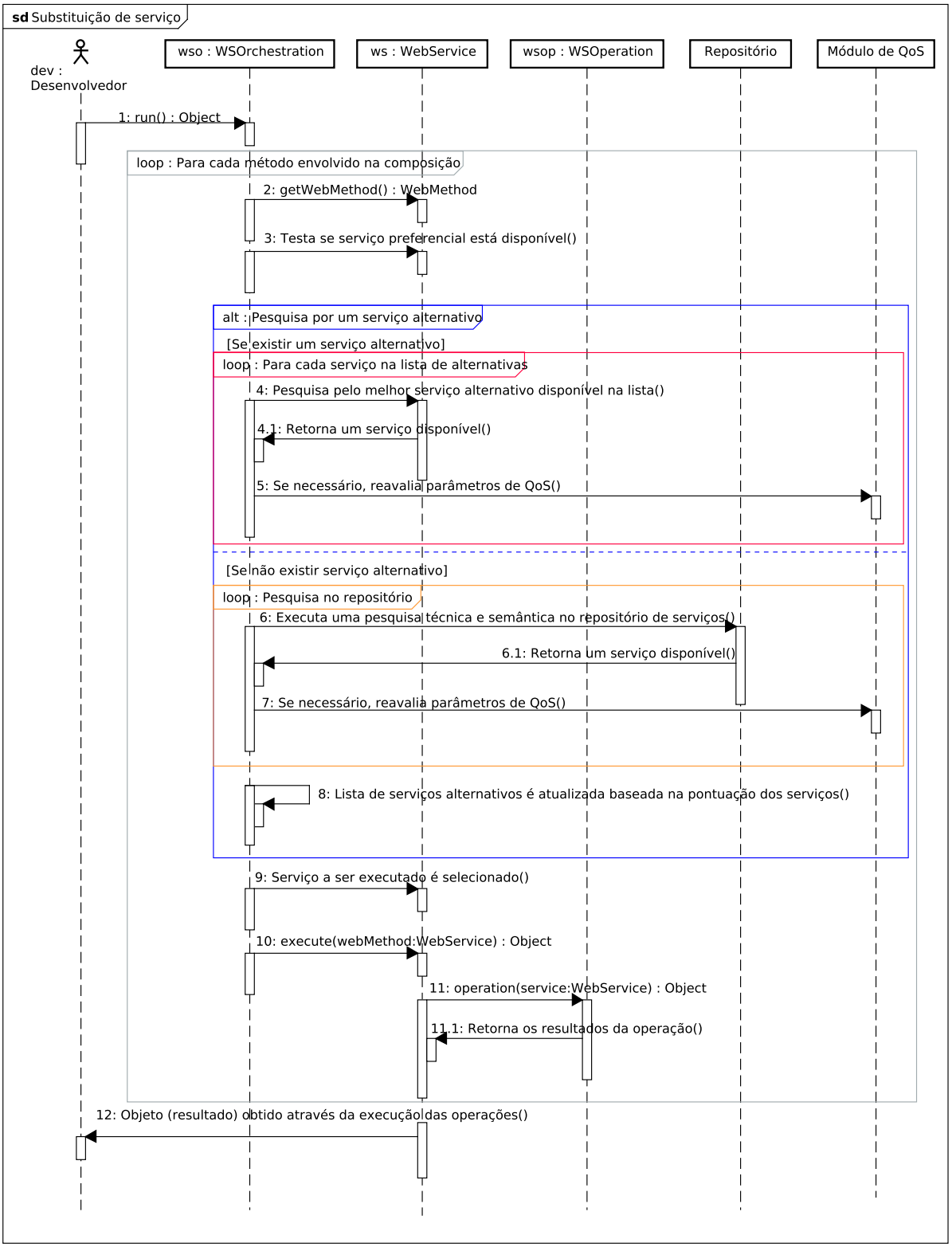
adaptação que necessite de uma substituição de serviços. Em síntese, a execução de uma coreografia (passo 1) envolve um laço de repetição responsável por obter as informações de todos os métodos que serão executados e, para cada um, testar se o serviço preferencial associado com aquele método está disponível (passos 2 e 3). Caso não sejam encontrados erros, a operação da coreografia é executada (passos 9 e 10) e os resultados são devolvidos ao gerenciador de coreografias para que possam ser encaminhados ao responsável pela requisição (passos 11 e 12). Caso contrário, uma pesquisa por um serviço similar é realizada na lista de serviços alternativos (passo 4) e/ou no repositório (passo 6). Caso essa pesquisa resulte em ao menos um resultado, um serviço alternativo é selecionado e a execução da aplicação continua com a adaptação realizada. Porém, caso nenhum serviço seja encontrado, a execução da aplicação encerra com uma mensagem de erro.

6.2.2 Módulo de planejamento e tomada de decisões

O módulo de planejamento e tomada de decisões tem por objetivo monitorar o comportamento dos serviços no ambiente de execução, auxiliando no processo de adaptação. Basicamente, esse módulo engloba experiências anteriores de nosso grupo de pesquisa, a saber: (i) um *framework* baseado em técnicas de aprendizado de máquina e no *loop* de controle MAPE-K para tomada de decisão em SaS (AFFONSO *et al.*, 2015). Esse *framework* foi desenvolvido para apoiar a arquitetura de referência RA4SaS (AFFONSO; NAKAGAWA, 2013); e (ii) uma ferramenta para modelagem de sistemas para tomada de decisão baseada em tal *framework* (AFFONSO; LEITE; NAKAGAWA, 2016). Essas duas contribuições fornecem uma solução factível para modelagem de sistemas dessa natureza, permitindo classificar e analisar dados sensoreados para detectar autonomamente e mitigar anomalias em tempo de execução. Assim, no contexto deste projeto de mestrado, o *framework* de tomada de decisão supracitado foi incorporado a este módulo para lidar com a interpretação de mudanças e proposição de soluções que englobam tanto o ambiente de execução quanto os serviços Web nele inseridos. Além disso, vale ressaltar que este módulo opera em paralelo ao sistema de monitoramento de QoS apresentado na Seção 6.3.2.

No que diz respeito ao processo de interpretação de contexto, os dados coletados no ambiente de execução e serviços Web são analisados e, caso sejam observados nessas análises indicativos sobre a presença de algum tipo de anomalia como falha de execução e/ou degradação de QoS, um procedimento de adaptação pode ser conduzido. Dessa forma, tal procedimento resulta na substituição em tempo de execução do serviço com anomalia por outro "saudável". Assim, em cenários que apresentem um conjunto de serviços candidatos capazes de atender a necessidade de substituição, potencialmente localizados em diferentes servidores, é fundamental que os mesmos sejam apresentados aos desenvolvedores e/ou aplicação cliente ordenados por algum critério de interesse estabelecido pelo desenvolvedor na fase de *design*. Por exemplo, operações executando em segundo plano não necessitam dispor de tempo de resposta elevado,

Figura 30 – Diagrama de sequência para a realização de uma substituição de serviço.



Fonte: Elaborada pelo autor.

mas, ao mesmo tempo, baixa variação de latência é uma característica desejável. Diante desse cenário, o desenvolvedor pode considerar latência como um atributo prioritário. De maneira

complementar, operações que estejam sendo executadas por usuários podem exigir maior tempo de resposta, pois o usuário está ativamente trocando informações com o servidor[2].

6.2.3 Módulo de implantação

O módulo de implantação tem como objetivo auxiliar no processo de implantação de serviços, podendo atuar em ambas as fases: *design* e *runtime*. Em outras palavras, um serviço pode ser implantado, desimplantado e reimplantado de forma transparente, sem a percepção de seus interessados. Para isso, esse módulo atua como um cliente intermediário, sendo responsável por gerenciar a troca de dados entre a aplicação e o serviço que está sendo requisitado. Ao funcionar de maneira integrada com o núcleo de adaptação, este módulo permite gerenciar serviços SOAP e RESTful.

Para serviços SOAP (Figura 27 – classe `WSSoapModel`), a substituição é realizada sem que seja necessário gerar novas classes *stubs* no processo de implantação. A geração de *stubs* pode ser considerado um processo oneroso para domínios de aplicação específicos, principalmente para aqueles em que o tempo de execução é considerado um fator crítico ou que possuem recursos computacionais escassos. Dessa forma, para lidar com esse tipo de serviço, o *framework* DynaMS disponibiliza um processo de "implantação dinâmica" (Figura 28), em que não é necessário gerar tais *stubs* durante o processo de implantação de serviços. Assim, os serviços podem ser substituídos em tempo de execução sem interrupção da aplicação.

Para operações que envolvam serviços RESTful (Figura 27 – classe `WSRestModel`), por utilizar o protocolo HTTP como método de comunicação, o problema envolvendo *stubs* é inexistente. Portanto, o objetivo do módulo de implantação (Figura 28) para esse tipo de serviço é, através de um cliente HTTP, realizar o intermédio da comunicação entre a aplicação e o serviço RESTful.

Por fim, vale ressaltar que o módulo necessita apenas do nome do serviço e da operação que será executada via SOAP ou REST, informação que pode ser obtida por meio do modelo de representação do serviço. Esse modelo é gerado a partir da documentação expressa em formato WSDL ou JSON, ou obtido através do repositório de serviços, que será apresentado em detalhes na Seção 6.3.1.

6.2.4 Módulo de busca

O módulo de busca do *framework* DynaMS visa apoiar tanto os engenheiros de software na fase de *design* quanto os processos automatizados na fase de *runtime* na realização de buscas por serviços em repositório. Como esse módulo envolve o uso de uma infraestrutura mais

[2] É importante destacar que, embora os termos "latência" e "tempo de resposta" pareçam sinônimos, existem diferentes definições para ambos, como mostrado anteriormente na Tabela 2 (p. 39).

complexa, a qual é detalhada na Seção 6.3.1, optou-se por apresentar nesta seção apenas os tipos de busca suportados, a saber:

Técnica. Este tipo de busca é voltado para os aspectos técnicos de um serviço: (i) nome da operação, (ii) parâmetros de entrada, e (iii) tipo de retorno.

Semântica. Este tipo de busca visa analisar informações de contexto semântico por meio de consultas SPARQL, as quais tem por objetivo recuperar os serviços/operações mais alinhados ao contexto pretendido.

Híbrida. Este tipo de busca combina as informações técnicas e características semânticas de serviços Web utilizando os dois tipos de busca anteriores.

6.2.5 Módulo de desenvolvimento

O módulo de desenvolvimento tem por objetivo prover ao desenvolvedor um conjunto de diretrizes e facilidades para apoiar o desenvolvimento de *Self-Apps*. É a partir desse módulo que são fornecidas as estruturas e métodos necessários para especificar a utilização de um serviço e sua respectiva lista de alternativas.

Antes de determinar quais serviços serão utilizados por uma aplicação, é essencial encontrá-los em um repositório ou, se necessário, desenvolvê-los. O desenvolvimento de um serviço para utilização no *framework* DynaMS não difere do desenvolvimento de um serviço comum baseado nos protocolos SOAP ou REST. Entretanto, é indicado que durante seu desenvolvimento sejam realizadas anotações determinando ao menos as características básicas da operação, como, por exemplo, nome do método, nome dos atributos necessários para sua execução e qual resposta será produzida. Para serviços SOAP, esse conjunto de características é disponibilizado automaticamente através dos documentos WSDL, tornando-se opcional ao desenvolvedor a inserção de descrições complementares sobre a tarefa que cada operação se propõe a realizar. A Listagem 3 mostra o esqueleto de construção do controlador de um serviço SOAP baseado na API JAX-WS (do inglês, *Java API for XML Web Services*). Na linha 1 pode-se observar que foi definido um serviço de nome “Clientes”. Adicionalmente, é destacado o método `getUserInfo`, responsável por retornar o conjunto de informações de um cliente a partir de seu nome de usuário (linhas 10 a 17). O conjunto de características necessárias para a utilização desse serviço serão gerados pelo documento WSDL, o qual será posteriormente interpretado pelo *parser* WSDL, resultando em uma instância do modelo de serviço Web.

Já a utilização de serviços RESTful envolve a necessidade de um passo adicional, representado pela utilização de um *framework* para especificar a documentação do serviço. Como mostrado na Seção 6.2.1, optou-se pela utilização do *framework* Swagger e, consequentemente, o

Listagem 3 – Esqueleto de uma operação de um serviço baseado em SOAP

```
1  @WebService(serviceName = "Clientes")
2  public class UsersController {
3      //...
4
5      /**
6       * Operação de um serviço Web SOAP para coletar as informações de um
            ↪ cliente
7       * @param user Nome do usuário
8       * @return Informações do usuário
9       */
10     @WebMethod(operationName = "getUserInfo")
11     public String getUserInfo(@WebParam(name = "info") String user) {
12         /*
13         ** Código fonte da operação
14         */
15
16         return dados;
17     }
18
19     //...
20 }
```

framework DynaMS requer sua utilização para a produção da documentação dos serviços desenvolvidos. Embora sua utilização seja recomendada por ser suportada nativamente pelo *framework* DynaMS, o desenvolvedor é livre para escolher uma outra alternativa para documentação, sendo necessário adotar um dos seguintes caminhos alternativos:

1. Transformação da documentação alternativa para o padrão utilizado pelo *Swagger*, que deve ser um arquivo JSON com a mesma estrutura e nomes de variáveis. Tal procedimento torna possível a utilização do *parser* durante a extração dos dados de serviço especificados na documentação produzida pelo *Swagger*; ou

2. Implementação de um novo *parser* que suporte o novo *framework* de documentação escolhido pelo desenvolvedor, traduzindo os dados do serviço especificados na documentação para o modelo de serviço Web.

A Listagem 4 mostra o esqueleto de construção do controlador de um serviço REST desenvolvido a partir do *framework* Spring Boot[3]. Inicialmente, esse exemplo de código especifica que a classe `Clientes` representa um controlador REST, cujo *endpoint* está localizado no endereço terminado em "`/clientes`" (linhas 1 e 2). A linha 4 representa uma anotação do *framework* Spring (`@AutoWired`), que é responsável por injetar as dependências necessárias para

[3] Disponível em: <https://spring.io/>. Acesso em 03/05/2020

a criação de uma nova instância da classe `UserService`. Nas linhas 9 e 10 são utilizadas duas anotações do *framework* Swagger, que especificam a operação que um método implementa e a(s) possível(is) resposta(s) produzida(s), respectivamente. Em seguida, linha 14, pode-se observar o método de requisição HTTP utilizado pelo método e como os dados serão fornecidos. Nesse caso, uma *string*, que representa o nome do usuário, informada após o *endpoint* `/clientes`). Finalmente, linha 15, o parâmetro "String user" do método `get` sendo mapeado pela anotação `@PathVariable` para receber o valor de "/info" como parâmetro. Ainda nessa linha, pode-se observar a anotação `@ApiParam` do *framework* Swagger, que é utilizada para documentar tal parâmetro.

Listagem 4 – Esqueleto de uma operação de um serviço baseado em REST

```
1  @RestController
2  @RequestMapping(value = "/clientes", produces = "application/json;
       ↪ charset-utf-8")
3  public class Clientes {
4      @Autowired
5      private UserService userService;
6
7      //...
8
9      @ApiOperation(value = "Operação de um serviço Web REST para coletar as
           ↪ informações de um cliente")
10     @ApiResponses(value = {
11             @ApiResponse(code = 200, message = "Informações do usuário")
12         }
13     )
14     @GetMapping(value = "/{info}")
15     public ResponseEntity<String> get(@ApiParam(value = "Nome do usuário")
           ↪ @PathVariable(name = "info") String user){
16         /*
17         ** Código fonte da operação
18         */
19
20         return dados;
21     }
22
23     //...
24 }
```

Todo o processo discutido descreve uma etapa prévia referente ao desenvolvimento dos serviços que serão consumidos por outras aplicações. Tal processo foi apresentado apenas com o intuito de mostrar que o desenvolvimento dos serviços Web utilizados pelo *framework* não difere muito das etapas normalmente adotadas durante o desenvolvimento de um novo serviço, sendo necessário apenas uma atenção em particular com a documentação do serviço. É possível argumentar a necessidade de se desenvolver a documentação para serviços RESTful como um passo adicional, em especial se for considerada a característica de serviços SOAP de gerar

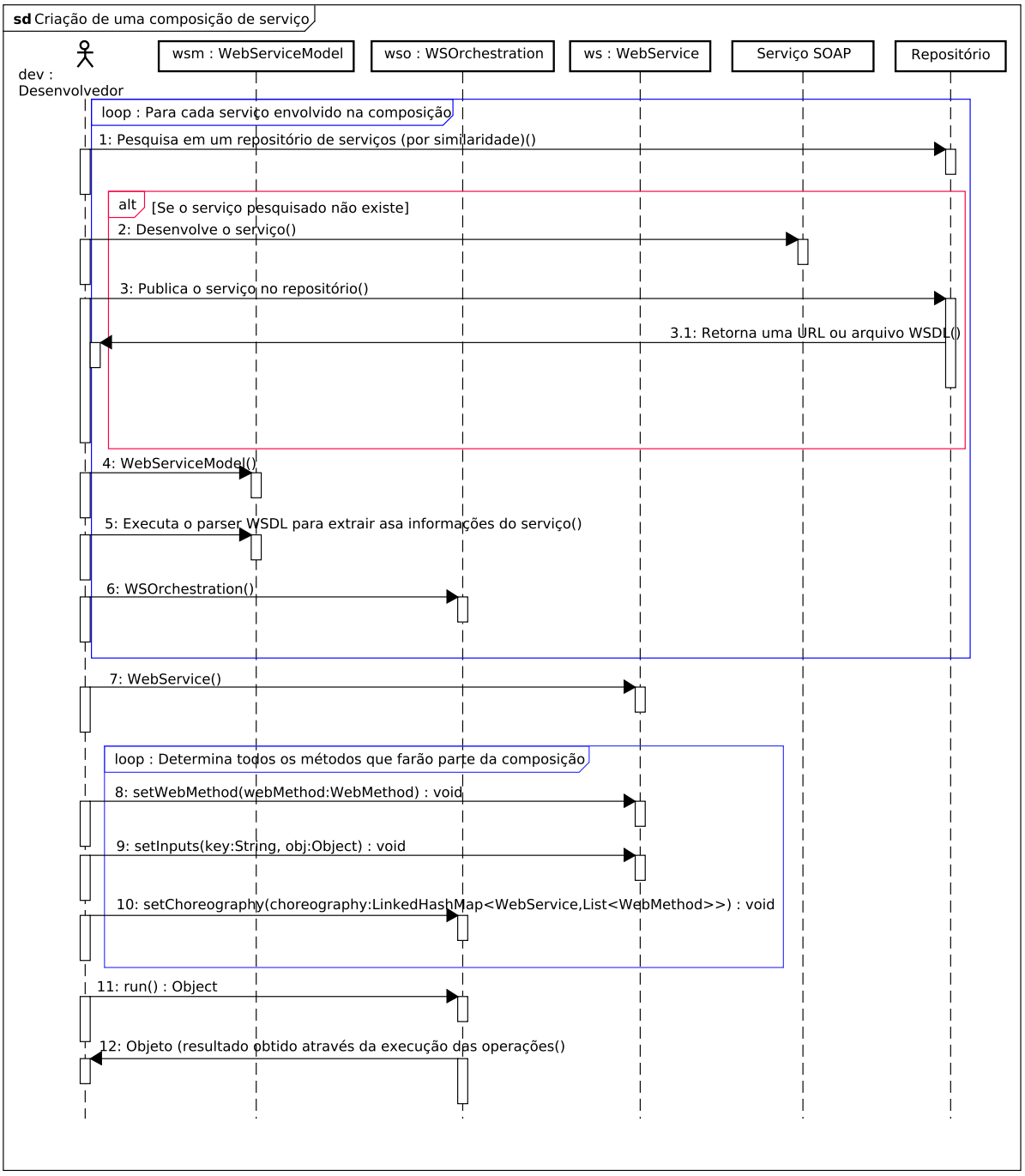
um documento WSDL automaticamente. Entretanto, é importante ressaltar que, sempre que possível, deve ser desenvolvida uma documentação clara e objetiva de um sistema de software, independentemente de qual tecnologia está sendo utilizada. Dessa forma, o funcionamento de tais sistemas poderá ser entendido com facilidade por outras pessoas. Sendo assim, embora serviços RESTful envolvam a necessidade de se desenvolver uma documentação para interagirem adequadamente com o *framework* DynaMS, em um cenário ideal tal documentação já seria desenvolvida independentemente da adoção do *framework*.

Para o desenvolvimento de aplicações que utilizem o *framework* DynaMS, fica à cargo do desenvolvedor, durante a fase de *design*, especificar qual(is) serviço(s) será(ão) utilizado(s) e os métodos que devem ser executados. Em caso de serviços compostos, além das informações para a primeira operação, é necessário fornecer também as informações complementares necessárias para as operações seguintes da coreografia. O diagrama de sequência apresentado na Figura 31 sintetiza o conjunto de tarefas envolvidas durante a composição de um novo serviço. Inicialmente, durante o desenvolvimento de uma nova composição, cabe ao desenvolvedor pesquisar em um repositório por serviços capazes de atender aos requisitos (passo 1) e escolher o serviço preferencial e, opcionalmente, um conjunto de alternativas. Essa lista contendo o conjunto de serviços alternativos também pode sofrer alterações durante a fase de execução do serviço, sendo modificada a partir da realização de novas consultas ao repositório. Caso não seja possível encontrar nenhum serviço, um novo deve ser desenvolvido e publicado no repositório (passos 2 e 3). Em seguida, os dados do serviço são obtidos a partir de seu respectivo *parser* (passos 4 e 5), uma nova orquestração de serviços é instanciada (passo 6) e os dados do serviço a ser utilizado são recuperados (passo 7). Em seguida, os métodos que irão compor o novo serviço são definidos (passo 8), o conjunto de parâmetros de entrada é informado (passo 9) e o serviço preferencial é inserido na coreografia junto da lista de alternativas (passo 10). Finalmente, após a conclusão da criação da coreografia, a tarefa representada pela composição pode ser executada. Embora o diagrama retrate um cenário envolvendo um serviço SOAP, o processo é análogo para serviços RESTful, diferenciando-se apenas na obtenção dos detalhes do serviço. Vale enfatizar que tais detalhes não são obtidos através de um documento WSDL, mas de um outro formato de descrição mais adequado para esse padrão, como um arquivo JSON.

6.2.6 Processo de adaptação

A Figura 32 ilustra o processo de adaptação do *framework* DynaMS, o qual engloba os módulos apresentados nas Seções 6.2.1 a 6.2.5, além da infraestrutura complementar do ambiente de execução. Basicamente, uma aplicação desenvolvida com base em tal *framework* pode ser organizada em duas camadas: (i) dispositivos clientes; e (ii) servidor de serviços Web. A seguir, detalhes de cada camada são apresentados.

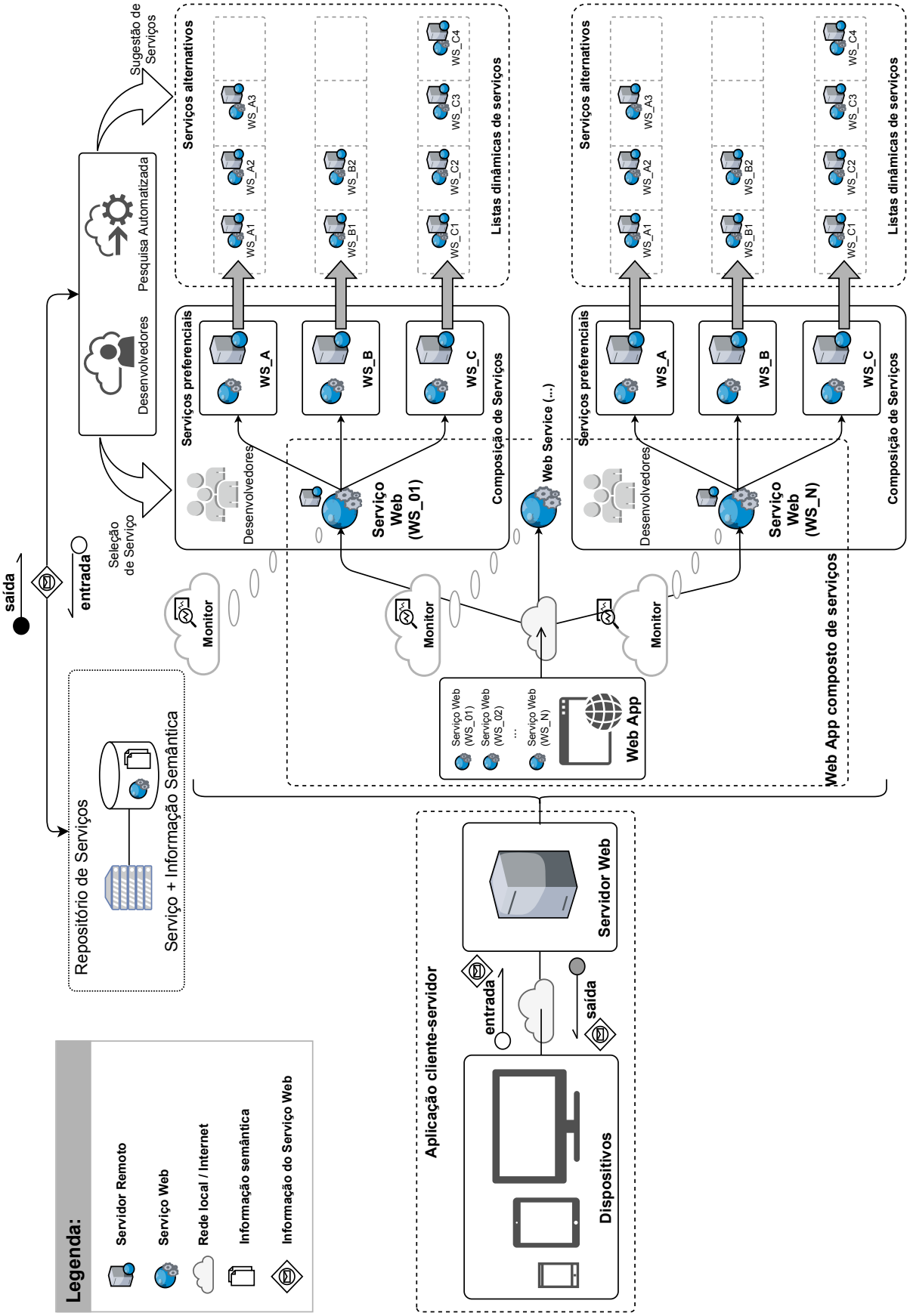
Figura 31 – Diagrama de sequência para a criação de uma composição de serviço.



Fonte: Elaborada pelo autor.

A camada de **dispositivos clientes** representa as aplicações que se comunicam com o servidor Web para prover um conjunto de funcionalidades a seus usuários. Em resumo, essas aplicações podem ser classificadas em *desktop*, Web, móveis, serviços e *mashup*. As aplicações *desktop*, Web e móveis podem realizar troca de informações com aplicações servidoras e apresentar os resultados para os usuários. As aplicações do tipo serviço podem ser caracterizadas como aplicações intermediárias, que consomem dados do servidor Web e disponibilizam os

Figura 32 – Processo de Adaptação.



Fonte: Elaborada pelo autor.

resultados para outros clientes. Por fim, as aplicações *mashup* tem por objetivo integrar diversos tipos de serviços distintos com intuito de fornecer um só serviço com todas as informações integradas. Tipicamente, essas aplicações são compostas por provedores de conteúdo (APIs), *mashup* Websites e aplicação cliente como navegadores Web. No contexto deste trabalho, esses dispositivos interagem diretamente com o servidor Web, que representa o local de implantação de serviços acessíveis através dos protocolos SOAP ou REST.

A camada de **servidor de serviços Web** representa a organização de uma aplicação. Serviços mais complexos são representados por meio de composição de serviço (`WS_01`), que possui um ou mais serviços preferencias (`WS_A` ... `WS_C`) e, opcionalmente, uma lista de serviços alternativos (`WS_A1`, ... , `WS_A3`) para cada serviços alternativo. Serviços preferenciais são selecionados pelo desenvolvedor como os mais aptos para realizar uma determinada tarefa. A lista de serviços alternativos, elaborada na fase *design*, permite armazenar serviços similares aos preferenciais que podem ser utilizados quando um serviço preferencial apresentar algum cenário de degradação de QoS ou falha de execução na fase de *runtime*. Para isso, o desenvolvedor utiliza o Repositório de Serviços Web para localizar os serviços preferenciais e alternativos da aplicação, sendo possível analisar informações de infraestrutura, como URL de conexão, e dados técnicos, como o conjunto de métodos e parâmetros. Por fim, cada serviço é monitorado em tempo de execução pelo componente `Monitor`, o qual é responsável por verificar constantemente o estado dos serviços utilizados pela aplicação e sugerir a necessidade da realização de uma adaptação quando eventuais falhas ou degradação no nível de qualidade do serviço foram detectadas. Tal sugestão deve vir acompanhada de um serviço sugerido, o qual será selecionado após um processo de seleção e do estabelecimento de um ranking, garantindo que a substituição selecione o melhor serviço possível, considerando fatores técnicos, contextuais e de qualidade.

6.3 Infraestrutura do ambiente

Durante a apresentação dos módulos do *framework* DynaMS (Seções 6.2.1 a 6.2.5), uma infraestrutura complementar, de natureza essencial para o funcionamento do mesmo, foi mencionada. Assim, como forma de apresentar tal infraestrutura, esta seção foi organizada da seguinte maneira: a Seção 6.3.1 apresenta os detalhes de projeto do repositório de serviços; e a Seção 6.3.2 reporta os detalhes de projeto do sistema de monitoramento de serviços.

6.3.1 Repositório de serviços

O desenvolvimento de um repositório visa facilitar o armazenamento e localização de serviços por engenheiros de software na fase de *design* e processos automatizados na fase *runtime*. Em síntese, os serviços são desenvolvidos/recuperados para compor uma aplicação na fase de *design*. No ambiente de execução os serviços podem ser recuperados para atender

uma necessidade de adaptação da aplicação, que pode ser em decorrência de uma falha ou degradação de QoS. Conforme reportado na Seção 6.2.1, o *framework* DynaMS oferece suporte ao desenvolvimento de aplicações baseadas em serviços SOAP e RESTful. Na Seção 6.2, diferenças entre esses serviços foram destacadas, sendo estas presentes no desenvolvimento desse repositório. A descoberta de serviços baseados em SOAP ocorre tradicionalmente com a utilização de um repositório denominado UDDI, o qual armazena os documentos WSDL dos serviços e permite realizar consultas e descobertas de serviços que atendam às necessidades de um desenvolvedor e/ou usuário. Entretanto, tal repositório apresenta uma descrição puramente técnica, ao passo que o *framework* detalhado neste trabalho incorpora também suporte à busca semântica ou híbrida (Seção 6.2.4). Adicionalmente, repositórios UDDI oferecem originalmente suporte apenas para serviços SOAP, sendo necessário buscar uma alternativa para serviços RESTful. Assim, baseado em tal restrição, optou-se pela adoção de uma ontologia para descrição de serviços e no desenvolvimento um novo repositório baseado em tal ontologia. A escolha por esse tipo de repositório teve como influência direta as evidências coletadas na QP5 do mapeamento da literatura apresentado no Capítulo 5.

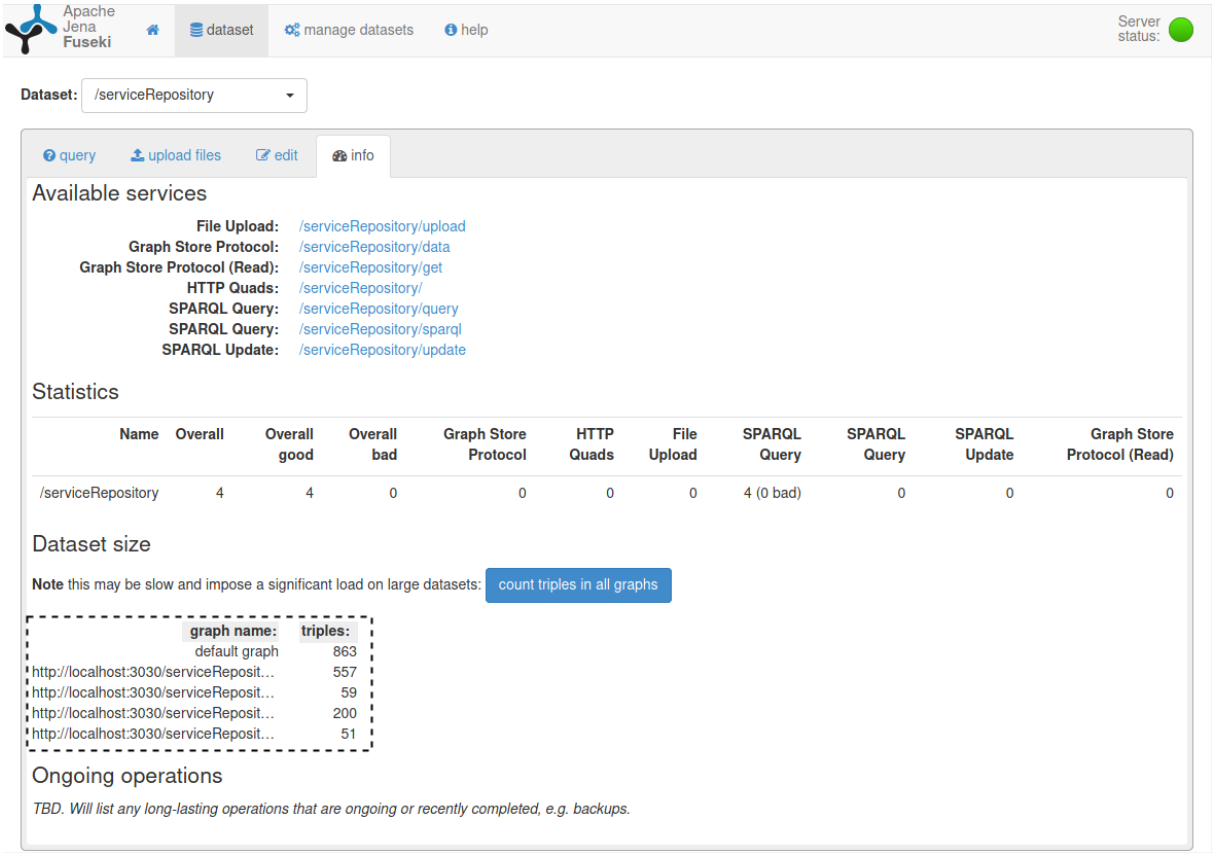
Conforme apresentado na Seção 4.2, existem diferentes ontologias para representar serviços Web. Para o desenvolvimento do repositório do *framework* DynaMS optou-se pela utilização da ontologia OWL-S por três motivos, a saber: (i) trata-se de uma das primeiras ontologias desenvolvidas com a finalidade de representar serviços Web, tendo sido utilizada consistentemente por outros pesquisadores em projetos dessa natureza ou para o desenvolvimento de novas ontologias; (ii) adota uma representação mais genérica para a descrição de serviços Web, o que, segundo nosso entendimento, facilita a manipulação de tecnologias de diferentes naturezas, como, por exemplo, SOAP e REST; e (iii) foi uma das ontologias mais reportadas pela QP5 do mapeamento conduzido. Como ilustrado na Figura 14 (Capítulo 4, p. 62), a OWL-S é resultado da composição de três outras ontologias, as quais são utilizadas para descrever um serviço de maneira geral (`Profile`), relatando os meios de acesso (`Grounding`) e o conjunto de métodos e parâmetros que implementa (`Process`). Assim, independentemente do tipo de serviço Web adotado, esse conjunto de informações deve ser disponibilizado de alguma maneira, tornando tal ontologia genérica o suficiente, de maneira que seja extensível para diferentes tecnologias de implementação e comunicação de serviços e, ao mesmo tempo, permita especificar os detalhes necessários para que um cliente seja capaz de descobri-los e realizar requisições.

Para a manipulação dos dados foi adotado o *framework* Apache Jena [4], o qual provê um conjunto de bibliotecas Java para auxiliar no desenvolvimento e manutenção de dados semânticos. Adicionalmente, inclui suporte para a utilização de motores de inferência (por exemplo, *Pellet*), facilitando a derivação de novas informações a partir da análise do conjunto de dados de serviços existentes. Tal *framework* suporta também um servidor SPARQL (Apache

[4] Disponível em: <https://jena.apache.org>. Acesso em 03/05/2020

Jena Fuseki [5]), providenciando uma camada robusta para armazenar e manipular o conjunto de dados que compõe o repositório. A Figura 33 exibe o painel de informações providenciado pelo servidor SPARQL. Nesse painel é apresentado um conjunto de *endpoints* que podem ser utilizados para estabelecer a comunicação com o repositório (*Available Services*) e estatísticas gerais relacionadas ao *dataset* (*Statistics* e *Dataset size*). A partir dessa figura é possível observar que quatro grafos estão sendo armazenados no *dataset* (linha tracejada). Esse conjunto de grafos representa as três partes distintas da ontologia OWL-S (`Profile`, `Process` e `Grounding`) e a ontologia `Service`, responsável por relacioná-las, compondo o vocabulário final da ontologia utilizada neste trabalho. Adicionalmente, vale mencionar a existência de um grafo padrão (*default graph*), o qual representa a união de todo o conjunto de dados do *dataset*. Tal funcionalidade vem desativada por padrão durante a criação de um *dataset* mas é interessante habilitá-la, pois deixará à cargo do Apache Jena Fuseki a tarefa de unir todos os grafos em um único, possibilitando a utilização de consultas SPARQL mais simples e abrangentes durante a realização de uma busca no repositório.

Figura 33 – Painel de informações do Apache Jena Fuseki



Fonte: Elaborado pelo autor.

Para inserir um novo serviço no repositório é necessário traduzi-lo para os padrões da ontologia OWL-S, sendo importante determinar quais serão os relacionamentos entre o modelo

[5] Disponível em: <https://jena.apache.org/documentation/fuseki2/>. Acesso em 03/05/2020

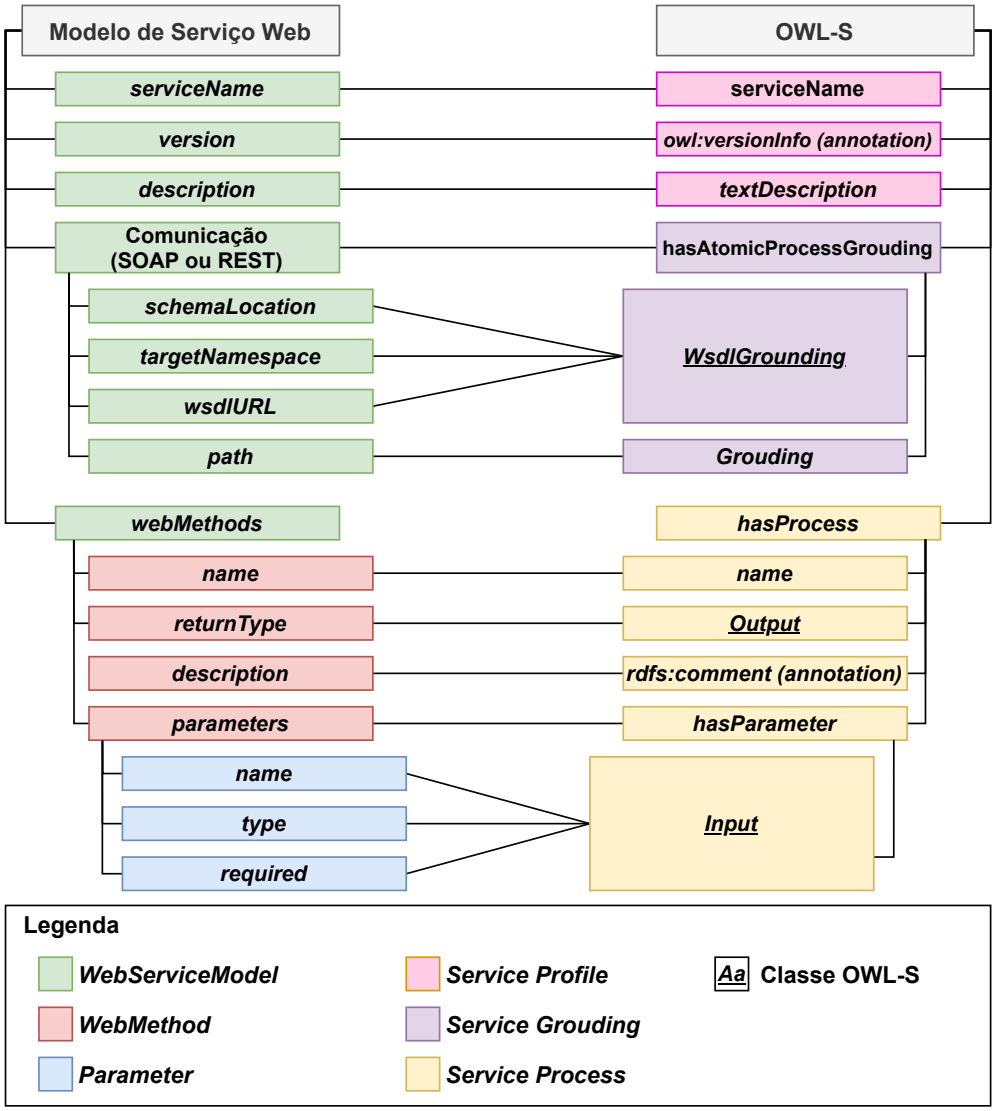
de serviço utilizado pelo *framework* e o conjunto de classes, propriedades e anotações fornecidos pela ontologia OWL-S. Portanto, após a análise da ontologia utilizada, foi estabelecido um mapeamento entre as duas estruturas de representação utilizadas, ilustrado na Figura 34. A partir dos relacionamentos estabelecidos por esse mapeamento, foi implementado um processo de tradução, cujo objetivo é receber um serviço representado a partir do metamodelo utilizado pelo *framework* (Figura 27) e convertê-lo para o modelo de representação utilizado pela ontologia. É importante ressaltar que o conjunto de classes e propriedades utilizadas pelo mapeamento ilustrado na figura não representa a totalidade de recursos disponibilizados pela ontologia OWL-S, a qual possui a capacidade de descrever informações adicionais de um serviço, tais quais informações de contato com a equipe ou empresa responsável e as condições prévias as quais o serviço deve obedecer antes de sua execução. Além disso, vale destacar que por se tratar da primeira versão do mapeamento, optou-se por adotar uma versão simplificada do modelo da ontologia OWL-S, facilitando o desenvolvimento e estabelecimento de uma versão inicial do repositório de serviços. Adicionalmente, cabe ressaltar ainda que, como toda ontologia, é possível modificar a OWL-S de acordo com as necessidades da aplicação que a utilizará. Isso resulta em um alto poder de adaptabilidade e flexibilidade do repositório, o qual pode ser moldado para as necessidades individuais de quem irá utilizá-lo, sendo necessário apenas realizar ajustes adicionais no algoritmo de tradução entre a representação de serviço e a ontologia.

Por fim, a Figura 35 ilustra o processo de inserção de serviços no repositório, o qual envolve três elementos distintos: (i) o serviço interessado em disponibilizar seus dados no repositório semântico; (ii) um serviço de interação com o repositório, que deve ser responsável por intermediar a comunicação entre o repositório de serviços, auxiliando na inserção, busca e remoção de informações e um cliente, representado por um usuário ou outro serviço; e (iii) o repositório, responsável por armazenar o vocabulário semântico e os dados referentes aos serviços que disponibiliza. Para inserir um serviço no repositório é necessário enviar uma solicitação ao *endpoint* responsável pela interação com o serviço de repositório (Figura 35a). O *endpoint*, por sua vez, encaminhará os dados descritivos do serviço requisitante, padronizado em WSDL ou JSON, para uma API de tradução, cuja função é converter esses dados para o modelo de representação utilizado pela ontologia OWL-S (Figura 35b). Após a etapa de tradução, o modelo resultante será devolvido para o *endpoint* (Figura 35c) que realizará a inserção de tal modelo no repositório de serviços através de um dos *endpoints* disponibilizados por padrão pelo Apache Jena Fuseki (Figura 35d), concluindo a operação de inserção de serviços no repositório.

6.3.2 Sistema de monitoramento

Como reportado na Seção 2.3, a adoção de diferentes atributos de QoS para estabelecer um conjunto de métricas de avaliação permite quantificar o nível de qualidade de um determinado serviço Web. A Tabela 2 (p. 39) apresenta um panorama dos atributos de QoS identificados pelo

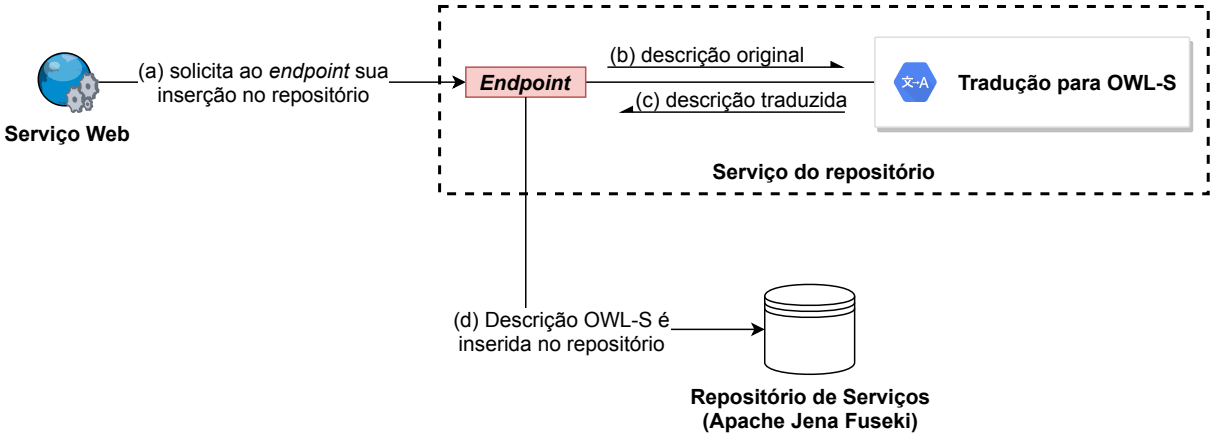
Figura 34 – Mapeamento entre o modelo de serviço do *framework* e a ontologia OWL-S.



Fonte: Elaborado pelo autor.

mapeamento apresentado neste trabalho e suas respectivas definições (veja o Capítulo 5). Para o desenvolvimento da versão inicial do sistema de monitoramento utilizado pelo *framework* DynaMS optou-se por selecionar alguns desses atributos, focando principalmente na utilização daqueles mais reportados pela Questão de Pesquisa 3 (QP3, Seção 5.1, p.75) do mapeamento conduzido. Portanto, os seguintes atributos de QoS foram adotados: (i) disponibilidade; (ii) latência; (iii) tempo de resposta; (iv) confiabilidade; e (v) carga, atributo referente ao consumo de CPU e/ou memória. Entretanto, é importante destacar que as informações relacionadas a cada um dos atributos podem ser obtidas a partir de origens distintas. Por exemplo, os dados referentes a disponibilidade e latência do serviço podem ser obtidos a partir da aplicação responsável por realizar a requisição ao serviço. A latência de um serviço, em particular, é um atributo que apresenta variações relacionadas com características específicas da rede em que o serviço e seu

Figura 35 – Processo de inserção de um novo serviço no repositório semântico.



Fonte: Elaborado pelo autor.

consumidor se encontram como, por exemplo, qual é a rota utilizada na comunicação entre os dois pontos. Consequentemente, tal atributo é melhor avaliado se medido pela aplicação cliente que está realizando a requisição. Entretanto, um atributo de QoS, como carga da aplicação, refere-se a uma informação normalmente acessível apenas através do sistema operacional onde o serviço está executando, sendo necessário disponibilizá-la através de algum meio de acesso externo como um *endpoint*. Para atender ao contexto exposto, o sistema de monitoramento foi organizado em duas partes: (i) *QoS data provider*; e (ii) *QoS core*.

O ***QoS data provider*** é disponibilizado através de uma biblioteca que deve ser vinculada ao serviço que será implantado. Seu objetivo é fornecer, através de um *endpoint*, um conjunto específico de informações do serviço Web e/ou do servidor que o executa. Como o *framework* DynaMS foi implementado em Java, as informações coletadas e disponibilizadas são diretamente relacionadas com a máquina virtual Java (do inglês, *Java Virtual Machine* – JVM). A Listagem 5 mostra as informações disponibilizadas no formato JSON através de um *endpoint*, na qual é possível observar três categorias, a saber:

1. *Informações específicas da JVM (linhas 2 à 9)*: nesta categoria (dados disponibilizados pelas *tags* `jvm` e `availableProcessors`) é possível visualizar a memória em *bytes*, determinando a quantia total alocada, a quantidade disponível e o valor máximo que a JVM tem permissão para utilizar, além de informações sobre as limitações de CPU da JVM como, por exemplo, o número de núcleos disponíveis para utilização;

2. *Informações do sistema operacional (linhas 10 à 12)*: nesta categoria (dados disponibilizados pela *tag* `system`) nota-se a carga atual de utilização do processador, a qual é representada por um número decimal dentro do intervalo [0, 1]. Em outras palavras, esse intervalo equivale ao percentual de utilização da CPU de 0 a 100%, sendo que quanto mais

Listagem 5 – Exemplo do conjunto de informações recuperado pelo *QoS data provider*

```
1  {
2      "jvm": {
3          "memory": {
4          "maximum": 4177526784,
5          "used": 262144000,
6          "free": 219961856
7          },
8          "availableProcessors": 12
9      },
10     "system": {
11         "cpu.currentLoad": 0.038639598244049916
12     },
13     "application": {
14         "status": "UP",
15         "uptime": "2857 ms"
16     }
17 }
```

próximo de "zero", menor é o consumo de CPU, e quanto mais próximo de "um", maior é seu uso. Ainda sobre esse percentual, é possível argumentar que a baixa utilização de processador não é suficiente para justificar uma disponibilidade de alto poder de processamento. Por exemplo, processadores menos potentes com baixo grau de utilização podem ainda apresentar baixo percentual de processamento quando comparados a processadores mais potentes com maior percentual de processamento. Entretanto, vale ressaltar que uma das premissas intrínsecas deste trabalho é a adoção de serviços Web para a comunicação e realização do processamento mais intensivo. Tais serviços são geralmente implantados em servidores, os quais possuem poder de processamento médio a alto ou dispõem de escalabilidade dinâmica, possibilitando ajustar a capacidade de processamento às necessidades da aplicação em tempo de execução; e

3. *Informações específicas da aplicação (linhas 13 à 16)*: nesta categoria (dados disponibilizados pela *tag* `application`) é possível observar o estado atual do serviço (`status`) e seu tempo de execução atual em milissegundos (`uptime`). Esse tipo de informação possibilita determinar valores para atributos relacionados a disponibilidade com base em dois dados distintos: (i) determinação do estado atual do serviço; e (ii) tempo decorrido desde que o serviço Web foi iniciado;

O ***QoS core*** tem por objetivo prover um mecanismo de avaliação individual de serviços Web e, dado um conjunto de serviços avaliados, realizar uma recomendação baseada em um ou mais atributos de qualidade. Optou-se por utilizar uma avaliação individual para cada serviço,

como mostra a Equação 6.1. Essa pontuação pode ser obtida através da soma dos produtos entre o valor de determinado atributo "β" e seu respectivo peso "α", calculada para cada um dos atributos dentro do intervalo $[1, m]$, onde m representa o total de atributos. O peso α deve ser definido pelo desenvolvedor para atender as necessidades de cada aplicação, especificando seus respectivos níveis de criticidade.

$$pontuacao_i = \sum_{i=1}^{m} \beta_i \times \alpha_i \tag{6.1}$$

O valor "β" de um atributo pode ser avaliado de duas maneiras distintas, como mostra a Tabela 6. Uma avaliação binária apenas determina se o serviço é capaz de atender ao atributo em análise por meio de um valor *booleano* verdadeiro ou falso. Por exemplo, ao avaliar a situação de confiabilidade de um serviço, para o contexto atual do *framework* DynaMS, espera-se obter como resposta "confiável" ou "não confiável". Já um método de avaliação contínuo, determina a capacidade daquele serviço em atender a um atributo de qualidade com base em um intervalo contínuo quantificado por uma unidade de medida. Por exemplo, ao avaliar a latência de um serviço, espera-se obter um número em milissegundos (`ms`) dentro do intervalo $[0, \infty]$. Por fim, há também atributos que podem ser avaliados a partir de ambos métodos, como disponibilidade. É possível determinar tanto o estado atual de disponibilidade, compondo uma avaliação binária que tem "disponível" e "indisponível" como possíveis valores ou o tempo decorrido desde a inicialização da aplicação, especificada através de uma avaliação contínua, em que os valores expressam o tempo de execução em uma determinada unidade de medida de tempo.

Tabela 6 – Lista de atributos de QoS e seus métodos de avaliação

Atributo	**Avaliação**
Disponibilidade	**BINÁRIA** (disponível ou indisponível) **CONTÍNUA** (tempo de execução em milissegundos)
Latência	**CONTÍNUA** (número em milissegundos)
Tempo de resposta	**CONTÍNUA** (número em milissegundos)
Confiabilidade	**BINÁRIA** (confiável ou não confiável)
Carga (CPU/memória)	**CONTÍNUA** (porcentagem de uso de CPU ou memória)

Fonte: Elaborado pelo autor.

É importante ressaltar que a simples definição de uma pontuação individual para cada atributo não deve ser considerada como suficiente. Podem existir casos em que determinados atributos sejam considerados melhores a partir da análise de um limite superior e outros por um limite inferior. Disponibilidade, latência e custo são atributos de QoS que ilustram a ressalva exposta, pois pode-se idealizar um cenário que um serviço possua valores elevados para disponibilidade e, ao mesmo tempo, valores o mais próximo possível de zero para latência e custo.

Nesse caso, torna-se necessário definir uma função de padronização (veja a Equação 6.2), a qual deve ser executada antes de se determinar uma pontuação final, garantindo que os resultados finais sejam coerentes. Nessa equação, β_i representa o valor que será calculado para cada um dos atributos, val_i representa o valor original do atributo avaliado e N representa o total de serviços necessários para a normalização dos dados.

$$\beta_i = \begin{cases} \frac{val_i}{N}, \text{se limite inferior} \\ 1 - \frac{val_i}{N}, \text{se limite superior} \end{cases} \tag{6.2}$$

Por fim, após a determinação da pontuação dos serviços, o *QoS core* também tem como objetivo auxiliar na tarefa de recomendação, indicando o serviço que pode ser utilizado como substituto. Para isso, os serviços pontuados são inseridos em uma lista ordenada através dos atributos de QoS avaliados. Em situações que envolvam apenas um serviço simples, essa tarefa pode ser facilmente realizada através de um algoritmo de ordenação como o *quick sort*. Esse algoritmo possui complexidade de tempo média $O(n \log n)$, sendo particularmente útil em cenários em que não se conhece o estado atual de ordenação da lista. Por conta desse conjunto de características, tal algoritmo prova-se útil para as necessidades básicas de ordenação de uma lista de pontuação de serviços simples. Em contrapartida, em situações que requerem a substituição de um serviço composto e a recomendação de mais de um serviço, a utilização de um simples algoritmo de ordenação pode não ser a melhor escolha. Como a seleção de serviços pode envolver diferentes atributos de qualidade e seus respectivos pesos, os quais podem variar de acordo com o serviço que está sendo analisado, é necessário garantir que o conjunto de potenciais substitutos sendo avaliados esteja otimizado para as necessidades individuais de cada uma das tarefas que serão realizadas. Essa situação remete a um problema clássico de otimização e requer a utilização de um algoritmo voltado para essa finalidade. Nesse sentido, experiências com dois algoritmos foram conduzidas, a saber: (i) algoritmo de Dijkstra; e (ii) algoritmo de otimização por enxame de partículas. Inicialmente, optou-se pela adoção do algoritmo de Dijkstra (Algoritmo 2) para atuar na recomendação de serviços do *framework* DynaMS. Essa escolha foi motivada por dois fatores: (i) o banco de dados de serviços possui uma representação em grafos, tipo de dado utilizado nesse algoritmo, e; (ii) não foi encontrado nenhum outro trabalho no mapeamento sistemático (ver Capítulo 5) que indicasse o uso desse algoritmo de maneira explícita.

Por fim, vale enfatizar que a combinação entre o sistema de monitoramento apresentado nesta seção e o módulo de planejamento e tomada de decisões (Seção 6.2.1) permite identificar anomalias, além de propor alternativas para sua correção em tempo de execução. Dessa forma, torna a substituição de serviços ou adaptação de uma composição uma tarefa transparente aos usuários dos sistemas e processos automatizados, evitando que aplicação se torne indisponível ou incapaz de satisfazer o nível de QoS exigido.

Algoritmo 2 Algoritmo de Dijkstra

```
1: função DIJKSTRA(G, S, D) ▷ Recebe um grafo G, uma aresta inicial S e uma aresta destino D
2:     dist[S] ← 0 ▷ Distância da origem a ela mesma é zero
3:     para todo V ∈ G faça
4:         se v ∉ S então
5:             dist[v] ← ∞ ▷ Seta demais distâncias como sendo infinito
6:             prev[v] ← −1 ▷ Marca nó pai
7:         fim se
8:         Q.add(v) ▷ Adiciona vértice v numa fila Q
9:     fim para
10:    enquanto Q ≠ ∅ faça
11:        u ← v ∈ Q com min(dist[v]) ▷ Seleciona vértice da fila com menor distância
12:        Q.rem(u) ▷ Remove vértice selecionado da fila
13:        se u = D então
14:            break ▷ Se a aresta selecionada é a aresta pesquisada, caminho foi encontrado
15:        fim se
16:        para todo v ∈ u faça ▷ Para os vizinhos de u em v
17:            alt ← dist[v] + dist_entre(v, u) ▷ Calcula nova distância
18:            se alt < dist[u] então ▷ Caminho mais curto encontrado
19:                prev[v] ← u ▷ Adiciona ao caminho
20:                dist[u] ← alt ▷ Atualiza distâncias
21:            fim se
22:        fim para
23:    fim enquanto
24: retorna dist[], prev[] ▷ Listas com a menor distância e caminho de S a D
25: fim função
```

6.4 Considerações finais

Neste capítulo foram apresentados os detalhes referentes ao *framework* DynaMS e sua infraestrutura de apoio. Inicialmente, foi apresentado um modelo que ilustra os possíveis estados pelos quais um serviço pode passar durante uma adaptação seguido por um levantamento de requisitos realizado para o desenvolvimento do *framework* e pelos objetivos pretendidos com o processo de adaptação. Em seguida, os principais elementos que compõem o *framework* foram apresentadas em detalhes, assim como a infraestrutura complementar, composta por um repositório semântico de serviços, a ontologia OWL-S, o processo de tradução de serviços e o sistema de monitoramento. O conteúdo deste capítulo fornece ao leitor uma visão mais completa e detalhada das características do *framework* DynaMS, seus objetivos e ambiente de atuação proposto. A seguir, Capítulo 7, é conduzida uma avaliação do *framework*.

7 Avaliação do *framework* DynaMS

O objetivo deste capítulo é apresentar a avaliação do *framework* DynaMS, a qual é constituída a partir da condução de duas abordagens distintas. Inicialmente, Seção 7.1, um estudo de caso que aborda o desenvolvimento de um sistema para um restaurante inteligente é reportado. O objetivo desse estudo de caso é validar o *framework* de um ponto de vista prático, observando características voltadas para o desenvolvimento de aplicações, além de sua aplicabilidade e funcionamento em uma aplicação que simule um cenário real de utilização. Ainda sobre essa abordagem de avaliação, três motivos foram determinantes para sua escolha, a saber: (i) abordagem frequentemente encontrada na revisão da literatura reportada no Capítulo 5, mostrando ser um método de validação utilizado de maneira recorrente; (ii) possibilidade de validar o *framework* DynaMS e, ao mesmo tempo, apresentar um exemplo de utilização desde a concepção de um novo serviço até sua operação no ambiente de execução; e (iii) carência de tempo para a condução de uma avaliação mais formal como um experimento, pois este requer a definição de procedimentos e maior número de participantes para compor um bom espaço de amostragem.

7.1 Estudo de caso – *App2Rest*

Como parte do processo de avaliação do *framework* DynaMS, um estudo de caso foi elaborado visando explorar suas características em relação ao desenvolvimento de uma aplicação. Segundo Zelkowitz e Wallace (1998 apud WOHLIN *et al.*, 2012), a condução de um estudo de caso possui caráter observacional e tem como objetivo delinear detalhes sobre um atributo. Nesse sentido, a condução de um estudo de caso voltado para o *framework* DynaMS tem como principal objetivo obter parâmetros relacionados à sua aplicabilidade em uma situação de uso próxima a um cenário do mundo real. Adicionalmente, esse tipo de avaliação também possibilita determinar as eventuais limitações do *framework*, evidenciando pontos passíveis de melhoramento em futuros trabalhos. Como reportado na Questão de Pesquisa 2 (QP2, Seção 5.1, p.72), o domínio de sistemas da informação destacou-se entre o conjunto de estudos avaliados. Nessa direção, optou-se por adotar o desenvolvimento de um sistema para um restaurante inteligente, o qual será referenciado deste ponto em diante como *App2Rest*. Basicamente, tal sistema aproveita as vantagens providas por dispositivos móveis, permitindo um atendimento simplificado aos usuários nas diferentes etapas de consumo como, por exemplo, reservas, seleção de mesas, consultas ao cardápio, pedidos e pagamentos. Para isso, cada mesa do restaurante deve ter embutido os seguintes equipamentos: um *tablet* para interação dos usuários com recursos do sistema, como, por exemplo, uma versão eletrônica do cardápio e uma máquina leitora

de cartões para pagamentos com cartões de crédito ou débito. Em situações que não seja viável disponibilizar mesas com tais dispositivos, outro possível cenário é a disponibilização do aplicativo do restaurante para instalação nos dispositivos pessoais de seus usuários. Esse aplicativo deve permitir que o usuário execute o mesmo conjunto de operações disponibilizadas pelo *tablet* embutido na mesa, a partir do qual é possível acessar o sistema do restaurante dedicado aos usuários. Nesse sentido, o restaurante deve disponibilizar aos usuários pontos de conexão com a Internet. Além disso, cada mesa deve possuir uma lâmpada LED (do inglês, *Light-Emitting Diode*) com a finalidade de indicar os possíveis estados da mesa, a saber: (i) disponível, indicado pela cor verde; (ii) indisponível, indicado pela cor vermelha; ou, (iii) aguardando limpeza, indicado pela cor laranja.

Do ponto de vista organizacional, o sistema *App2Rest* é composto por duas aplicações operando em diferentes plataformas: (i) um aplicativo desenvolvido para dispositivos móveis com sistema operacional Android; e (ii) um conjunto de serviços Web desenvolvido em Java hospedados em um ambiente de servidor. Para viabilizar a troca de informações entre as aplicações, o aplicativo móvel em Android foi integrado aos serviços Web em Java, gerando uma aplicação móvel orientada a serviços organizada em duas camadas, a saber:

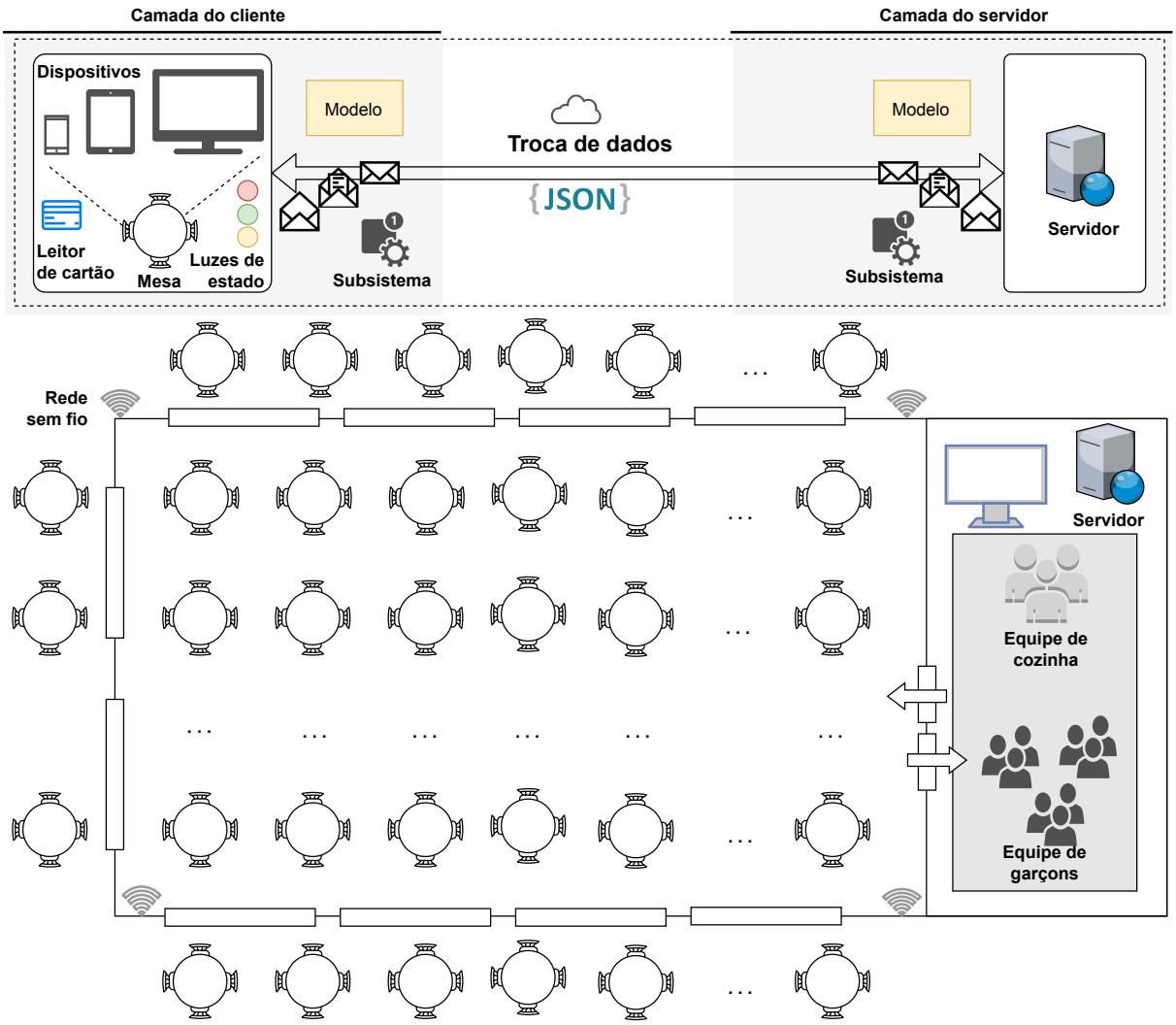
- **Cliente**, responsável pela interação do usuário com o sistema do restaurante, especificando a interface gráfica com a qual os clientes interagem (*front-end* da aplicação); e

- **Servidor**, responsável pela realização das operações solicitadas pelos clientes do restaurante (*back-end* da aplicação). As operações fornecidas por essa camada representam um conjunto de serviços Web disponíveis para a realização de funcionalidades do sistema como autenticação, visualização de cardápio e abertura de pedidos.

A Figura 36 mostra uma visão geral da arquitetura proposta para a aplicação *App2Rest*. O topo da figura ilustra a troca de informação entre as duas camadas cliente e servidor. O cliente faz, via mesa ou aplicativo, uma solicitação a uma funcionalidade do *App2Rest*, a qual é convertida em um modelo de dados em JSON para ser enviado ao lado servidor. Na camada servidor, as informações recebidas são transformadas novamente no modelo utilizado pela aplicação, o serviço é executado e o resultado da operação é novamente encapsulado em JSON e devolvido ao cliente que requisitou a operação. A parte inferior dessa figura ilustra uma organização física sugerida para o ambiente do restaurante. A organização das mesas e da cozinha em relação ao ambiente é irrelevante e pode variar de acordo com as necessidades do proprietário e/ou responsável pelo estabelecimento. Entretanto, uma característica crucial para o bom funcionamento dessa aplicação é a disponibilidade de sinal da rede do ambiente. Como o intuito dessa seção é descrever o estudo de caso de uma maneira mais geral, maiores informações

foram disponibilizadas no Apêndice D, o qual mostra um documento que descreve o conjunto de requisitos do sistema *App2Rest* e seu respectivo diagrama de classes.

Figura 36 – Visão geral da arquitetura do sistema *App2Rest*



Fonte: Elaborada pelo autor

A Figura 37 mostra as principais telas da aplicação Android desenvolvida para compor a camada de clientes do sistema avaliado. A Figura 37a ilustra a tela de abertura da aplicação, a partir da qual novos usuários podem criar uma nova conta ou realizar sua autenticação através de uma das opções disponibilizadas, a saber: (i) usuário privado criado especificamente para essa aplicação; ou (ii) uma conta Google. Após o processo de autenticação, a aplicação exibe uma tela principal (Figura 37b), na qual é disponibilizado um botão para iniciar um novo pedido. Também é apresentada uma sugestão do prato do dia e uma opção para consultar as ofertas disponíveis no restaurante. Ao deslizar para a direita obtém-se acesso ao menu principal da aplicação, o qual possibilita ao usuário consultar seus pedidos, o cardápio e suas informações pessoais. Ao selecionar a opção `Novo pedido`, o usuário é transferido para uma tela onde pode selecionar uma ou mais opções de pratos disponíveis no restaurante (Figura 37c). Por fim, ao

selecionar a imagem ou nome do prato, algumas informações como nome, descrição e preço são exibidas (Figura 37d), útil especialmente em situações em que não se conhece um determinado prato.

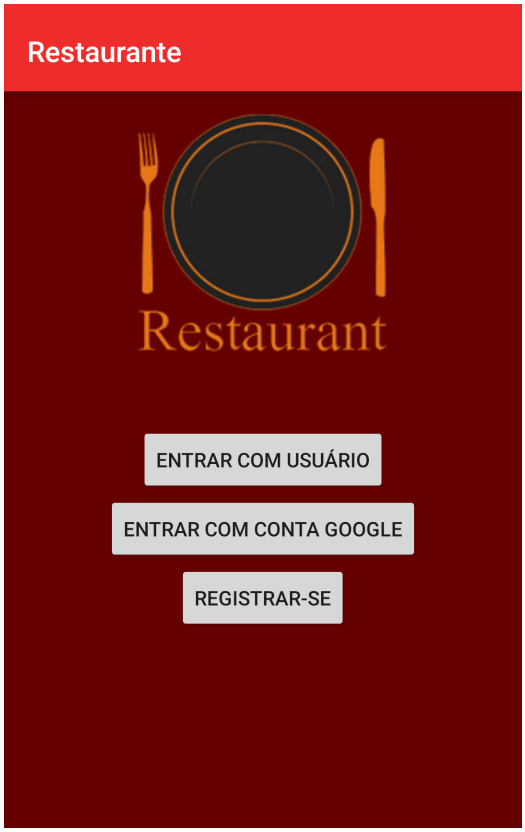
O estudo de caso que está sendo apresentado nesta seção descreve uma aplicação orientada a serviços e retrata um cenário observado com certa frequência durante o desenvolvimento de aplicações para dispositivos móveis, tais quais *smartphones*, *tablets* e dispositivos IoT. Tal modelo é especialmente benéfico em aplicações em que poder de processamento e/ou consumo de energia sejam limitados. Embora a aplicação proposta represente um domínio não crítico, a ocorrência de falhas na aplicação pode resultar em prejuízos na operação do restaurante. Portanto, dois cenários que justificam a utilização de um *framework* de adaptação são apresentados, a saber:

1. **Recuperação do sistema.** O objetivo deste cenário é lidar com a ocorrência de anomalias na execução da aplicação. Tais anomalias podem ser tratadas por meio da substituição dos serviços problemáticos por "saudáveis" em tempo de execução de maneira transparente aos interessados; e

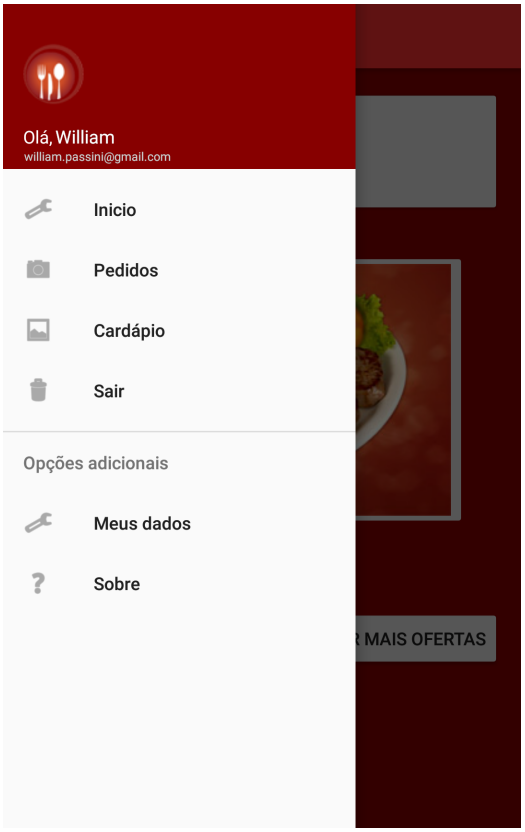
2. **Evolução do sistema.** O objetivo deste cenário é ilustrar uma característica de evolução relacionada ao sistema.

A Figura 38a ilustra um cenário de recuperação do sistema composto por 4 serviços. O primeiro serviço (`Serviço 1`) é responsável por autenticar o cliente no *App2Rest*. Para isso, o cliente deve fornecer suas credenciais de autenticação para que possam ser validadas pelo sistema. Se a autenticação não for bem-sucedida, o sistema solicita tais informações novamente. Uma vez autenticado, o aplicativo exibe a tela principal e o prato do dia é sugerido ao cliente. Essa operação requer que duas operações sejam executadas em paralelo, a saber: `Serviço 2`, que recupera os dados do cliente; e `Serviço 3`, que recupera os dados do menu do restaurante com sugestões do prato do dia. Em seguida, `Serviço 4`, uma comanda de pedidos é inicializada para que o cliente possa solicitar os produtos de consumo. Durante a fase *runtime*, esses serviços (1 a 4) são monitorados pelo componente Monitor, conforme reportado na Seção 6.2.6 – Figura 32. Para ilustrar o primeiro cenário de adaptação, uma anomalia será injetada propositadamente no serviço preferencial `Serviço 2`. Vale destacar que essa anomalia pode ser uma falha ou uma degradação de QoS, que, independentemente do tipo, irá necessitar do mesmo procedimento de reparo. Nesse sentido, pode-se partir da identificação da anomalia pelo componente Monitor a qual resultará em duas situações: (i) a lista de serviços alternativos possui um serviço que pode substituí-lo; (ii) a lista de serviços alternativos pode estar vazia. Esta última ilustra dois cenários, sendo o primeiro voltado para o desenvolvedor, que não selecionou os serviços alternativos na fase de *design* da aplicação, e o segundo, para o processo automatizado, que consumiu os serviços

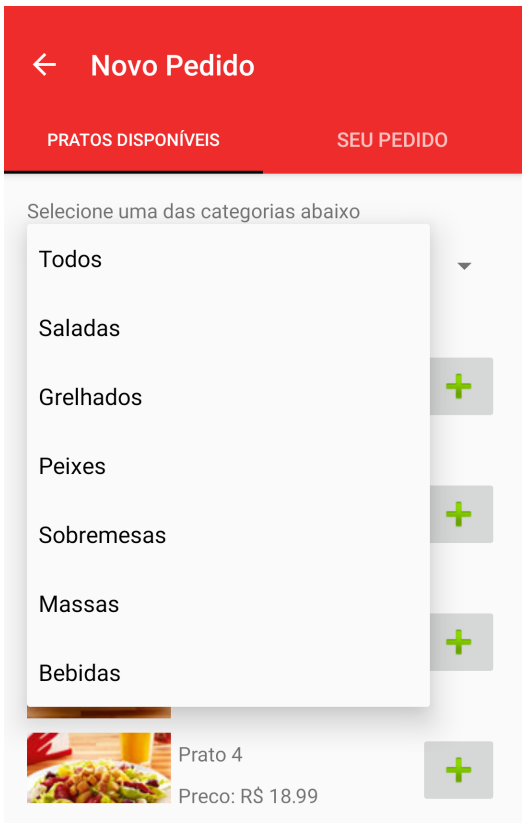
Figura 37 – Conjunto de telas da aplicação *App2Rest*



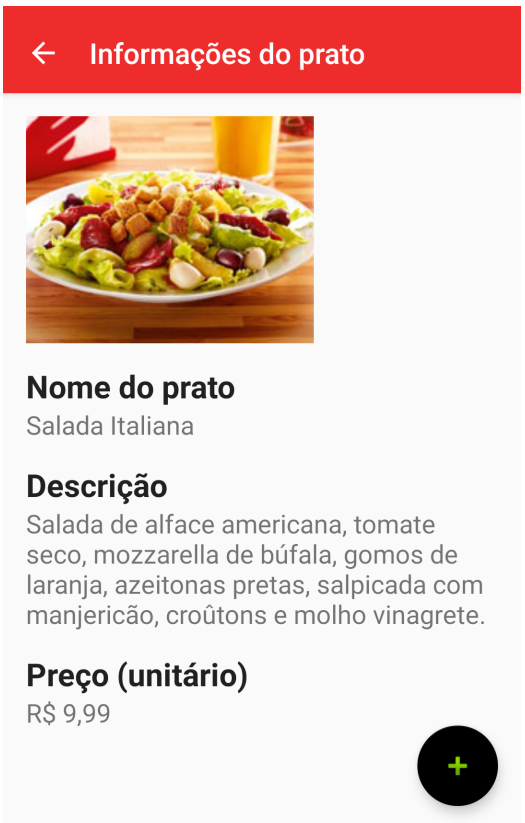
(a) Autenticação de usuário



(b) Tela principal após autenticação



(c) Inicialização de novo pedido

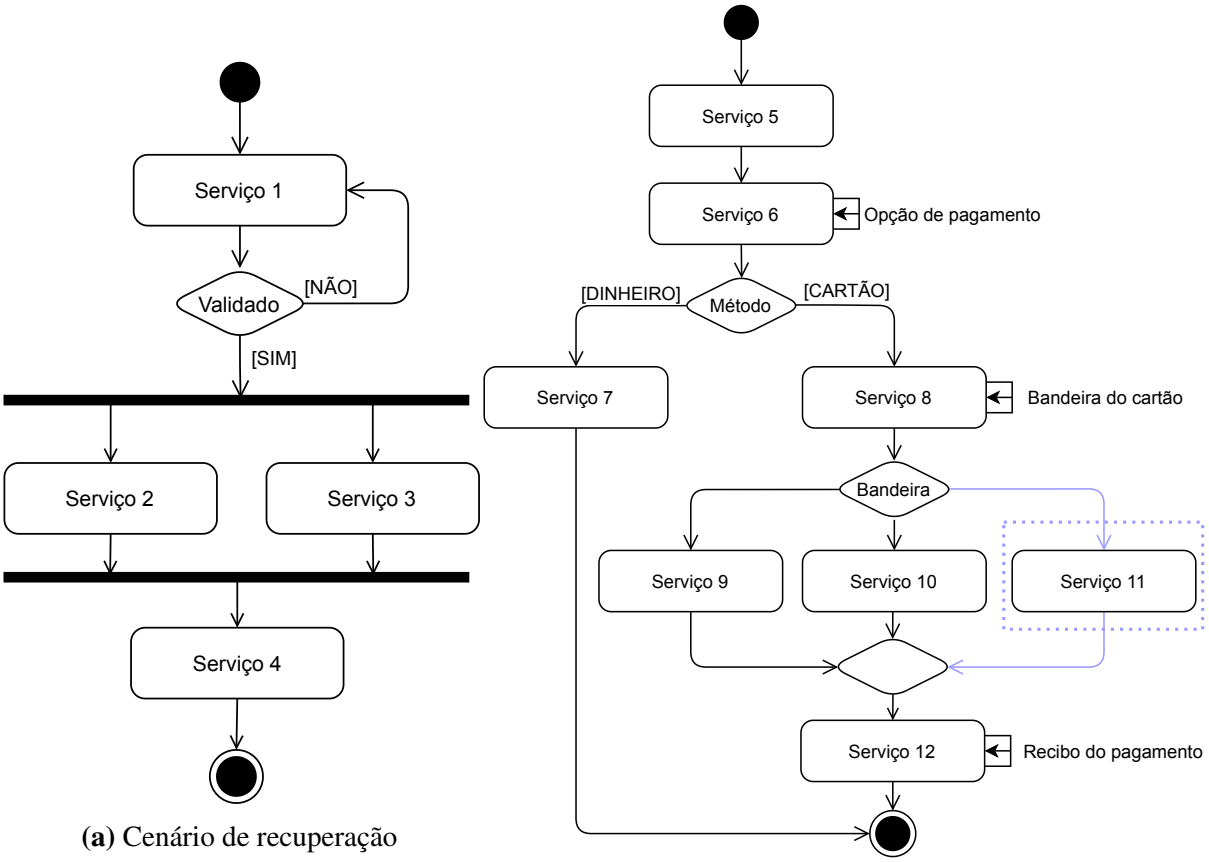


(d) Informações do prato

Fonte: Elaborada pelo autor

alternativos da lista com o passar do tempo. No que diz respeito ao processo automatizado, o componente Monitor deve iniciar uma pesquisa no Repositório de Serviços Web para encontrar um serviço semelhante. Vale ressaltar que essa pesquisa pode retornar um conjunto de serviços classificado por medidas estatísticas para atender aos parâmetros inicialmente fornecidos. Por fim, uma vez selecionado, o serviço alternativo assume o papel do preferencial até que o mesmo se torne disponível novamente.

Figura 38 – Possíveis cenários de adaptação envolvidos na aplicação *App2Rest*



(a) Cenário de recuperação

(b) Cenário de evolução

Fonte: Elaborada pelo autor

A Figura 38b ilustra um cenário de adaptação que representa uma evolução do sistema. Para evitar fornecer uma descrição redundante em relação ao cenário anterior, apenas os passos que caracterizam esse cenário são reportados. De maneira geral, esse cenário envolve o fechamento da comanda e pagamento da mesma. Assim, após solicitar o fechamento de seu pedido (`Serviço 5`), o usuário é encaminhado ao ambiente de pagamento (`Serviço 6`). Caso o cliente decida pagar em dinheiro, o recebimento é processado pelo `Serviço 7` e o atendimento é finalizado. Porém, caso o cliente escolha cartão como método de pagamento, os dados do cartão devem ser encaminhados a um serviço capaz de reconhecer a bandeira utilizada pelo cartão (`Serviço 8`) e enviar uma solicitação de pagamento ao serviço da operadora responsável

pela bandeira (`Serviço 9` e `Serviço 10`). Na impossibilidade de processar pagamentos para uma determinada bandeira, uma exceção será lançada pelo sistema durante a etapa de pagamento. Tal exceção, após analisada e interpretada pelo módulo de planejamento e tomada de decisões, deve resultar na tentativa de inclusão de um novo serviço, o qual deve ser capaz de processar a solicitação inicial do usuário. Para isso, uma pesquisa deve ser realizada no Repositório de Serviços Web, resultando na adição do `Serviço 11`. Dessa forma, ocorre uma evolução da aplicação através da inclusão de um novo serviço, o qual será utilizado para atender a nova demanda. Finalmente, o pagamento é processado pelo `Serviço 12` e um recibo é fornecido ao cliente.

Para ilustrar a configuração necessária para a utilização do *framework* DynaMS, o `Serviço 2` (Figura 38a) será considerado, a partir do qual serão abordadas as seguintes operações: (i) tradução dos dados do sistema para um modelo em JSON; e (ii) tradução desse modelo (passo i) para a ontologia OWL-S. A seguir, Listagem 6, é apresentado o código fonte da operação `getUserInfo`, utilizada pelo `Serviço 2` para a recuperação dos dados de um cliente. Dado um nome de usuário, representado pela variável `user` (linha 2), essa operação recupera o conjunto de informações do usuário em particular, tais como: nome completo, CPF, endereço e telefone. Para isso, o método `getByUser` (linha 11) é invocado e uma pesquisa na base de dados do sistema é realizada, a qual pode retornar dois possíveis resultados: (i) uma instância da classe `Cliente` contendo todos os dados do usuário pesquisado; ou (ii) um valor nulo, que indica a inexistência desse usuário no sistema. Por fim, os dados recuperados são transformados em JSON (linha 12) e fornecidos à aplicação requisitante (linha 14). Em relação ao processo de conversão utilizado, optou-se por utilizar JSON pelos seguintes fatores: (i) facilidade em serialização e desserialização de objetos em formato texto; e (ii) compactação de dados. Esses fatores minimizam o custo computacional das operações de conversão, além de reduzir o tráfego de rede durante a comunicação entre a aplicação cliente e o servidor provedor de serviços Web.

Listagem 6 – Exemplo de uma operação de recuperação de dados de um usuário

```
1   @WebMethod(operationName = "getUserInfo")
2   public String getUserInfo(@WebParam(name = "user") String user) {
3       Acesso data;
4       Cliente c;
5       ClienteServiceImpl clienteService = new ClienteServiceImpl();
6       String dados;
7
8       Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd").create();
9       data = gson.fromJson(user, Acesso.class);
10
11      c = clienteService.getByUser(data.getUsuario());
12      dados = gson.toJson(c);
13
14      return dados;
15  }
```

Após o desenvolvimento do serviço que será utilizado na aplicação, é necessário incluí-lo em um repositório de serviços Web. Entretanto, como apresentado na Seção 6.3.1, tal repositório dispõe de uma estrutura específica, baseada na utilização de uma ontologia para representação de serviços. Portanto, é necessário traduzir a documentação do `Serviço 2`, expressa através do modelo de serviço Web utilizado pelo *framework* DynaMS, para a ontologia OWL-S, utilizada pelo repositório de serviços. Para isso, o processo de mapeamento mostrado na Figura 34 (p. 110) deve ser seguido, o qual resultará na representação dos dados do `Serviço 2` (por exemplo, *endpoint* e conjunto de operações e seus respectivos parâmetros), dentro do vocabulário utilizado pela ontologia OWL-S. A Listagem 7 mostra a ontologia OWL-S resultante do processo de mapeamento. Por motivos de objetividade e concisão, optou-se por apresentar apenas o trecho da tradução correspondente a operação `getUserInfo` do `Serviço 2`. A tradução apresentada está disposta em notação Turtle (veja Seção 4.1) e, consequentemente, utiliza uma sintaxe baseada na tripla `(sujeito, predicado, objeto)`. Entre as linhas 1 e 11 são definidos os prefixos utilizados pela notação e, embora seja possível utilizar as URIs diretamente, a adoção de prefixos facilita a leitura e manipulação dos dados representados pela ontologia. Em seguida, nas linhas 13 a 18 são definidos quais vocabulários são importados, ou seja, qual o conjunto de classes, relações e propriedades que compõem a ontologia. A interpretação desses dados permite identificar que os dados armazenados no repositório utilizam o conjunto de termos declarados nos vocabulários `Service, Process, Profile` e `Grounding`, os quais compõem a ontologia OWL-S. Finalmente, as demais linhas mostram o conjunto de indivíduos que estão sendo representados. Entre as linhas 23 e 27 é declarada a visão mais geral do serviço (`Service`), explicitando onde estão especificados os dados mais detalhados dos demais vocabulários. As linhas 29 a 35 definem o `Profile` do serviço, o qual apresenta uma versão mais geral do serviço, especificando atributos como, por exemplo, nome, descrição sobre quais são seus objetivos gerais, versão e uma conexão para a descrição dos processos que disponibiliza. Entre as linhas 37 e 43 é apresentado o `Grounding` do serviço, parte onde são mostrados os detalhes referentes ao *endpoint* de conexão e o caminho para a documentação original do serviço. Já as linhas 45 a 47 mostram uma lista em que cada elemento aponta para um dos processos definidos por aquele serviço (`Process`). Por estar sendo representado apenas um processo, apenas uma entrada (`getUserInfo`, linha 47) é exibida. Por fim, as linhas 49 a 62 mostram a definição mais detalhada de um processo, ou seja, suas informações mais específicas, o conjunto de entradas e tipos utilizados e qual é a resposta produzida.

Listagem 7 – Operação parte do serviço Web traduzida para a ontologia OWL-S

```
1 @prefix : <http://ws.ra4selfmobapps.rc.unesp.br/repository/App2Rest#> .
2 @prefix owl: <http://www.w3.org/2002/07/owl#>.
3 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
4 @prefix xml: <http://www.w3.org/XML/1998/namespace>.
5 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
6 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
```

```
@prefix Process: <http://www.daml.org/services/owl-s/1.2/Process.owl#>.
@prefix Profile: <http://www.daml.org/services/owl-s/1.2/Profile.owl#>.
@prefix Service: <http://www.daml.org/services/owl-s/1.2/Service.owl#>.
@prefix Grounding: <http://www.daml.org/services/owl-s/1.2/Grounding.owl#>.
@base <http://ws.ra4selfmobapps.rc.unesp.br/repository/App2Rest>.

<http://ws.ra4selfmobapps.rc.unesp.br/repository/App2Rest>
  rdf:type owl:Ontology ;
  owl:imports <http://www.daml.org/services/owl-s/1.2/Service.owl> ,
    <http://www.daml.org/services/owl-s/1.2/Process.owl> ,
    <http://www.daml.org/services/owl-s/1.2/Profile.owl> ,
    <http://www.daml.org/services/owl-s/1.2/Grounding.owl> .

#################################################################
#    Individuals
#################################################################
:ClientService rdf:type owl:NamedIndividual ,
  Service:Service ;
  Service:describes :getUserInfoProcesses ;
  Service:presents :getUserInfoProfile ;
  Service:supports :getUserInfoGrounding .

:getUserInfoProfile rdf:type owl:NamedIndividual ,
  Profile:Profile ;
  Profile:has_process :getUserInfoProcesses ;
  Service:presentedBy :ClientService ;
  Profile:serviceName "ClientService"^^xsd:string ;
  Profile:textDescription "This service aims at providing a set of
      ↪ operations to handle client information"^^xsd:string ;
  owl:versionInfo "1"^^xsd:int .

:getUserInfoGrounding rdf:type owl:NamedIndividual ,
  Grounding:WsdlGrounding ;
  Grounding:owlsProcess :getUserInfo ;
  Grounding:operation "getUserInfo"^^xsd:anyURI ;
  Grounding:portType "ClientService"^^xsd:anyURI ;
  Grounding:wsdlDocument
      ↪ "http://localhost:8080/App2Rest/ClientService?wsdl"^^xsd:anyURI ;
  Grounding:wsdlPort
      ↪ "http://localhost:8080/App2Rest/ClientService?wsdl"^^xsd:anyURI .

:getUserInfoProcesses rdf:type owl:NamedIndividual ,
  Process:Process ;
  Process:performedBy :getUserInfo .

:getUserInfo rdf:type owl:NamedIndividual ,
  Process:AtomicProcess ;
  Grounding:hasAtomicProcessGrounding :getUserInfoGrounding ;
  Process:hasInput :getUserInfoInput_info ;
  Process:hasOutput :getUserInfoResponse ;
  Process:name "getUserInfo"^^xsd:string ;
  rdfs:comment "This process is responsible for retrieving all the
```

```
        ↪ information regarding an user" .
56
57  :getUserInfoInput_info rdf:type owl:NamedIndividual ,
58    Process:Input ;
59    Process:parameterType
        ↪ "https://www.w3.org/2001/XMLSchema#String"^^xsd:anyURI .
60
61  :getUserInfoResponse rdf:type owl:NamedIndividual ,
62    Process:Output ; Process:parameterType
        ↪ "https://www.w3.org/2001/XMLSchema#String"^^xsd:anyURI .
```

Por fim, após o desenvolvimento do conjunto de serviços que serão utilizados e suas respectivas traduções e inserções em um repositório de serviços, a configuração de um serviço simples ou de uma composição no contexto do *framework* DynaMS deve ser apresentada, como mostra a Listagem 8. Para isso, assume-se a existência de duas instâncias do `Serviço 2`, devidamente traduzidas e cadastradas em um repositório de serviços. Esse serviço é do tipo SOAP e possui um conjunto de operações relacionadas ao gerenciamento de clientes como, por exemplo, a operação `getUserInfo` (Listagem 6, p.123).

Listagem 8 – Exemplo de uma coreografia com a capacidade de adaptação

```
1     //Carrega o modelo do serviço a partir do WSDL
2     WSSoapModel wsm =
          ↪ ServiceParser.getWebServiceInformationFromWSDL(prefService);
3     WSSoapModel wsmAlt =
          ↪ ServiceParser.getWebServiceInformationFromWSDL(altService);
4
5     //Define operação de recuperação de dados do usuário e vincula o
          ↪ método "getUserInfo" (localizado no serviço preferencial)
6     WebService getUserInfoOperation = new WebService();
7     getUserInfoOperation.setWebMethod(wsm.getWebMethods()
8                                         .get("getUserInfo"));
9
10    //Define dados de entrada: String com o nome de usuário do cliente
11    getUserInfoOperation.setInputs("user", username);
12
13    //Define coreografia e os serviços (preferencial e alternativo)
14    WSOrchestration wso = new WSOrchestration();
15    wso.setOperations(getUserInfoOperation, getUserInfoOperationAlt);
16
17    //Executa a coreografia e salva o resultado (String JSON serializada
          ↪ com as informações do usuário ou nula)
18    String result = (String) wsc.run();
```

A seguir, uma descrição para cada um dos procedimentos realizados durante a configuração desse serviço (Listagem 8) é apresentada, sendo esta organizada em um conjunto de quatro passos, a saber:

Passo 1 – Extração de dados do serviço. Neste passo é mostrada a extração do conjunto de

informações do `Serviço 2`. As linhas 2 e 3 mostram, respectivamente, a obtenção dos dados do serviço preferencial (representado pela variável `urlprefService`) e de um serviço alternativo (representado pela variável `altService`). Nesse exemplo, parte-se do pressuposto que os serviços já foram pesquisados em um repositório e as URLs são previamente conhecidas pelo desenvolvedor. A extração das informações dispostas no documento WSDL resultará na criação de um objeto da classe `WSSoapModel`, no qual estarão todas as informações técnicas do serviço em questão. É importante ressaltar que em um cenário que envolvesse um serviço RESTful, o método `getWebServiceInformationFromSwagger` seria requisitado, sendo informada a URL do *endpoint* de documentação provido pelo *Swagger* e o resultado seria um objeto da classe `WSRestModel`.

Passo 2 – Especificação dos métodos utilizados. Após a leitura do conjunto de informações do serviço que será utilizado, é necessário especificar quais métodos serão utilizados. Para isso, é necessário instanciar um novo objeto do tipo `WebService`, através do qual serão representados o serviço Web (linha 6), o método que será executado na coreografia (linha 7) e o conjunto de parâmetros que devem ser informados ao serviço (linha 11).

Passo 3 – Definição da coreografia de serviços. Em seguida, é necessário inicializar uma nova coreografia (linha 14) e definir quais operações que a compõem. Nesse exemplo, optou-se por desenvolver uma coreografia com apenas uma operação, sendo utilizada apenas as definições de um serviço simples (`getUserInfoOperation`) e de seu serviço alternativo (`getUserInfoOperationAlt`), os quais permitem ilustrar um cenário de adaptação em função de falha ou degradação de QoS. Para a adição de mais serviços dentro dessa coreografia, todo o processo seria análogo, sendo necessário obter as informações do serviço em questão (conjunto de operações, parâmetros e resultados produzidos), definir uma operação e seu conjunto de entradas e, finalmente, inseri-la dentro da coreografia. Em outras palavras, é necessário executar novamente o passo 1 para cada serviço envolvido na coreografia e o passo 2 para cada uma das operações que serão executadas.

Passo 4 – Execução do serviço. Finalmente, após a definição da coreografia, basta executá-la e armazenar o resultado produzido (linha 18). É importante ressaltar a importância de se fazer um *casting* do tipo de saída produzido pelo método `run()`, pois, por trabalhar com diferentes tipos de dados, será devolvido um objeto Java genérico. Como o serviço que está sendo consultado produz uma *string* JSON, o valor final da variável `result` será uma *string* contendo um objeto representado através da notação JSON. Portanto, cabe ao desenvolvedor da aplicação cliente transformar esse resultado para o modelo de classes utilizado por sua aplicação por meio de uma atividade de desserialização do objeto.

7.2 Comparação com outros *frameworks*

De maneira complementar ao estudo de caso apresentado na Seção 7.1, uma avaliação direcionada ao conjunto de recursos providos pelo *framework* DynaMS foi conduzida. Para isso, foi realizada uma nova seleção no conjunto de estudos do mapeamento sistemático apresentado no Capítulo 5. Tal seleção teve como objetivo identificar os estudos que apresentassem evidências claras das funcionalidades disponibilizadas pelos *frameworks* propostos, além das abordagens e tecnologias adotadas durante suas respectivas fases de desenvolvimento. Portanto, essa seleção focou na inclusão de estudos que mostrassem detalhes sobre a implementação dos *frameworks* propostos e o conjunto de características a partir das quais são compostos. A realização dessa seleção tornou-se necessária devido ao alto número de estudos obtidos por meio de nossa revisão da literatura. Vale relembrar que foram recuperados 71 estudos, dos quais 65 foram originados a partir do mapeamento sistemático e outros 6 obtidos a partir da pesquisa realizada em bases nacionais. Inicialmente, para facilitar a extração das características de cada *framework*, foram elaborados três critérios de seleção mais específicos, de maneira que fosse possível selecionar um grupo reduzido de estudos que detalhassem o maior número possível de características e, ao mesmo tempo, um alto grau de compatibilidade com o *framework* DynaMS, possibilitando estabelecer um comparativo justo para todos os envolvidos. Tais critérios são apresentados a seguir:

1. **O estudo deve apresentar ao menos um método de avaliação.** Este critério visa selecionar estudos que apresentem algum indicativo de avaliação, que pode ser um estudo de caso, protótipo ou utilização de uma aplicação já existente. Esse critério foi definido por três motivos: (i) mesmo que o autor não tenha apresentado explicitamente recursos e/ou tecnologias utilizados por seu *framework*, avaliações mais práticas facilitam inferir as respostas para o conjunto de questionamentos utilizados durante o comparativo, maximizando a potencial extração de dados sobre o trabalho avaliado; (ii) proximidade com um dos métodos de avaliação adotados para avaliar o *framework* DynaMS, facilitando a definição de parâmetros para a comparação e tornando-a mais justa; e (iii) a condução de uma avaliação prática. Dentro desse cenário, trabalhos que envolveram o desenvolvimento de uma aplicação receberam maior prioridade, pois para que uma aplicação possa utilizar o *framework* proposto por seu respectivo autor é necessário que ao menos uma implementação parcial do *framework* proposto tenha sido realizada;

2. **Deve especificar as relações de serviço que suporta.** Este critério tem por objetivo definir quais estratégias de gerenciamento de serviços são utilizadas pelos *frameworks* propostos, como, por exemplo, descoberta, seleção, composição e orquestração de serviço. Durante a revisão da literatura foram encontrados estudos que focam em apenas uma estratégia de gerenciamento e, embora tenham sua importância para responder as questões

de pesquisa do mapeamento (ver Apêndice A), não retratam a totalidade do processo de substituição. Portanto, a aplicação desse critério auxilia a categorização das propostas de acordo com a abrangência de interação com serviços Web. Sendo assim, preferiu-se estudos que apresentassem maior abrangência, ou seja, envolvessem todos os processos desde a descoberta de um novo serviço até sua substituição, e possibilitassem a criação de novas funcionalidades através da composição e orquestração de serviços. Entretanto, para que um estudo se enquadre nesse critério é necessário especificar ao menos a habilidade de efetuar a substituição de serviço, etapa que representa a principal forma de adaptação proposta neste trabalho; e

3. **Deve especificar como a seleção do serviço candidato a substituição ocorre.** Por fim, o objetivo deste critério é analisar o funcionamento da seleção de serviços. Portanto, as respostas esperadas para esse critério visam apresentar detalhes sobre os seguintes itens: (i) tipo de estrutura de dados utilizada quando um serviço é substituído em tempo de execução; (ii) técnica de similaridade utilizada para selecionar um serviço nas fases de *design* e *runtime*; e (iii) método de avaliação e recomendação de serviços quando uma atividade de adaptação é realizada. Nesse sentido, avaliação individual de serviços, parâmetros de avaliação adotados e métricas de QoS são exemplos para esse critério.

A aplicação desse conjunto de critérios resultou em 5 estudos selecionados para um comparativo com o *framework* DynaMS, proposto neste trabalho. A seguir, a Tabela 7 sintetiza os resultados obtidos a partir dessa avaliação. A coluna "Característica" identifica a característica que está sendo avaliada e as demais colunas os estudos que estão sendo avaliados. O conjunto de estudos comparado é rotulado de acordo com sua identificação atribuída no mapeamento sistemático, os quais também foram disponibilizados de maneira sintetizada na Tabela 12 (veja Apêndice B). Linhas preenchidas com a sigla "NI" indicam que a característica representada por aquela linha não foi informada pelo estudo avaliado, o que pode significar que o respectivo *framework* não suporta determinada característica ou que suporta, mas o autor preferiu não detalhar sua existência naquele estudo. Esse conjunto de características foi definido considerando em particular os dados obtidos durante as fases de extração e síntese das informações de cada estudo avaliado pelo mapeamento. O objetivo a partir dessa tabela é oferecer uma comparação direta entre as abordagens e tecnologias utilizadas pelo *framework* DynaMS e os demais *frameworks* encontrados nos estudos analisados neste trabalho. Adicionalmente, a condução desse comparativo permite destacar pontos de atuação em que o *framework* DynaMS se sobressai e também quais são os pontos que podem necessitar de uma atenção extra durante o desenvolvimento de futuras versões desse trabalho.

Como pode-se observar na Tabela 7, o comparativo foi estabelecido com base em um conjunto de 15 características, as quais foram consideradas suficientes para determinar uma

Tabela 7 – Comparativo entre o *framework* DynaMS e os propostos pelos estudos obtidos através do mapeamento da literatura

Característica \ Estudo	DynaMS	S4	S22	S44	S61	S63
Modelo de adaptação	MAPE-K	MAPE-K	Agentes	MAPE-K ou Agentes	NI	NI
Tipo de sistema	Reativo	Reativo	Orientado a objetivos	Reativo	Reativo	Proativo
Abordagem de adaptação	Externa	Externa	Externa	Externa	NI	Externa
Protocolos de comunicação	SOAP e REST	SOAP	SOAP	SOAP	SOAP	SOAP
Suporta composição de serviços	Sim	NI	Sim	Sim	Sim	Sim
Suporta orquestração de serviços	Sim	NI	Sim	NI	Sim	NI
Seleção em tempo de execução	Sim	Sim	Sim	Sim	Sim	Sim
Armazena serviços alternativos	Sim	Sim	NI	NI	Sim	Não
Método de armazenamento de serviços alternativos	Lista de compatíveis	Planos com um serviço vinculado	Lista de compatíveis	NI	Lista de compatíveis	NI
Estratégia de seleção de serviços	Técnica e Semântica	NI	Técnica e Semântica	Técnica e Semântica	Técnica	Técnica
Estrutura de armazenamento de serviços (Repositório)	Ontologia (Grafo)	UDDI	UDDI	UDDI	UDDI	NI
Repositório facilita extensão	Sim	Não	Não	Não	Não	NI
Monitoramento de QoS	Sim	Sim	Sim	Sim	Sim	Sim
Ranking de serviços	Sim	Não	Sim	Sim	Sim	NI
Recomendação de serviços	Sim	Sim	NI	Sim	Sim	Sim

Fonte: Elaborada pelo autor

visão geral das principais funcionalidades e abordagens utilizadas em cada um dos *frameworks* analisados. Com relação aos resultados, os seguintes aspectos podem ser destacados:

- O suporte a serviços baseados em SOAP ou REST é um diferencial do *framework* DynaMS, garantindo maior abrangência em relação aos protocolos de comunicação mais comumente utilizados e, consequentemente, quanto aos tipos de serviços que podem ser utilizados pelo *framework* durante o desenvolvimento de *Self-Apps*; e
- Todos os *frameworks* avaliados suportam seleção de serviços por meio da realização de buscas de caráter técnico. Adicionalmente, outros três (DynaMS e os *frameworks* apresentados nos estudos S22 e S44) suportam a realização de pesquisas de caráter semântico, possibilitando avaliar os serviços a partir do contexto em que atuam. Entretanto, o *framework* DynaMS mostrou-se o único a adotar a utilização de um repositório baseado em uma ontologia, sendo este também um diferencial do *framework*. A utilização desse tipo de repositório se mostrou relevante, pois modificações no vocabulário utilizado implicam diretamente na possibilidade de adicionar suporte para novos tipos de serviços. Isso pode ser considerado um importante fator quanto à facilidade de extensão do repositório em futuras versões do *framework* que adotem diferentes arquiteturas de serviços.

7.3 Considerações finais

Este capítulo teve como objetivo oferecer uma avaliação do *framework* DynaMS, a qual foi subdivida em dois diferentes aspectos: (i) prático, cujo objetivo foi, através de um estudo de caso, oferecer uma visão mais direcionada a utilização do *framework* e suas possíveis aplicações em uma aplicação do mundo real. Para isso, foram apresentados detalhes sobre um sistema para um restaurante inteligente que utiliza um conjunto de serviços Web para seu funcionamento e que, em uma situação de falha, devem ser reparados tão rápido quanto possível. Esse procedimento de reparo deve ser realizado preferencialmente sem a necessidade de interação humana, o que descreve um cenário de adaptação; e (ii) empírico, cenário voltado para a análise de características, técnicas e tecnologias adotadas pelo *framework* em comparação com outros estudos encontrados durante o mapeamento da literatura apresentado no Capítulo 5. A análise dessa tabela evidencia que o *framework* DynaMS incorpora novas funcionalidades e abordagens, tais como: inclusão do suporte a serviços RESTful e proposta de desenvolvimento de um repositório de serviços específico para as necessidades do *framework*.

8 Conclusão

Este trabalho abordou o projeto do *framework* DynaMS, cujo objetivo é apoiar o desenvolvimento de *Self-Apps*. Em linhas gerais, o *framework* lida com três tópicos importantes para o contexto desse tipo de aplicação, a saber: implantação dinâmica, métricas de QoS e técnicas de seleção de serviços. Esse tipo de aplicação deve identificar as anomalias internas, relacionadas aos serviços Web, e/ou externas, relacionadas ao ambiente de execução, e corrigi-las sem a percepção de seus usuários. Conforme apresentado neste trabalho, o desenvolvimento de *Self-Apps* pode ser considerado uma atividade não trivial, pois engloba o conhecimento de várias áreas de pesquisa. Inicialmente, Capítulos 2 à 4, foram apresentados os assuntos relacionados ao tema de pesquisa desse *framework*. Em seguida, Capítulo 5, foi apresentada uma revisão da literatura, cujas evidências nortearam o projeto do *framework* DynaMS, além de revelar que o tema deste trabalho tem se mantido como oportunidade de investigação. O projeto e a avaliação conduzidas em tal *framework* foram apresentadas, respectivamente, nos Capítulos 6 e 7. Por fim, este capítulo apresenta as conclusões deste trabalho da seguinte maneira: a Seção 8.1 apresenta as contribuições da dissertação; a Seção 8.2 reporta as principais dificuldade e limitações enfrentadas durante o desenvolvimento do *framework* DynaMS; e a Seção 8.3 elenca as principais perspectivas futuras e oportunidades de investigação.

8.1 Contribuições da dissertação

Conforme reportado no Capítulo 5, uma revisão da literatura foi realizada, sendo parte dessa revisão um mapeamento sistemático, o qual foi conduzido seguindo as diretrizes estabelecidas por Kitchenham e Charters (2007), Kitchenham, Dyba e Jorgensen (2004), Petersen *et al.* (2008), Petersen, Vakkalanka e Kuzniarz (2015). Esse mapeamento teve como objeto de investigação oito questões de pesquisa, o que nos permitiu elaborar um panorama completo sobre o tema de pesquisa deste trabalho, assim como identificar as principais lacunas e assuntos em aberto sobre o tema. Portanto, esse mapeamento pode ser considerado como uma contribuição deste trabalho, pois acredita-se que esse panorama possa oferecer aos profissionais da indústria e comunidades científicas um guia para nortear o aprimoramento e desenvolvimento de novas soluções para o domínio de *Self-Apps*.

Em paralelo, os resultados do mapeamento apresentado no Capítulo 5 revelam um conjunto de iniciativas para apoiar o desenvolvimento de *Self-Apps*. No entanto, como sintetizado pela Tabela 7 (p. 130), nenhuma dessas iniciativas apresenta uma solução completa para apoiar o desenvolvimento de *Self-Apps*. Nesse sentido, o *framework* proposto pode ser considerado

como a principal contribuição deste trabalho, pois cobre um conjunto maior de características e se mostra uma solução factível para contornar os problemas relacionados ao conhecimento e grau de complexidade envolvidos na utilização de determinadas tecnologias e recursos para o desenvolvimento de *Self-Apps*. Assim, em decorrência dessa abstração de complexidade e da automatização de certas etapas do desenvolvimento desse tipo de aplicação, eficiência, custo, suporte técnico, segurança, integração e usabilidade podem ser mencionados como os principais benefícios proporcionados pela adoção desse *framework*.

A avaliação do *framework* DynaMS pode ser considerada como mais uma contribuição deste trabalho, pois fornece indicativos quanto a aplicabilidade e comportamento do mesmo quando aplicado em um cenário real de funcionamento. A avaliação conduzida pode ser sintetizada em duas etapas: (i) condução de um estudo de caso para o domínio de sistemas de informação; e (ii) elaboração de um comparativo com outros *frameworks* selecionados pela revisão da literatura. Na primeira etapa, uma aplicação para o gerenciamento de restaurantes foi desenvolvida, a qual é composta por duas camadas: (i) *front-end* para dispositivos móveis (sistema operacional Android) e Web; e (ii) *back-end* baseado em serviços desenvolvidos utilizando a linguagem Java. Ainda sobre essa etapa, vale destacar que tanto o domínio do sistema quanto a técnica de avaliação utilizada foram os mais encontrados nos estudos apresentados em nosso mapeamento da literatura (veja Capítulo 5). Na segunda etapa, o *framework* DynaMS foi comparado com outros *frameworks* encontrados na literatura, mostrando ser o mais completo quando analisado em relação a um conjunto de 15 características. Diante do exposto, vale destacar que a avaliação realizada permitiu obter parâmetros quanto a aplicabilidade, funcionamento e aderência aos requisitos especificados, sendo complementada por um comparativo entre as características mais comuns de *frameworks* para *Self-Apps* e os diferenciais apresentados neste trabalho.

Por fim, vale destacar que o *framework* DynaMS e sua temática podem estar presentes em diferentes domínios de sistema de software. Nesse sentido, este trabalho pode ser um ponto de partida/base para futuras iniciativas relacionadas ao desenvolvimento e/ou otimização desse tipo aplicação (*Self-Apps*).

8.2 Dificuldades e limitações

Neste trabalho foram priorizados a especificação e o desenvolvimento de um *framework* abrangente para os possíveis diferentes cenários envolvendo *Self-Apps*, objetivando garantir sua compatibilidade com o maior número de aplicações possíveis. Entretanto, ainda é possível que desenvolvedores apresentem necessidades específicas, conforme o conjunto de requisitos e os domínios de aplicação envolvidos. Portanto, embora seja possível afirmar que o *framework* proposto é capaz de suportar diferentes aplicações e métodos de comunicação SOAP e REST,

não é possível afirmar que o *framework* é capaz de satisfazer todos os possíveis cenários de desenvolvimento. Entretanto, é importante ressaltar a capacidade de extensão do *framework* DynaMS para que se adéque aos requisitos e/ou necessidades dos desenvolvedores. Por exemplo, embora o *framework Swagger* tenha sido utilizado para o gerenciamento de serviços RESTful, o desenvolvedor tem a flexibilidade de realizar os ajustes necessários para utilizar uma outra ferramenta de documentação de sua preferência, como mencionado na Seção 6.2.5.

Por fim, embora o estudo de caso conduzido apresente o funcionamento e viabilidade do *framework* DynaMS, não foi possível, por exemplo, conduzir uma avaliação direcionada para o aspecto de performance, em especial, o estabelecimento de comparações com outros *frameworks* similares existentes na literatura. Tal limitação ocorreu especificamente em virtude das limitações de tempo inerentes em um projeto de mestrado acadêmico.

8.3 Trabalhos futuros

Durante a execução deste trabalho foram observadas três atividades como Trabalhos Futuros (TF), a saber:

TF-1. Conduzir outros estudos de casos e experimentos para verificar o comportamento do *framework* DynaMS e sua infraestrutura, submetendo-o a outros cenários de utilização, tais como: experiência do usuário e escala do ambiente. Além disso, espera-se obter, com esses novos tipos de avaliação, parâmetros importantes sobre o comportamento de três funções do *framework*, a saber: (i) módulo de avaliação de serviços; (ii) implantação dinâmica de serviços; e (iii) técnicas de busca em repositório;

TF-2. Conduzir estudos de casos em outros domínios de software. A avaliação apresentada no Capítulo 7 foi direcionada em aplicações do domínio de sistemas da informação. Seria interessante avaliar o comportamento do *framework* DynaMS em aplicações voltadas para o domínio de sistemas críticos, como, por exemplo, um sistema para cuidado de pacientes dentro de um ambiente doméstico. Esse sistema possui níveis de criticidade mais elevado e a ocorrência de anomalias pode resultar em sérios danos ao paciente que está sendo monitorado. Nesse domínio, o comportamento dos itens supracitados (TF-1) são extremamente relevantes, visto que esse tipo de aplicação demanda alta confiabilidade e resiliência. Em ambos os casos (TF-1 e TF-2) espera-se ter ambientes de maior escala e menos controlados, expondo o *framework* aos possíveis cenários de falha oriundos da implantação da aplicação em ambiente de produção e da interação com um grupo de usuários; e

TF-3. Acompanhar a evolução da área de pesquisa deste projeto através da atualização do mapeamento sistemático reportado no Capítulo 5. Dessa forma, espera-se ter parâmetros

sobre as técnicas utilizadas, bem como o interesse da comunidade científica e da indústria por esse tipo de aplicação.

Por fim, além dos trabalhos futuros supracitados, este trabalho também abre perspectiva para a investigação de outros tópicos relacionados ao desenvolvimento de *Self-Apps* e seu relacionamento com as tecnologias abordadas por essa temática, a saber:

- Estudo do modelo de desenvolvimento proposto por microsserviços. O objetivo desse estudo é determinar a viabilidade de utilizar o *framework* DynaMS também em serviços desenvolvidos baseado nesse estilo arquitetural. Embora seja possível estimar sua viabilidade de aplicação, existem diversas outras variáveis que devem ser analisadas antes de realizar qualquer afirmação relacionada a capacidade do *framework* de oferecer suporte para esse tipo de aplicação. Nesse sentido, o estudo de microsserviços e de sua viabilidade de utilização ou dos procedimentos necessários para adaptação do *framework* DynaMS mostra-se um importante trabalho a ser conduzido futuramente; e
- Aprimoramento do sistema de monitoramento, de maneira que seja possível distinguir quais serviços apresentam maior tendência a falhas ou degradações. Para esse cenário, a utilização de elementos da computação autoconsciente em conjunto com algoritmos de aprendizado mostra-se uma abordagem interessante para o desenvolvimento de um futuro trabalho. Essa problemática é evidenciada por Ye *et al.* (2018), onde são explorados os benefícios relacionados a esse tipo de abordagem. Em particular, os autores comentam sobre a possibilidade de balancear a quantidade de recursos utilizada no monitoramento de serviços e, consequentemente, reduzir os custos computacional e financeiro relacionados a essa operação. Ao determinar quais serviços apresentam maior chance de apresentar anomalias, torna-se possível monitorá-los de maneira mais efetiva. Através dessa abordagem, aumentam-se as chances de detecção de anomalias, reduzindo o tempo potencialmente decorrido entre as etapas de detecção e correção e, consequentemente, resultando em um melhor aproveitamento dos recursos alocados para a tarefa de monitoramento.

Referências

ABDELHAK, E.; FETH-ALLAH, H.; MOHAMMED, M. Qos uncertainty handling for an efficient web service selection. In: *Proceedings of the 9th International Conference on Information Systems and Technologies*. New York, NY, USA: Association for Computing Machinery, 2019. (icist 2019). ISBN 9781450362924. Disponível em: <https://doi.org/10.1145/3361570.3361592>. Citado na página 25.

ABNT. *NBR ISO 9000: Sistemas de gestão da qualidade - Fundamentos e vocabulário*. ABNT, 2015. Disponível em: <https://www.abntcatalogo.com.br/norma.aspx?ID=345040>. Acesso em: 03/05/2020. Citado na página 37.

ACHTAICH, A. *et al.* Designing a framework for smart iot adaptations. In: BELQASMI, F. *et al.* (Ed.). *Emerging Technologies for Developing Countries*. Cham: Springer International Publishing, 2018. p. 57–66. ISBN 978-3-319-67837-5. Citado na página 25.

AFFONSO, F. J.; LEITE, G.; NAKAGAWA, E. Y. Dms-modeler: A tool for modeling decision-making systems for self-adaptive software domain. In: *The 28th International Conference on Software Engineering & Knowledge Engineering (SEKE' 16)*. California, USA: [s.n.], 2016. p. 617–622. ISSN 2325-9086. Citado na página 97.

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 and Knowledge Engineering. (SEKE' 2015)*. Wyndham Pittsburgh University Center, Pittsburgh, USA: Knowledge Systems Institute Graduate School, 2015. p. 24–29. ISSN 2325-9086. Citado na página 97.

AFFONSO, F. J.; NAKAGAWA, E. Y. A reference architecture based on reflection for self-adaptive software. In: *7th Brazilian Symposium on Software Components, Architectures and Reuse*. [S.l.: s.n.], 2013. p. 129–138. Citado 3 vezes na(s) página(s) 87, 90 e 97.

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 2 vezes na(s) página(s) 87 e 90.

AHMAD, A. *et al.* Smart cyber society: Integration of capillary devices with high usability based on cyber–physical system. *Future Generation Computer Systems*, v. 56, p. 493 – 503, 2016. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X15002563>. Acesso em: 03/05/2020. Citado na página 23.

ALHOSBAN, A. *et al.* Integrating fault-tolerance method into service-oriented computing. In: *2015 IEEE Symposium on Service-Oriented System Engineering*. [S.l.: s.n.], 2015. p. 161–168. Citado na página 23.

ANGARITA, R. *et al.* A knowledge-based approach for self-healing service-oriented applications. In: *The 8th International Conference on Management of Digital EcoSystems*. Biarritz, France: ACM, 2016. (MEDES), p. 1–8. ISBN 978-1-4503-4267-4. Citado na página 88.

ARCAINI, P.; RICCOBENE, E.; SCANDURRA, P. Modeling and validating self-adaptive service-oriented applications. *SIGAPP Appl. Comput. Rev.*, ACM, New York, NY, USA, v. 15, n. 3, p. 35–48, out. 2015. ISSN 1559-6915. Disponível em: <http://doi.acm.org/10.1145/2835260.2835262>. Citado na página 25.

ASCHOFF, R. R.; ZISMAN, A.; ALEXANDRE, P. Parallel adaptation of multiple service composition instances. In: ____. *Engineering Adaptive Software Systems: Communications of NII Shonan Meetings*. Singapore: Springer Singapore, 2019. p. 115–134. ISBN 978-981-13-2185-6. Disponível em: <https://doi.org/10.1007/978-981-13-2185-6_5>. Citado na página 25.

BARRETO, L.; AMARAL, A.; PEREIRA, T. Industry 4.0 implications in logistics: an overview. *Procedia Manufacturing*, v. 13, p. 1245 – 1252, 2017. ISSN 2351-9789. Manufacturing Engineering Society International Conference 2017, MESIC 2017, 28-30 June 2017, Vigo (Pontevedra), Spain. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2351978917306807>. Acesso em: 03/05/2020. Citado na página 21.

BARRY, D. K. Chapter 5 - growing impact of web services. In: BARRY, D. K. (Ed.). *Web Services and Service-Oriented Architectures*. San Francisco: Morgan Kaufmann, 2003, (The Savvy Manager's Guides). p. 61 – 73. ISBN 978-1-55860-906-8. Disponível em: <http://www.sciencedirect.com/science/article/pii/B978155860906850007X>. Acesso em: 03/05/2020. Citado 3 vezes na(s) página(s) 23, 29 e 30.

BENNACEUR, A.; NUSEIBEH, B. Chapter 12 - the many facets of mediation: A requirements-driven approach for trading off mediation solutions. In: MISTRIK, I. *et al.* (Ed.). *Managing Trade-Offs in Adaptable Software Architectures*. Boston: Morgan Kaufmann, 2017. p. 299 – 322. ISBN 978-0-12-802855-1. Acesso em: 03/05/2020. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780128028551000125>. Citado na página 61.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999. (Santa Fe Institute Studies on the Sciences of Complexity). ISBN 9780198030157. Disponível em: <https://books.google.com.br/books?id=fcTcHvSsRMYC>. Acesso em: 03/05/2020. Citado na página 78.

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>. Acesso em: 03/05/2020. Citado na página 42.

CÁMARA, J. *et al.* Self-aware computing systems: Related concepts and research areas. In: ____. *Self-Aware Computing Systems*. Cham: Springer International Publishing, 2017. p. 17–49. ISBN 978-3-319-47474-8. Disponível em: <https://doi.org/10.1007/978-3-319-47474-8_2>. Acesso em: 03/05/2020. Citado 3 vezes na(s) página(s) 41, 48 e 49.

CHAKRABORTY, D.; JOSHI, A. *Dynamic Service Composition: State-of-the-Art and Research Directions*. [S.l.], 2001. Citado na página 33.

CHEN, T.; BAHSOON, R.; YAO, X. A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 3, p. 61:1–61:40, jun 2018. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3190507>. Acesso em: 03/05/2020. Citado 2 vezes na(s) página(s) 51 e 52.

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>. Acesso em: 03/05/2020. Citado 3 vezes na(s) página(s) 24, 42 e 83.

CUGOLA, G. *et al.* Selfmotion: A declarative approach for adaptive service-oriented mobile applications. *Journal of Systems and Software*, v. 92, p. 32 – 44, 2014. ISSN 0164-1212. Citado na página 23.

DAIGNEAU, R. *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. 1. ed. [S.l.]: Addison-Wesley Professional, 2011. ISBN 032154420X. Citado na página 23.

DATTA, S. K. *et al.* Integrating connected vehicles in internet of things ecosystems: Challenges and solutions. In: *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. [S.l.: s.n.], 2016. p. 1–6. Citado na página 22.

DEORA, V. *et al.* A quality of service management framework based on user expectations. In: ORLOWSKA, M. E. *et al.* (Ed.). *Service-Oriented Computing - ICSOC 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 104–114. ISBN 978-3-540-24593-3. Citado na página 38.

DÍAZ, M.; MARTÍN, C.; RUBIO, B. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *Journal of Network and Computer Applications*, v. 67, p. 99 – 117, 2016. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S108480451600028X>. Acesso em: 03/05/2020. Citado na página 22.

DIESTE, O.; GRIMÁN, A.; JURISTO, N. Developing search strategies for detecting relevant experiments. *Empirical Softw. Engg.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 14, n. 5, p. 513–539, oct 2009. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-008-9091-7>. Acesso em: 03/05/2020. Citado na página 152.

DYBA, T.; DINGSOYR, T.; HANSSEN, G. K. Applying Systematic Reviews to Diverse Study Types: An Experience Report. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. [S.l.: s.n.], 2007. p. 225–234. ISSN 1949-3770. Citado na página 153.

EBERHART, R.; SHI, Y.; KENNEDY, J. *Swarm Intelligence*. Elsevier Science, 2001. (The Morgan Kaufmann Series in Artificial Intelligence). ISBN 9780080518268. Disponível em: <https://books.google.com.br/books?id=Z1z3byh1JAsC>. Acesso em: 03/05/2020. Citado na página 78.

ERL, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005. ISBN 0131858580. Citado na página 34.

ERL, T. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007. ISBN 0132344823. Citado 5 vezes na(s) página(s) 23, 25, 29, 30 e 32.

ESFAHANI, N.; ELKHODARY, A.; MALEK, S. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Transactions on Software Engineering*, v. 39, n. 11, p. 1467–1493, Nov 2013. ISSN 2326-3881. Citado na página 42.

FARAHANI, A. *et al.* An evaluation method for self-adaptive systems. In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.: s.n.], 2016. p. 002814–002820. ISSN null. Citado na página 42.

FIONDA, V.; PIRRÒ, G. Ontology: Querying languages and development. In: ____. [S.l.: s.n.], 2018. ISBN 9780128096338. Citado na página 59.

FORESTIERO, A. *et al.* A proximity-based self-organizing framework for service composition and discovery. In: *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. [S.l.: s.n.], 2010. p. 428–437. Citado na página 25.

FORMAN, I. R.; FORMAN, N. *Java Reflection in Action (In Action series)*. Greenwich, CT, USA: Manning Publications Co., 2004. ISBN 1932394184. Citado na página 41.

GARLAN, D.; SCHMERL, B. Model-based adaptation for self-healing systems. In: *Proceedings of the First Workshop on Self-healing Systems*. New York, NY, USA: ACM, 2002. (WOSS '02), p. 27–32. ISBN 1-58113-609-9. Disponível em: <http://doi.acm.org/10.1145/582128.582134>. Acesso em: 03/05/2020. Citado na página 41.

GATOUILLAT, A.; BADR, Y.; MASSOT, B. Qos-driven self-adaptation for critical iot-based systems. In: BRAUBACH, L. *et al.* (Ed.). *Service-Oriented Computing – ICSOC 2017 Workshops*. Cham: Springer International Publishing, 2018. p. 93–105. ISBN 978-3-319-91764-1. Citado na página 23.

GAYO, J. E. L. *et al. Validating RDF Data*. [S.l.]: Morgan & Claypool Publishers, 2017. ISBN 1681731649, 9781681731643. Citado 3 vezes na(s) página(s) 59, 60 e 61.

GHOSH, D. *et al.* Self-healing systems — survey and synthesis. *Decision Support Systems*, v. 42, n. 4, p. 2164 – 2185, 2007. ISSN 0167-9236. Decision Support Systems in Emerging Economies. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167923606000807>. Acesso em: 03/05/2020. Citado na página 89.

GILL, S. *et al.* Radar: Self-configuring and self-healing in resource management for enhancing quality of cloud services. *Concurrency Computation*, v. 31, n. 1, 2019. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85052374573&doi=10.1002%2fcpe.4834&partnerID=40&md5=ea5727e234c50ba4699e42a50cd3c2ad>. Citado na página 23.

GONCALVES, A. Soap web services. In: ____. *Beginning Java EE 7*. Berkeley, CA: Apress, 2013. p. 455–494. ISBN 978-1-4302-4627-5. Disponível em: <https://doi.org/10.1007/978-1-4302-4627-5_14>. Acesso em: 03/05/2020. Citado na página 25.

GRANADOS-COLMENARES, L. *et al.* Radical technological innovations and their impact on society. In: . [S.l.: s.n.], 2018. Citado na página 21.

GUERRA, C. A. N. *Um modelo para ambientes inteligentes baseado em serviços web semânticos*. Dissertação (Mestrado) — Universidade de São Paulo – USP, Instituto de Matemática e Estatística – IME, São Paulo, SP, 8 2007. Citado na página 84.

GUIMARÃES, L. F. *Um Framework para Desenvolvimento de Agentes Autoadaptatitos em Dispositivos Móveis*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, Departamento de Informática, 4 2012. Citado na página 84.

GUPTA, R.; KAMAL, R.; SUMAN, U. A qos-supported approach using fault detection and tolerance for achieving reliability in dynamic orchestration of web services. *International Journal of Information Technology*, v. 10, n. 1, p. 71–81, Mar 2018. ISSN 2511-2112. Disponível em: <https://doi.org/10.1007/s41870-017-0066-z>. Citado na página 25.

HAN, S. N.; CRESPI, N. Semantic service provisioning for smart objects: Integrating iot applications into the web. *Future Generation Computer Systems*, v. 76, p. 180 – 197, 2017. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X16308676>. Acesso em: 03/05/2020. Citado na página 23.

HATTON, L.; SPINELLIS, D.; VAN GENUCHTEN, M. The long-term growth rate of evolving software: Empirical results and implications. *Journal of Software: Evolution and Process*, v. 29, n. 5, p. e1847, 2017. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1847>. Acesso em: 03/05/2020. Citado na página 21.

IBM (Ed.). *An Architectural Blueprint for Autonomic Computing*. [S.l.], 2005. Citado 2 vezes na(s) página(s) 42 e 44.

IEEE Computer Society. *Software and Systems Engineering Vocabulary – (SEVOCAB)*. 2019. On-line. Disponível em: <https://pascal.computer.org/sev_display/index.action>. Acesso em: 03/05/2020. Citado na página 39.

IORIO, A. *et al.* Describing bibliographic references in rdf. *CEUR Workshop Proceedings*, v. 1155, 01 2014. Citado 2 vezes na(s) página(s) 57 e 58.

ITU. *Recommendation E.800 (09/08)*. 2008. On-line. ITU-T Recommendation, versão em inglês. Disponível em: <https://www.itu.int/rec/T-REC-E.800-200809-I/en>. Acesso em: 03/05/2020. Citado na página 38.

JAMSHIDI, P.; AHMAD, A.; PAHL, C. Cloud migration research: A systematic review. *IEEE Transactions on Cloud Computing*, v. 1, n. 2, p. 142–157, July 2013. ISSN 2168-7161. Citado na página 23.

JOIA, P. *et al.* Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, v. 17, n. 12, p. 2563–2571, Dec 2011. Citado na página 71.

JONES, C. *The Technical and Social History of Software Engineering*. 1st. ed. [S.l.]: Addison-Wesley Professional, 2013. 23-33 p. ISBN 0321903420, 9780321903426. Citado na página 22.

JUNIOR, E. C. *SASeS: um framework para o desenvolvimento de aplicações autoadaptativas baseadas em serviços*. Dissertação (Mestrado) — Universidade Estadual do Ceará – UECE, Centro de Ciência e Tecnologia – CCT, Fortaleza, CE, 3 2015. Citado na página 85.

KHANFIR, E.; DJMEAA, R. B.; AMOUS, I. Self-adaptive goal-driven web service composition based on context and qos. In: *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*. [S.l.: s.n.], 2017. p. 201–207. Citado na página 23.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Citado 4 vezes na(s) página(s) 24, 65, 66 e 133.

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. Disponível em: <https://doi.org/10.1016/j.infsof.2010.03.006>. Acesso em: 03/05/2020. Citado na página 153.

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. Disponível em: <http://dl.acm.org/citation.cfm?id=998675.999432>. Acesso em: 03/05/2020. Citado 4 vezes na(s) página(s) 24, 65, 66 e 133.

KRAMER, J.; MAGEE, J. Self-managed systems: an architectural challenge. In: *Future of Software Engineering, 2007. FOSE '07*. [S.l.: s.n.], 2007. p. 259 –268. Citado na página 47.

KRITIKOS, K.; PLEXOUSAKIS, D. Requirements for qos-based web service description and discovery. *IEEE Transactions on Services Computing*, v. 2, n. 4, p. 320–337, Oct 2009. ISSN 1939-1374. Citado na página 38.

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>. Acesso em: 03/05/2020. Citado 2 vezes na(s) página(s) 45 e 46.

LEITNER, P. *et al.* Cost-based prevention of violations of service level agreements in composed services using self-adaptation. In: *2012 First International Workshop on European Software Services and Systems Research - Results and Challenges (S-Cube)*. [S.l.: s.n.], 2012. p. 34–35. Citado na página 25.

LEMOS, R. de *et al.* Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In: LEMOS, R. de *et al.* (Ed.). *Software Engineering for Self-Adaptive Systems III. Assurances*. Cham: Springer International Publishing, 2017. p. 3–30. ISBN 978-3-319-74183-3. Citado na página 24.

LEMOS, R. de *et al.* Software engineering for self-adaptive systems: A second research roadmap. In: ____. *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 1–32. ISBN 978-3-642-35813-5. Disponível em: <https://doi.org/10.1007/978-3-642-35813-5_1>. Acesso em: 03/05/2020. Citado na página 24.

LI, G. *et al.* A self-healing framework for qos-aware web service composition via case-based reasoning. In: ISHIKAWA, Y. *et al.* (Ed.). *Web Technologies and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 654–661. ISBN 978-3-642-37401-2. Citado na página 25.

LI, W.; ZHANG, P.; YANG, Z. A framework for self-healing service compositions in cloud computing environments. In: *2012 IEEE 19th International Conference on Web Services*. [S.l.: s.n.], 2012. p. 690–691. Citado na página 25.

LIMTHANMAPHON, B.; ZHANG, Y. Web service composition with case-based reasoning. In: *Proceedings of the 14th Australasian Database Conference - Volume 17*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2003. (ADC '03), p. 201–208. ISBN

0-909-92595-X. Disponível em: <http://dl.acm.org/citation.cfm?id=820085.820122>. Acesso em: 03/05/2020. Citado na página 32.

MAHDAVI-HEZAVEHI, S. *et al.* A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems. *Information and Software Technology*, v. 90, p. 1 – 26, 2017. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584917302860>. Acesso em: 03/05/2020. Citado 2 vezes na(s) página(s) 42 e 45.

MALYK, N. *The difference Between REST and SOAP APIs*. 2017. On-line. DZone, a Devada Media Property. Disponível em: <https://dzone.com/articles/difference-between-rest-and-soap-api>. Acesso em: 03/05/2020. Citado 2 vezes na(s) página(s) 23 e 37.

MARCHAND, A.; HENNIG-THURAU, T. Value creation in the video game industry: Industry economics, consumer benefits, and research opportunities. *Journal of Interactive Marketing*, v. 27, n. 3, p. 141 – 157, 2013. ISSN 1094-9968. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1094996813000170>. Acesso em: 03/05/2020. Citado na página 22.

MENASCE, D. *et al.* Sassy: A framework for self-architecting service-oriented systems. *IEEE Software*, v. 28, n. 6, p. 78–85, Nov 2011. ISSN 0740-7459. Citado na página 23.

MENDONÇA, T. F. de. *Um Middleware para Coreografias de Serviços Web Escaláveis em Ambientes de Computação em Nuvem*. Dissertação (Mestrado) — Universidade de São Paulo – USP, Instituto de Matemática e Estatística – IME, São Paulo, SP, 7 2015. Citado na página 85.

MULLIGAN, G.; GRACANIN, D. A comparison of soap and rest implementations of a service based interaction independence middleware framework. In: *Proceedings of the 2009 Winter Simulation Conference (WSC)*. [S.l.: s.n.], 2009. p. 1423–1432. ISSN 0891-7736. Citado na página 36.

MUÑOZ-FERNÁNDEZ, J. C. *et al.* 10 challenges for the specification of self-adaptive software. In: *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. [S.l.: s.n.], 2018. p. 1–12. ISSN 2151-1357. Citado 2 vezes na(s) página(s) 51 e 52.

MWENDI, E. Software frameworks, architectural and design patterns. *Journal of Software Engineering and Applications*, v. 07, p. 670–678, 01 2014. Citado na página 24.

MYLLÄRNIEMI, V. *et al.* Development as a journey: factors supporting the adoption and use of software frameworks. *Journal of Software Engineering Research and Development*, v. 6, 12 2018. Citado na página 24.

NARENDRA, N. C.; ORRIENS, B. Requirements-driven modeling of the web service execution and adaptation lifecycle. In: ____. *Distributed Computing and Internet Technology: Third International Conference, ICDCIT 2006, Bhubaneswar, India, December 20-23, 2006. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 314–324. ISBN 978-3-540-68380-3. Disponível em: <https://doi.org/10.1007/11951957_28>. Acesso em: 03/05/2020. Citado na página 32.

NARENDRA, N. C.; ORRIENS, B. The notion of self-aware computing. In: ____. *Self-Aware Computing Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017. p. 314–324. ISBN 978-3-540-68380-3. Disponível em: <https://doi.org/10.1007/11951957_28>. Acesso em: 03/05/2020. Citado na página 49.

NETO, B. F. dos S. *JAAF: Implementando Agentes Auto-Adaptativos Orientados a Serviços*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, Departamento de Informática, 3 2010. Citado na página 84.

OASIS. *Reference Architecture Foundation for Service Oriented Architecture*. 2012. On-line. OASIS Committee Specification 01. Disponível em: <https://goo.gl/m2pEm4>. Acesso em: 03/05/2020. Citado 2 vezes na(s) página(s) 34 e 35.

OLIVEIRA, P. A. de. *Seleção de serviços web em composições coreografadas*. Dissertação (Mestrado) — Universidade de São Paulo – USP, Instituto de Matemática e Estatística – IME, São Paulo, SP, 6 2014. Citado na página 85.

OQUENDO, F. Dynamic software architectures: Formally modelling structure and behaviour with pi-adl. In: *ICSEA' 2008*. [S.l.: s.n.], 2008. p. 352–359. Citado na página 47.

ORACLE. *What Are Web Services? - Java EE Documentation*. 2017. On-line. Disponível em: <http://docs.oracle.com/javaee/7/tutorial/webservices-intro001.htm>. Acesso em: 03/05/2020. Citado na página 29.

PARK, S.; SONG, J. Self-adaptive middleware framework for internet of things. In: *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*. [S.l.: s.n.], 2015. p. 81–82. Citado na página 25.

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. Citado 2 vezes na(s) página(s) 87 e 90.

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*. [S.l.: s.n.], 2018. p. 286 – 291. Citado 2 vezes na(s) página(s) 87 e 90.

PETERSEN, K. *et al.* Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, UK: BCS Learning & Development Ltd., 2008. (EASE'08), p. 68–77. Disponível em: <http://dl.acm.org/citation.cfm?id=2227115.2227123>. Acesso em: 03/05/2020. Citado 4 vezes na(s) página(s) 24, 65, 66 e 133.

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. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584915000646>. Acesso em: 03/05/2020. Citado 5 vezes na(s) página(s) 24, 66, 133, 153 e 155.

PROVENSI, L. L. *Uma Plataforma de Middleware Reflexivo com Suporte para Auto-Adaptação*. Dissertação (Mestrado) — Universidade Federal de Goiás – UFG, Instituto de Informática – INF, Goiânia, GO, 3 2009. Citado na página 84.

PSAIER, H.; DUSTDAR, S. A survey on self-healing systems: approaches and systems. *Computing*, v. 91, n. 1, p. 43–73, Jan 2011. ISSN 1436-5057. Citado na página 88.

PSAIER, H.; DUSTDAR, S. A survey on self-healing systems: Approaches and systems. *Computing*, v. 91, p. 43–73, 08 2011. Citado na página 89.

PSAIER, H. *et al.* Behavior monitoring in self-healing service-oriented systems. In: *2010 IEEE 34th Annual Computer Software and Applications Conference*. [S.l.: s.n.], 2010. p. 357–366. ISSN 0730-3157. Citado na página 23.

RICHARDS, M. *Microservices vs. Service-Oriented Architecture*. [S.l.]: O'Reilly Media, 2016. ISBN 978-1-491-94161-4. Citado na página 33.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594. Citado na página 49.

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>. Acesso em: 03/05/2020. Citado 10 vezes na(s) página(s) 24, 41, 43, 47, 49, 50, 51, 83, 88 e 90.

SEGARAN, T.; EVANS, C.; TAYLOR, J. *Programming the Semantic Web*. [S.l.]: Manning Publications Co., 2009. ISBN 978-0-5961-5381-6. Citado 3 vezes na(s) página(s) 55, 56 e 58.

SHAFIUZZAMAN, M.; NAHAR, N.; RAHMAN, M. R. A proactive approach for context-aware self-adaptive mobile applications to ensure quality of service. In: *2015 18th International Conference on Computer and Information Technology (ICCIT)*. [S.l.: s.n.], 2015. p. 544–549. Citado na página 23.

SIKOS, L. F. *Mastering Structured Data on the Semantic Web: From HTML5 Microdata to Linked Open Data*. [S.l.]: Manning Publications Co., 2015. ISBN 978-1-4842-1049-9. Citado na página 56.

SMARTBEAR. *Understanding SOAP and REST Basics and Differences*. 2013. On-line. Disponível em: <https://smartbear.com/blog/test-and-monitor/understanding-soap-and-rest-basics>. Acesso em: 03/05/2020. Citado na página 37.

SOAPUI. *SOAP vs REST 101: Understand The Differences*. 2018. On-line. Disponível em: <https://www.soapui.org/learn/api/soap-vs-rest-api.html>. Acesso em: 03/05/2020. Citado na página 36.

STEPHAN, K. D. *et al.* Social implications of technology: The past, the present, and the future. *Proceedings of the IEEE*, v. 100, n. Special Centennial Issue, p. 1752–1781, May 2012. ISSN 1558-2256. Citado na página 21.

TAO, M. *et al.* Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes. *Future Generation Computer Systems*, v. 78, p. 1040 – 1051, 2018. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X16305775>. Acesso em: 03/05/2020. Citado na página 23.

TREIBER, M. *et al.* Programming evolvable web services. In: *Proceedings of the 2Nd International Workshop on Principles of Engineering Service-Oriented Systems*. New York, NY, USA: ACM, 2010. (PESOS '10), p. 43–49. ISBN 978-1-60558-963-3. Disponível em: <http://doi.acm.org/10.1145/1808885.1808895>. Citado na página 25.

TRENDS, G. *SOAP x REST: interesse ao longo do tempo*. 2020. On-line. Google Trends. Disponível em: <https://trends.google.com.br/trends/explore?date=all&q=%2Fm%2F077dn,%2Fm%2F03nsxd>. Acesso em: 03/05/2020. Citado na página 36.

TRINUGROTO, Y. B. D.; REICHERT, F.; FENSLI, R. W. A soa-based health service platform in smart home environment. In: *2011 IEEE 13th International Conference on e-Health Networking, Applications and Services*. [S.l.: s.n.], 2011. p. 201–204. Citado na página 23.

VINOSKI, S. A time for reflection [software reflection]. *Internet Computing, IEEE*, v. 9, n. 1, p. 86 – 89, jan.-feb. 2005. ISSN 1089-7801. Citado na página 41.

W3C. *Web service usage scenario: Travel reservation*. 2002. On-line. Disponível em: <https://www.w3.org/2002/04/17-ws-usecase>. Acesso em: 03/05/2020. Citado na página 33.

W3C. *OWL-S: Semantic Markup for Web Services*. 2004. On-line. W3C Member Submission 22 November 2004. Disponível em: <https://www.w3.org/Submission/OWL-S/>. Acesso em: 03/05/2020. Citado 3 vezes na(s) página(s) 61, 62 e 64.

W3C. *Web Services Architecture*. 2004. On-line. Introduction, Purpose of the Web Service Architecture. Disponível em: <https://www.w3.org/TR/ws-arch/>. Acesso em: 03/05/2020. Citado 2 vezes na(s) página(s) 29 e 31.

W3C. *RDF 1.1 Concepts and Abstract Syntax*. 2014. On-line. W3C Recommendation 25 February 2014. Disponível em: <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>. Acesso em: 03/05/2020. Citado na página 58.

W3C. *RDF 1.1 Turtle*. 2014. On-line. Terse RDF Triple Language. W3C Recommendation 25 February 2014. Disponível em: <https://www.w3.org/TR/turtle/>. Acesso em: 03/05/2020. Citado na página 58.

WEISER, M. The computer for the 21 st century. *Scientific American*, Scientific American, a division of Nature America, Inc., v. 265, n. 3, p. 94–105, 1991. Citado na página vii.

WEYNS, D. Engineering self-adaptive software systems – an organized tour. In: . [S.l.: s.n.], 2018. p. 1–2. Citado 2 vezes na(s) página(s) 43 e 44.

WEYNS, D. *et al.* Claims and supporting evidence for self-adaptive systems: A literature study. p. 89–98, 06 2012. Citado na página 51.

WILLIAMS, B. J.; CARVER, J. C. Characterizing software architecture changes: A systematic review. *Information and Software Technology*, v. 52, n. 1, p. 31 – 51, 2010. ISSN 0950-5849. Citado na página 41.

WOHLIN, C. *et al. Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN 3642290434. Disponível em: <https://www.springer.com/gp/book/9783642290435>. Acesso em: 05/06/2020. Citado na página 117.

WU, Z.; DENG, S.; WU, J. Chapter 2 - service-oriented architecture and web services. In: WU, Z.; DENG, S.; WU, J. (Ed.). *Service Computing*. Boston: Academic Press, 2015. p. 17 – 42. ISBN 978-0-12-802330-3. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780128023303000023>. Acesso em: 03/05/2020. Citado na página 61.

WU, Z.; DENG, S.; WU, J. Chapter 4 - service discovery. In: WU, Z.; DENG, S.; WU, J. (Ed.). *Service Computing*. Boston: Academic Press, 2015. p. 79 – 104. ISBN 978-0-12-802330-3. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780128023303000047>. Acesso em: 03/05/2020. Citado na página 61.

YE, D. *et al.* Detection of transmissible service failure in distributed service-based systems. *Journal of Parallel and Distributed Computing*, v. 119, p. 36 – 49, 2018. ISSN 0743-7315. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0743731518301783>. Acesso em: 03/05/2020. Citado na página 136.

ZELKOWITZ, M. V.; WALLACE, D. R. Experimental models for validating technology. *Computer*, v. 31, n. 5, p. 23–31, May 1998. ISSN 1558-0814. Citado na página 117.

ZHANG, L.; ZHANG, J.; CAI, H. *Services Computing*. [S.l.]: Springer Berlin Heidelberg, 2008. ISBN 9783540382843. Citado na página 33.

ZHANG, Y.; CHEN, J.; CHENG, B. Integrating events into soa for iot services. *IEEE Communications Magazine*, v. 55, n. 9, p. 180–186, Sep. 2017. ISSN 1558-1896. Citado na página 23.

Apêndices

APÊNDICE A – Protocolo de pesquisa

Este documento apresenta o protocolo de pesquisa que será utilizado em nosso mapeamento. Inicialmente, a Seção A.1 apresenta um contexto sobre o tema de nosso projeto e as questões de pesquisa propostas. A Seção A.2 mostra a estratégia de pesquisa utilizada, ou seja, nossa *string* de busca, as bases de pesquisa utilizadas e os critérios de inclusão e exclusão utilizados para selecionar os estudos. Em seguida, a Seção A.3 mostra como os dados foram coletados e sintetizados. A Seção A.4 mostra as possíveis limitações da pesquisa. Depois, na Seção A.5, são apresentados como os resultados obtidos devem ser reportados e, por fim, a Seção A.6 mostra um cronograma com a lista de tarefas que devem ser realizadas durante o mapeamento e uma estimativa de tempo para sua conclusão.

A.1 Questões de pesquisa

O objetivo principal deste mapeamento é identificar estudos primários que tenham lidado com *frameworks* para *Self-Apps*. Sendo assim, esse mapeamento tem por objetivo estabelecer um panorama de quais são as características desses *frameworks*. Para isso, o seguinte conjunto de Questões de Pesquisa (QP) foi elaborado:

QP1: Quais são as características demográficas de estudos sobre *frameworks* para *Self-Apps*?
O objetivo dessa questão é determinar a origem de publicações nessa área de pesquisa. A partir dessa questão, quatro subquestões de pesquisa foram elaboradas, a saber:

QP1.1: *Quando os estudos foram publicados?*

QP1.2: *Onde os estudos foram publicados?*

QP1.3: *Como as publicações se distribuem entre a academia e a indústria?*

QP1.4: *Como as publicações se distribuem geograficamente?*

QP2: Quais domínios de aplicação se beneficiam da existência de *frameworks* para *Self-Apps*?
O objetivo dessa questão é identificar os domínios de aplicação que propuseram ou utilizaram frameworks para Self-Apps.

QP3: Quais são os atributos de qualidade de serviço propostos para *Self-Apps*?
O objetivo dessa questão é entender quais são os atributos de qualidade (por exemplo, segurança, performance, confiabilidade) utilizados no desenvolvimento de frameworks

para Self-Apps. A partir das respostas para essa questão, arquiteturas de software, frameworks e arquiteturas de referência podem ser analisadas ou desenvolvidas com base nos atributos adotados por frameworks já existentes.

QP4: Quais são as evidências que motivaram a adoção dos *frameworks* existentes?
O objetivo dessa questão é sintetizar evidências de maturidade e processos de validação utilizados por esses frameworks. A partir dessa síntese é possível obter um panorama da adoção desses frameworks por profissionais da área e pesquisadores.

QP5: Quais estratégias de busca de serviços estão sendo aplicadas em *frameworks* para *Self-Apps*?
O objetivo dessa questão é sintetizar evidências de técnicas, estratégias, abordagens, entre outras, adotadas nesses frameworks para a seleção de serviços por similaridade.

QP6: Quais são as estratégias de gerenciamento que estão sendo utilizadas por *frameworks* para *Self-Apps*?
O objetivo dessa questão é sintetizar evidências de adaptação e evolução de serviços adotadas por esses frameworks.

QP7: Como *Self-Apps* estão sendo projetados?
O objetivo dessa questão é entender qual(is) técnica(s) está(ão) sendo utilizada(s) para projetar Self-Apps tanto em fase de projeto quanto de execução.

QP8: Quais propriedades Self-* estão sendo utilizadas por *frameworks* para *Self-Apps*?
O objetivo dessa questão é identificar quais propriedades Self- foram utilizadas no desenvolvimento de frameworks para Self-Apps através dos anos. Adicionalmente, é de interesse identificar se tais propriedades foram utilizadas de maneira combinada ou independente.*

A.2 Estratégia de pesquisa

A estratégia de pesquisa, composta por critério de seleção das bases de pesquisa, lista de bases pesquisadas, idioma do estudo e, palavras-chave e seus termos relacionados foi definida com base nas questões de pesquisa apresentadas na Seção A.1. A seguir, detalhes da estratégia de pesquisa utilizada são apresentados:

1. **Critério de seleção das bases de pesquisa:** serão adotados os seguintes critérios para selecionar as bases de pesquisa utilizadas neste 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 nosso mapeamento são exibidas na Tabela 8. Segundo Dyba, Dingsoyr e Hanssen (2007), Kitchenham *et al.* (2010) 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.

Tabela 8 – 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>
Springer Link	<link.springer.com>
Web of Science	<www.isiknowledge.com>

3. **Idioma dos estudos:** Por tratar-se do idioma mais comum em publicações científicas e para auxiliar em futuras reproduções desse trabalho, apenas estudos primários escritos em inglês serão considerados;

4. **Palavras-chave:** "Framework", "Web Service" e "Self-*", com os seguintes sinônimos:

   a) ***Framework:*** *"Framework"* e *"Library"*;

   b) ***Web Service:*** *"Cloud-Computing", "Composition", "IoT", "Orchestration", "Service", "Services"* e *"Workflow"*;

   c) ***Self-*:*** *"Self-adaptive", "Self-configurable", "Self-configured", "Self-configuring", "Self-healing", "Self-optimizing", "Self-organizing", "Self-recovery", "Self-repair"* e *"Self-tuning"*;

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 mostrada na Tabela 9.

Outro elemento importante para o planejamento e execução de um mapeamento é definir quais Critérios de Inclusão (CI) e Critérios de Exclusão (EC) serão adotados. A partir da definição desses critérios é possível incluir estudos primários relevantes para responder as questões de pesquisa e excluir estudos que não possuem respostas para as mesmas. Sendo assim, o seguinte conjunto de critérios de inclusão foi definido:

- **CI1:** O estudo primário propõe um *framework* para *Self-Apps*;

Tabela 9 – *String* de pesquisa

({framework} OR {library})
AND
({cloud computing} OR {composition} OR {iot} OR {orchestration} OR {service} OR {services} OR {workflow})
AND
({self-adaptive} OR {self-configurable} OR {self-configured} OR {self-configuring} OR {self-healing} OR {self-optimizing} OR {self-organizing} OR {self-recovery} OR {self-repair} OR {self-tuning})

- **CI2:** O estudo primário reporta um *framework* para *Self-Apps*;
- **CI3:** O estudo primário avalia um *framework* para *Self-Apps*;
- **CI4:** O estudo primário discute um *framework* para *Self-Apps*.

Os critérios de exclusão definidos para o mapeamento foram:

- **EC1:** O estudo não evidencia *frameworks* para *Self-Apps*;
- **EC2:** O estudo é um editorial, artigo de opinião, palestra, opinião, tutorial, poster ou painel;
- **EC3:** O estudo foi escrito em um idioma diferente do definido pela pesquisa;
- **EC4:** O estudo possui apenas um resumo ou não está disponível para ser lido na íntegra;
- **EC5:** 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.
- **EC6:** O estudo é um trabalho secundário.

Nenhuma medida de aceitação será aplicada ao nosso mapeamento devido ao pequeno número de pesquisadores envolvidos. Para cada base de dados mencionada na Seção A.2 será documentado o número de estudos retornado. Para isso, nossa *string* de busca será customizada e executada em cada uma das bases de pesquisa explorando os seguintes campos de busca: título, resumo e palavras-chave. Adicionalmente, serão registrados o número de estudos resultantes para cada meio de publicação 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[1], o título e o resumo de cada

[1] Estudos que restaram após a validação da *string* de busca com os campos de busca determinados e idioma.

estudo será lido e avaliado conforme os critérios de inclusão e exclusão definidos nesta seção. Quando necessário, a introdução e a conclusão também serão lidas.

A.3 Extração e síntese de dados

A fase de coleta de dados resultará em um conjunto de tópicos de pesquisa que descrevem cada estudo primário. Para isso, Petersen, Vakkalanka e Kuzniarz (2015) propuseram o preenchimento de um formulário para cada estudo envolvido no mapeamento com o intuito de auxiliar nas tarefas de extração e síntese de dados. Com base nos campos propostos pelos autores, um formulário específico foi elaborado para esse mapeamento, como mostra a Tabela 10. Alguns dos tópicos de pesquisa desse formulário serão utilizados posteriormente para responder às questões de pesquisa estabelecidas.

Tabela 10 – 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	—
Ano	Ano de publicação	RQ1.1
Base de pesquisa	Nome da base de pesquisa que recuperou o artigo	RQ1.2
Natureza do *framework*	Industrial, acadêmica ou mista	RQ1.3
Países dos autores	País de cada um dos autores	RQ1.4
Veículo de publicação	Nome do veículo de publicação do artigo	—
Domínio de aplicação	Domínio em que o *framework* foi aplicado	QP1
		QP4
Atributos de qualidade	Atributos de qualidade listados	QP2
		QP5
Estratégias de gerenciamento de serviço	Lista de estratégias adotadas	QP5
		QP6
		QP7
Propriedades *Self-**	Lista de propriedades Self-* utilizadas	QP8

A.4 Limitações

A seguir são descritas as possíveis limitações que um trabalho dessa natureza pode apresentar:

1. **Omissão de artigos.** Nosso mapeamento será realizado através de um processo de pesquisa automatizado (ou seja, a *string* de busca será processada pelo mecanismo de busca/pesquisa de cada base). Serão utilizadas seis bases de pesquisa para a obtenção das publicações avaliadas pelo nosso mapeamento e, apesar dos esforços em tornar este trabalho o mais confiável possível, não é possível afirmar que todos os estudos primários relacionados a essa área poderão ser recuperados. Entretanto, com o intuito de calibrar a *string* final utilizada, deverão ser realizadas pesquisas com diferentes variações. Em seguida, a *string* final será posteriormente adaptada para os padrões de cada uma das bases de pesquisa. Portanto, acreditamos 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, acreditamos que os resultados apresentados 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.

4. **Extração de dados.** Outra possível limitação deste mapeamento é com relação a extração de dados. As informações utilizadas para responder as questões de pesquisa 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, foi adotado um formulário de pesquisa com o objetivo de auxiliar na tarefa de coleta e síntese dos dados, produzindo um arquivo padronizado contendo os dados obtidos.

5. **Avaliação da qualidade.** Como nosso objetivo com esse mapeamento é identificar os *frameworks* para *Self-Apps* existentes, a qualidade dos estudos não será avaliada. Entretanto, reconhecemos 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 dessa pesquisa.

Por fim, vale mencionar que um único pesquisador conduzirá as etapas de extração e síntese de dados. Embora um conjunto de diretrizes tenha sido definido com a finalidade de garantir que o mapeamento fosse imparcial, não é possível garantir que toda a informação coletada seja completamente imparcial. Para o futuro esperamos possuir ao menos duas pessoas envolvidas nessas etapas.

A.5 Relatório

Ao reportar os resultados deste mapeamento, será: (i) realizada uma discussão sobre os resultados obtidos (ou seja, descrição de cada estudo primário e detalhes de qualquer meta-análise que tenha sido realizada); (ii) apresentada uma discussão das principais descobertas, forças e fraquezas do mapeamento e os significados dessas descobertas como, por exemplo, aplicabilidade, benefícios, efeitos adversos e riscos; (iii) apresentada uma 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 (iv) elaborada uma discussão sobre as limitações do mapeamento.

A.6 Cronograma

A Tabela 11 mostra uma lista de tarefas a serem realizadas durante o mapeamento e suas respectivas estimativas de conclusão.

Tabela 11 – Lista de tarefas a serem realizadas pelo mapeamento

#	Tarefa	Estimativa
1	Desenvolver o protocolo de pesquisa do mapeamento sistemático	Semana 1
2	Executar a consulta em todas as bases de pesquisa envolvidas	Semana 4
3	Selecionar estudos com base nos critérios de inclusão e exclusão	Semanas 4-6
4	Extrair e sintetizar dados dos estudos selecionados	Semanas 7-10
5	Escrever o relatório final	Semanas 11-14

APÊNDICE B – Lista de estudos selecionados pelo mapeamento sistemático

A Tabela 12 apresenta os estudos que compõem os dados reportados na Seção 5.1 (Capítulo 5), obtidos através da condução do mapeamento sistemático.

Tabela 12 – Lista final de estudos selecionados para a extração e síntese de dados

#	Título	Autores	Ano
S1	A Cloud-Oriented Services Self-Management Approach Based on Multi-agent System Technique	Hou, Fu and Mao, Xinjun and Wu, Wei and Liu, Lu and Panneerselvam, John	2014
S2	A framework for context-aware self-adaptive mobile applications SPL	Mizouni, R. and Matar, M.A. and Mahmoud, Z.A. and Alzahmi, S. and Salah, A.	2014
S3	A Framework for Distributed Management of Dynamic Self-adaptation in Heterogeneous Environments	M. Zouari and M. Segarra and F. André	2010
S4	A framework for monitoring and runtime recovery of Web service-based applications	Pegoraro, R. and Halima, R.B. and Drira, K. and Guennoun, K. and Rosário, J.M.	2008
S5	A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing	Hielscher, Julia and Kazhamiakin, Raman and Metzger, Andreas and Pistore, Marco	2008
S6	A Framework for Self-Healing Service Compositions in Cloud Computing Environments	W. Li and P. Zhang and Z. Yang	2012
S7	A framework to improve adaptability in web service composition	W. Lian and Y. Liang and Q. Zeng and Jingjing Yan	2010
S8	A JINI framework for distributed service flexibility	D. Cotroneo and C. Di Flora and S. Russo	2002
S9	A large-scale monitoring and measurement campaign for Web Services-based applications	Ben Halima, R. and Fki, E. and Drira, K. and Jmaiel, M.	2010
S10	A proactive approach for context-aware self-adaptive mobile applications to ensure Quality of Service	M. Shafiuzzaman and N. Nahar and M. R. Rahman	2015
S11	A QoS-based Self-adaptive Framework for OpenAPI	L. Sun and S. Chen and Q. Liang and Z. Feng	2011

Continua na próxima página

Continuando da página anterior

#	Título	Autores	Ano
S12	A QoS-based Service Composition for Content Adaptation	K. El-Khatib and G. v. Bochmann and A. El-Saddik	2007
S13	A Scalable Approach to QoS-Aware Self-adaption in Service-Oriented Architectures	Cardellini, Valeria and Casalicchio, Emiliano and Grassi, Vincenzo and Lo Presti, Francesco and Mirandola, Raffaela	2009
S14	A Self-healing Framework for QoS-Aware Web Service Composition via Case-Based Reasoning	Li, Guoqiang and Liao, Lejian and Song, Dandan and Wang, Jingang and Sun, Fuzhen and Liang, Guangcheng	2013
S15	A Self-Healing Framework for Web Services	H. Naccache and G. C. Gannod	2007
S16	A Self-healing Web Server Using Differentiated Services	Naccache, Henri and Gannod, Gerald C. and Gary, Kevin A.	2006
S17	A self-optimizing QoS-aware service composition approach in a context sensitive environment	Shen, Yuan-hong and Yang, Xiao-hu	2011
S18	A self-organizing P2P framework for collective service discovery	Carlo Mastroianni and Giuseppe Papuzzo	2014
S19	A semantic framework for self-adaptive networked appliances	P. Fergus and M. Merabti and M. B. Hanneghan and A. Taleb-Bendiab and A. Mingkhwan	2005
S20	A Unified Framework for Resource Discovery and QoS-aware Provider Selection in Ad Hoc Networks	Liu, Jiangchuan and Zhang, Qian and Li, Bo and Zhu, Wenwu and Zhang, Jun	2002
S21	ACCADA: A Framework for Continuous Context-Aware Deployment and Adaptation	Gui, Ning and De Florio, Vincenzo and Sun, Hong and Blondia, Chris	2009
S22	AFAWS: An Agent based Framework for Autonomic Web Services	Chainbi, W. and Mezni, H. and Ghedira, K.	2012
S23	Ajax API Self-adaptive Framework for End-to-end User	Li, Xiang and Feng, Zhiyong and Huang, Keman and Chen, Shizhan	2015
S24	An approach to constructing high-available decentralized systems via self-adaptive components	Sun, X. and Zhou, L. and Zhuang, L. and Jiao, W.P. and Mei, H.	2009
S25	Automated Publish, Discovery and Composition of Intentional Web Services Adaptable to Both Quality and Context	E. Khanfir and R. B. Djmeaa and I. Amous	2016
S26	Automated self-healing framework for service-oriented systems	A. Alhosban and E. Najmi and K. Hashmi and Z. Malik	2013
S27	Autonomic Composite-service Architecture with MAWeS	E. P. Mancini and M. Rak and U. Villano	2010
S28	AWS-Ont: An Ontology for the Self-Management of Service-Based Systems	H. Mezni and M. Sellami	2015

Continua na próxima página

Continuando da página anterior

#	Título	Autores	Ano
S29	Behavior Monitoring in Self-Healing Service-Oriented Systems	H. Psaier and F. Skopik and D. Schall and S. Dustdar	2010
S30	Building self-adapting services using service-specific knowledge	An-Cheng Huang and P. Steenkiste	2005
S31	Component & service-based agent systems: Self-OSGi	Dragone, M.	2012
S32	Composing Components and Services Using a Planning-Based Adaptation Middleware	Rouvoy, Romain and Eliassen, Frank and Floch, Jacqueline and Hallsteinsen, Svein and Stav, Erlend	2008
S33	Composition and Self-Adaptation of Service-Based Systems with Feature Models	Cubo, Javier and Gamez, Nadia and Fuentes, Lidia and Pimentel, Ernesto	2013
S34	Context-aware mobile services adaptation to dynamic resources. Application to mHealth	E. Nageba and P. Rubel and J. Fayn	2012
S35	Cost-based prevention of violations of service level agreements in composed services using self-adaptation	P. Leitner and S. Dustdar and B. Wetzstein and F. Leymann	2012
S36	Defining a Self-Healing QoS-based Infrastructure for Web Services Applications	F. Moo-Mena and J. Garcilazo-Ortiz and L. Basto-Díaz and F. Curi-Quintal and F. Alonzo-Canul	2008
S37	Design of an autonomic services oriented architecture	M. A. C. Bhakti and A. B. Abdullah	2010
S38	Designing a Framework for Smart IoT Adaptations	Achtaich, Asmaa and Souissi, Nissrine and Mazo, Raul and Salinesi, Camille and Roudies, Ounsa	2018
S39	Developing adaptive distributed applications: A framework overview and experimental results	Da Silva E Silva, F.J. and Endler, M. and Kon, F.	2003
S40	DISC: A Declarative Framework for Self-Healing Web Services Composition	E. Zahoor and O. Perrin and C. Godart	2010
S41	DISCO: A dynamic self-configuring discovery service for semantic web services	Elgedawy, I.	2017
S42	Dynamic deployment of pervasive services	I. Satoh	2005
S43	Integrating Fault-Tolerance Method into Service-Oriented Computing	A. Alhosban and K. Hashmi and Z. Malik and B. Medjahed	2015
S44	JAAF-S: A framework to implement autonomic agents able to deal with web services	Neto, B.F.D.S. and Da Costa, A.D. and De Lucena, C.J.P. and Da Silva, V.T. and De Netto, M.T.A.	2009
S45	Modeling and Validating Self-adaptive Service-oriented Applications	Arcaini, Paolo and Riccobene, Elvinia and Scandurra, Patrizia	2015

Continua na próxima página

Continuando da página anterior

#	Título	Autores	Ano
S46	MOSES: A Platform for Experimenting with QoS-Driven Self-Adaptation Policies for Service Oriented Systems	Cardellini, Valeria and Casalicchio, Emiliano and Grassi, Vincenzo and Iannucci, Stefano and Lo Presti, Francesco and Mirandola, Raffaela	2017
S47	Nature-inspired multimedia service composition in a media cloud-based healthcare environment	Alamri, Atif	2016
S48	Predictive self-healing of web services using health score	Sharifi, M. and Ramezani, S.B. and Amirlatifi, A.	2012
S49	Programming Evolvable Web Services	Treiber, Martin and Juszczyk, Lukasz and Schall, Daniel and Dustdar, Schahram	2010
S50	QoS-Driven Self-adaptation for Critical IoT-Based Systems	Gatouillat, Arthur and Badr, Youakim and Massot, Bertrand	2018
S51	QoS-Driven Self-Healing Web Service Composition Based on Performance Prediction	Dai, Yu and Yang, Lei and Zhang, Bin	2009
S52	RADAR: Self-configuring and self-healing in resource management for enhancing quality of cloud services	Gill, S.S. and Chana, I. and Singh, M. and Buyya, R.	2019
S53	Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems	H. Psaier and L. Juszczyk and F. Skopik and D. Schall and S. Dustdar	2010
S54	Self-Adaptation of Service Based Systems Based on Cost/Quality Attributes Tradeoffs	R. Mirandola and P. Potena	2010
S55	Self-Adaptive Goal-Driven Web Service Composition Based on Context and QoS	E. Khanfir and R. B. Djmeaa and I. Amous	2017
S56	SHõWA: A Self-healing framework for web-based applications	Magalhães, J. P. and Silva, L.M.	2015
S57	Service Discovery and Provision for Autonomic Mobile Computing	Polyzos, George C. and Ververidis, Christopher N. and Efstathiou, Elias C.	2006
S58	Towards a Unified Framework for the Monitoring and Recovery of BPEL Processes	Baresi, Luciano and Guinea, Sam and Pasquale, Liliana	2008
S59	Using Service Clustering and Self-Adaptive MOPSO-CD for QoS-Aware Cloud Service Selection	Giandomenico Spezzano	2016
S60	An adaptive and scalable framework for automated service discovery	Sikri, M.	2016
S61	A Generic Context-Aware Service Discovery Architecture for IoT Services	Sasirekha, S. and Swamynathan, S. and Keerthana, S.	2018

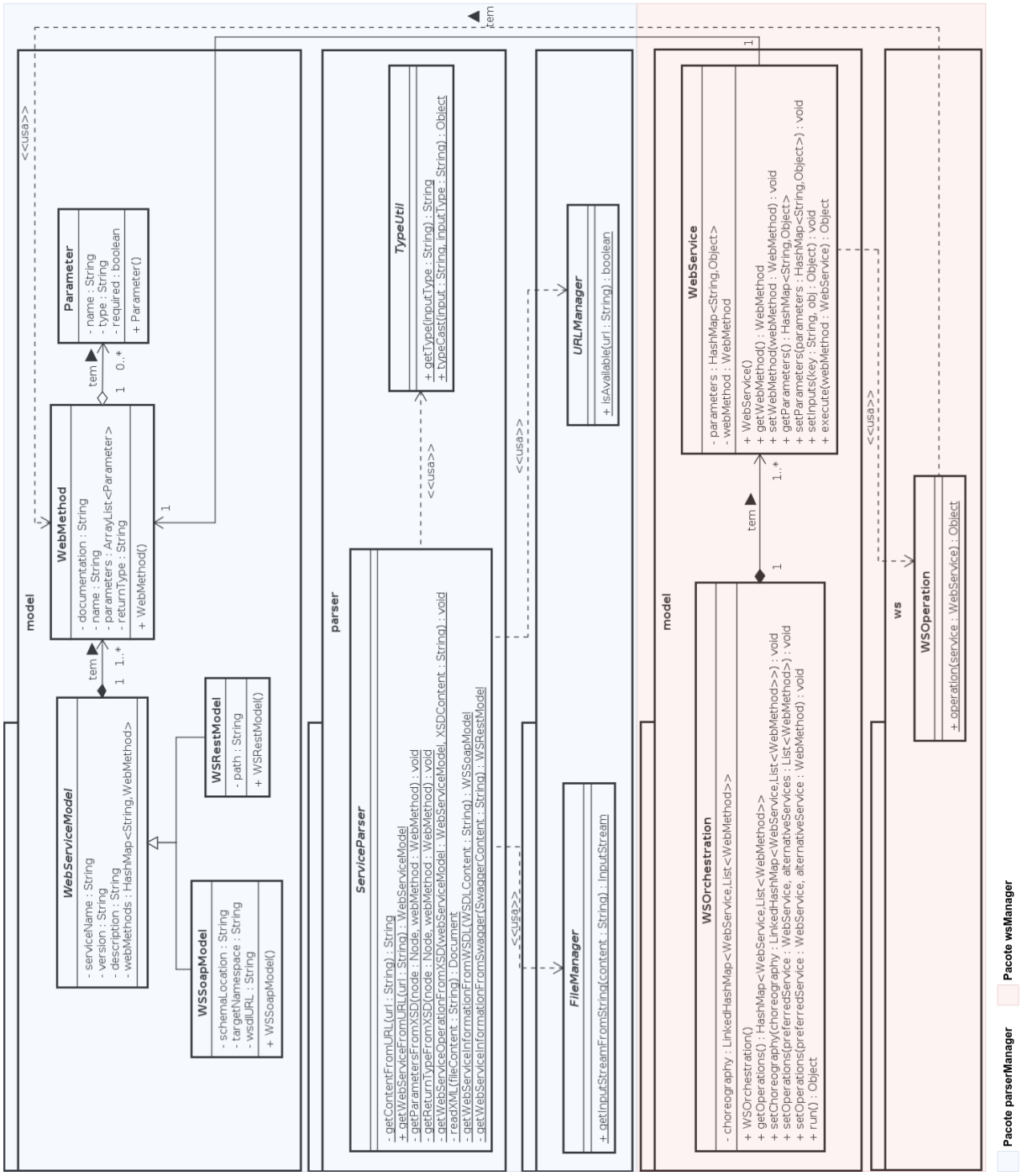
Continua na próxima página

Continuando da página anterior

#	Título	Autores	Ano
S62	A QoS-supported approach using fault detection and tolerance for achieving reliability in dynamic orchestration of web services	Gupta, Reena and Kamal, Raj and Suman, Ugrasen	2018
S63	QoS provisioning framework for service-oriented internet of things (IoT)	Badawy, Mahmoud M. and Ali, Zainab H. and Ali, Hesham A.	2019
S64	QoS Uncertainty Handling for an Efficient Web Service Selection	Abdelhak, Etchiali and Feth-Allah, Hadjila and Mohammed, Merzoug	2019
S65	Parallel Adaptation of Multiple Service Composition Instances	Aschoff, Rafael Roque and Zisman, Andrea and Alexandre, Pedro	2019

APÊNDICE C – Diagrama de classes do *framework* DynaMS

Figura 39 – Diagrama de classes completo do *framework* DynaMS



Fonte: Elaborado pelo autor

APÊNDICE D – Documento de Requisitos do sistema *App2Rest*

D.1 VISÃO GERAL DO SISTEMA

O sistema para restaurantes (*App2Rest*) consiste em gerenciar os serviços providos por um restaurante e os recursos relacionados a administração do estabelecimento, como, por exemplo, controle e monitoramento de estoque. O sistema é composto por uma parte *online*, responsável por controlar ações como a reserva de lugares (com a opção de realizar o pedido adiantado), a qual está interligada com os demais dispositivos do restaurante, tais quais *tablets* integrados nas mesas e telas nas cozinhas responsáveis por informar os pedidos.

D.2 REQUISITOS FUNCIONAIS

D.2.1 Lançamentos diversos

1. O sistema deve permitir a inclusão, alteração e remoção de clientes do restaurante, com os seguintes atributos: nome, endereço (rua, número, bairro, cep, cidade, estado), telefone residencial, telefone celular, e-mail, CPF e data de nascimento.

2. O sistema deve permitir a inclusão, alteração e remoção de produtos no estoque do restaurante com os seguintes atributos: nome e preço.

3. O sistema deve permitir a inclusão, alteração e remoção de atendimentos do cliente, com os seguintes atributos: data solicitada, data de solicitação, pedido, número de pessoas e tipo de atendimento (imediato ou reserva).

4. O sistema deve armazenar o histórico de pedidos dos clientes cadastrados.

5. O sistema deve permitir a inclusão, alteração e remoção dos tipos de serviços adicionais oferecidos pelo restaurante. Cada tipo de serviço adicional possui os seguintes atributos: código do tipo de serviço oferecido, descrição do serviço e preço.

6. O sistema deve permitir o processamento da reserva de mesas, com os seguintes atributos: data solicitada, data de solicitação, pedido, número de pessoas, tipo, número da mesa e status.

7. O sistema deve gerenciar o estado de disponibilidade das mesas, através dos seguintes atributos: número da mesa e status.

8. O sistema deve exibir aos funcionários que estão na cozinha os pedidos conforme eles são realizados pelas mesas, contendo os itens dos pedidos e a identificação de qual mesa eles pertencem.

9. O sistema deve permitir o envio de uma mensagem eletrônica para os aniversariantes do mês, contendo chamadas promocionais.

10. O sistema deve permitir o envio de uma mensagem eletrônica para os clientes inativos, contendo chamadas promocionais.

D.2.2 Impressão de diversos tipos de relatórios e consultas

11. O sistema deve permitir a impressão de uma listagem dos pedidos em aberto.

12. O sistema deve permitir a impressão de uma listagem das reservas efetuadas para a data atual, contendo o nome do cliente e telefone para contato.

13. O sistema deve permitir a impressão de um comprovante de pagamento do pedido realizado.

14. O sistema deve permitir a impressão de um relatório contendo as faturas em atraso no período (por exemplo, semanal ou quinzenal), contendo, para cada dia do período, o nome do cliente, a data de vencimento e o valor devido pelo cliente.

15. O sistema deve permitir a impressão de um relatório contendo os aniversariantes do mês.

16. O sistema deve permitir a impressão de um relatório contendo os clientes inativos, ou seja, aqueles que não realizaram pedidos ao restaurante por um prazo superior a 365 dias.

D.3 REQUISITOS NÃO FUNCIONAIS

D.3.1 Confiabilidade

17. O sistema deve ter capacidade para recuperar os dados perdidos da última operação que realizou em caso de falha.

18. O sistema deve fornecer facilidades para a realização de backups dos arquivos do sistema.

19. O sistema deve possuir senhas de acesso e identificação para diferentes tipos de usuários: administrador do sistema, funcionários do restaurante e clientes.

20. O sistema deve permitir a recuperação da senha de acesso do cliente caso ele tenha se esquecido.

21. O sistema deve permitir o acompanhamento do estado do pedido.

22. O sistema deve possuir alguma forma de redundância para garantir que o restaurante se mantenha operacional em caso de breves falhas no abastecimento de energia.

D.3.2 Eficiência

23. O sistema deve responder a consultas on-line em menos de 5 segundos.

24. O sistema deve exibir a listagem dos últimos 5 pedidos realizados em tempo real.

25. O sistema deve exibir a listagem de pedidos que devem ser entregues nas respectivas mesas.

D.3.3 Portabilidade

26. O sistema deve ser executado em terminais locais do restaurante (funcionários), em pequenos terminais instalados nas mesas (pagamento e realização de pedidos) e via Web (*tablet*s, *smartphones* ou qualquer computador capaz de executar algum dos navegadores mais recentes).

27. O sistema deve ser capaz de armazenar os dados em uma base de dados.

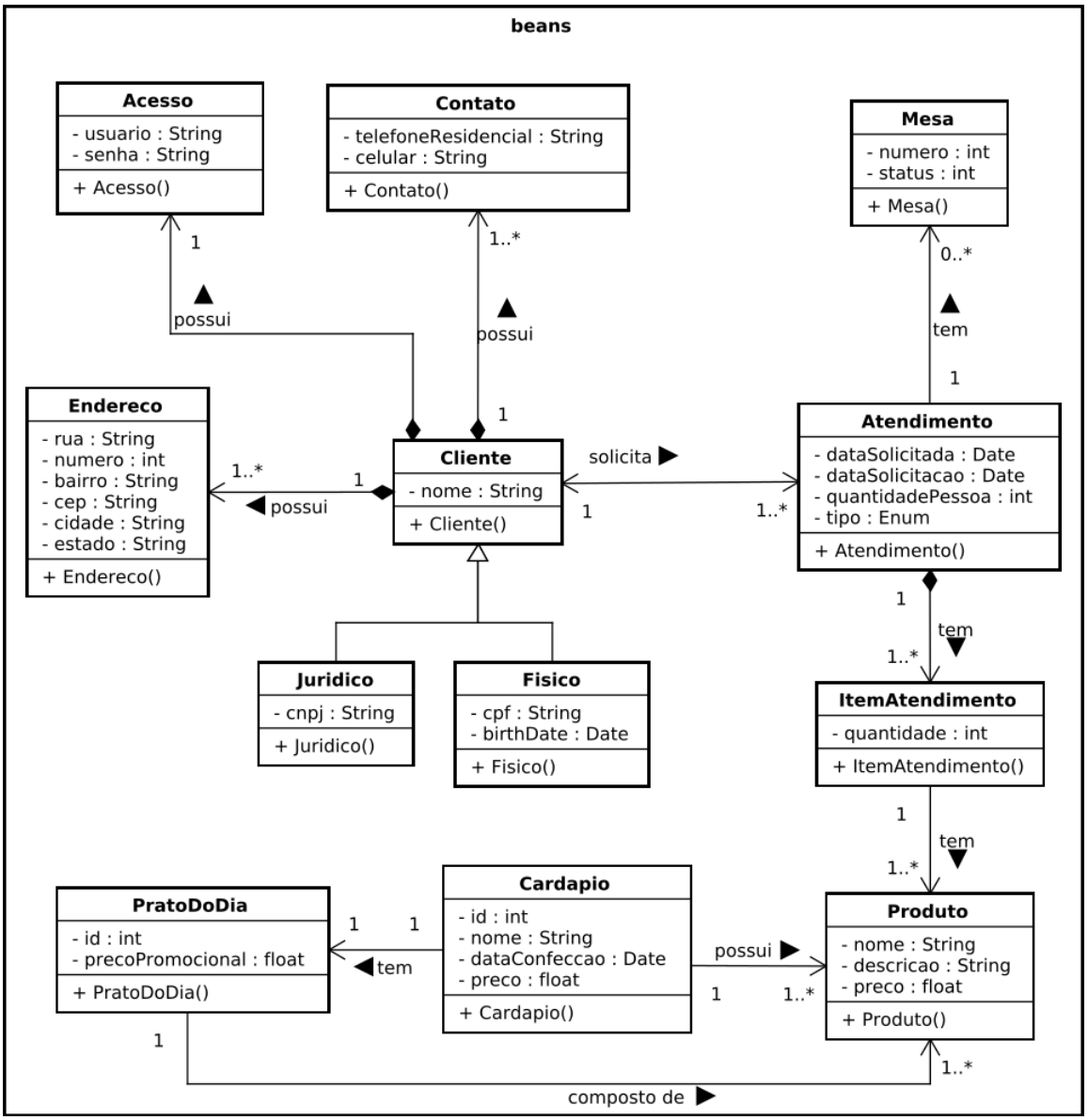
D.4 Diagrama de classes e descrição do sistema

Com base na especificação de requisitos apresentada, foi elaborado um diagrama de classes para definir a estrutura geral e as relações das classes do sistema, como ilustra a Figura 40. Nesse modelo são apresentadas as classes responsáveis por armazenar as informações dos clientes (classes `Cliente, Endereco, Contato e Acesso`), produtos fornecidos pelo restaurante (classe `Produto`), pratos do restaurante que são ofertados aos clientes via cardápio (classes `Cardapio e PratoDoDia`) e atendimento aos clientes do restaurante (classes `Atendimento, Pedido, ItemPedido e Mesa`). Os dados de um atendimento à um cliente (classe `Atendimento`) devem ser sempre vinculados a um cliente e podem ter vários pedidos (classes `Pedido` e `ItemPedido`).

Do ponto de vista operacional, um procedimento foi elaborado para implementar os requisitos apresentado neste apêndice. O cliente, ao chegar no restaurante, senta-se à mesa, cuja luz de controle se tornará vermelha assim que uma ordem de serviço for aberta, indicando que a

mesa está sendo utilizada. Ao selecionar um item do cardápio, o cliente deve informar ao sistema sobre sua solicitação para que o mesmo seja inserido em uma fila simples de atendimento. Na cozinha do restaurante os pedidos são preparados e encaminhados à mesa dos clientes pelos garçons. Durante sua permanência no restaurante o cliente pode realizar diversos pedidos e, ao optar por encerrar sua comanda, o cliente escolha a forma de pagamento. Se a opção cartão for selecionada, o cliente poderá efetuar o pagamento na máquina acoplada em sua mesa. Caso contrário, poderá se dirigir ao caixa ou solicitar a presença de um funcionário do restaurante para recebimento do pagamento. Assim que o pagamento for concretizado, a luz de controle da mesa muda para a cor laranja, indicando que aquela mesa foi utilizada e está aguardando limpeza por um funcionário responsável. A luz de controle torna-se verde novamente após a realização da limpeza, indicando que aquela mesa está pronta para um novo atendimento.

Figura 40 – Diagrama de classes do sistema de restaurante inteligente.



Fonte: Elaborado pelo autor.