UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

SELEÇÃO AUTOMATIZADA DE SERVIÇOS WEB BASEADA EM MÉTRICAS FUNCIONAIS E ESTRUTURAIS

ALYSSON ALVES DE LIMA

João Pessoa - PB
Agosto de 2016

UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

SELEÇÃO AUTOMATIZADA DE SERVIÇOS WEB BASEADA EM MÉTRICAS FUNCIONAIS E ESTRUTURAIS

ALYSSON ALVES DE LIMA

João Pessoa - PB
Agosto de 2016

ALYSSON ALVES DE LIMA

SELEÇÃO AUTOMATIZADA DE SERVIÇOS WEB BASEADA EM MÉTRICAS FUNCIONAIS E ESTRUTURAIS

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como requisito parcial para obtenção do título de Mestre em Informática (Sistemas de Computação).

Orientador: Prof. Dr. Gledson Elias da Silveira

João Pessoa – PB

Agosto de 2016

Dedico este trabalho a todos aqueles que por ventura sabem que são importantes para mim, e estiveram juntos comigo nessa caminhada.

In memory

Severino José de Lima

AGRADECIMENTOS

Primeiramente agradeço a Deus, que me abençoa em todos os momentos e que me deu força, sabedoria, inspiração e paciência para realização deste trabalho.

Um agradecimento mais que especial ao meu orientador Gledson Elias da Silveira, por suas ideias, compreensão, conhecimentos e habilidades que me conduziram e incentivaram tornando possível a realização desse projeto.

Agradeço a toda minha família e amigos, por todo incentivo e apoio.

RESUMO

Engenharia de Software é uma disciplina que engloba todos os aspectos da produção de um sistema de software, incluindo desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo utilizado. Uma área de estudo bastante interessante da Engenharia de Software é o reuso de software, que impacta positivamente na redução do tempo, dos custos e dos riscos provenientes de um processo de desenvolvimento de software. Portanto, é possível afirmar que o reuso de software melhora, não apenas o processo de desenvolvimento de software, mas também o próprio produto. Uma das principais abordagens de reuso de software é o desenvolvimento orientado a serviços, que adota o paradigma da Arquitetura Orientada a Serviços (SOA – *Service-Oriented Architecture*). No paradigma SOA, serviços representam uma evolução natural do desenvolvimento baseado em componentes, e, portanto, podem ser definidos como componentes de software de baixo acoplamento, reusáveis, que encapsulam funcionalidades discretas, que podem ser distribuídos e acessados remotamente de forma programática. É importante destacar que, enquanto SOA é um paradigma arquitetural para desenvolvimento de sistemas de software, serviços web (*web services*) representam a tecnologia existente mais amplamente adotada para implementar SOA explorando protocolos baseados em padrões da internet e em XML (*eXtensible Markup Language*). Com o crescimento do mercado e utilização dos serviços web, a tendência é sempre aumentar o número de serviços disponíveis para montagem de aplicações em diferentes contextos, tornando impraticável a tarefa de selecionar de forma manual os serviços requeridos para compor um sistema de software. Consequentemente, é possível afirmar que o esforço necessário para selecionar os serviços requeridos tende a aumentar cada vez mais, gerando um problema com um grande e complexo espaço de busca, tornando necessária a automatização do processo de seleção baseada em técnicas de busca metaheurística. Neste contexto, o trabalho proposto visa automatizar o processo de seleção de serviços web utilizando técnicas da Engenharia de Software Baseada em Buscas, cuja estratégia de seleção é orientada por métricas funcionais e estruturais, que têm o propósito de avaliar a similaridade entre as especificações e as respectivas implementações dos serviços candidatos, bem como as suas dependências, reduzindo assim o esforço de adaptação e integração de serviços web desenvolvidos por fornecedores distintos.

Palavras-chave: Engenharia de Software, SOA, Serviço Web, Seleção de Serviços, Métricas de Software.

ABSTRACT

Software Engineering is a discipline that encompasses all aspects of the production of a software system, from the early stages of the system specification to maintenance, when the system is already being used. A very interesting area in Software Engineering is software reuse, which impacts positively on reducing time, costs and risks in software development processes. Therefore, it can be stated that software reuse improves not only the software development process, but also the product itself. One of the main approaches for software reuse is service oriented development, which adopts the Service-Oriented Architecture (SOA) paradigm. In SOA, services represent a natural evolution of component-based development, and therefore can be defined as loosely coupled, reusable software components, that encapsulate discrete functionality, can be distributed and remotely accessed through coding. It is important to highlight that while SOA is an architectural paradigm for developing software systems, Web Services represent the most widely existing technology adopted to implement SOA exploring protocols based on Internet standards and on XML. With the growth of the market and the use of web services, the tendency is always increase the number of services available for assembly applications in different contexts, making impractical the task of manually selecting the services required to compose a software system. Consequently, one can state that the effort needed to select the required services tends to increase more and more, creating a problem with a large and complex search space, making it necessary the automation of the selection process based on metaheuristic search techniques. In this context, the proposed work aims to automate the web services selection process using techniques of Search-Based Software Engineering, in which the selection strategy is guided by structural and functional metrics that have the purpose of evaluating the similarity between the specifications and respective implementations of candidate services as well as their dependencies, thus reducing the effort of adaptation and integration of web services developed by different suppliers.

Keywords: Software Engineering, SOA, Web Service, Selection of Services, Software Metrics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
API	Application Programming Interface
BPEL	Business Process Execution Language
CRUD	Create, Read, Update e Delete
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JMS	Java Message Service
JSON	JavaScript Object Notation
MCDC	Multi-Criteria Decision Making
OASIS	Advancing Open Standards for the Information Society
QoS	Quality of Service
REST	Representational State Transfer
RPC	Remote Procedure Call
SBSE	Search-Based Software Engineering
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture

SUMÁRIO

Capítulo 1

Introdução

Os avanços na Engenharia de Software têm contribuído para o aumento da produtividade no processo de desenvolvimento de software, e um campo interessante dessa disciplina é o reuso de software, que impacta positivamente na redução do tempo de desenvolvimento, dos custos e dos riscos. Com isso, é possível afirmar que o reuso de software melhora não apenas o ciclo de desenvolvimento do produto, mas também o próprio produto (Jacobson, 1997). Uma das principais abordagens de reuso de software é a utilização de serviços para compor um sistema de software (Dan, 2008). Os serviços podem ser considerados uma evolução natural dos componentes de software em que o modelo de componentes é, em essência, um conjunto de padrões associados, como por exemplo: os padrões associados com *web services*. Um serviço pode ser definido como um componente de software de baixo acoplamento, reusável, que encapsula funcionalidades, que pode ser distribuído e acessado por meio de programas (Newcomer, 2005). Um serviço web também pode ser definido como uma representação padrão para algum recurso computacional ou de informações que podem ser usadas por outros programas (Lovelock, 1996).

Arquitetura Orientada a Serviço (SOA - *Service-Oriented Architecture*) é uma forma de desenvolvimento de sistemas distribuídos em que os componentes são serviços autônomos, executados em computadores geograficamente distribuídos. Tais sistemas de software podem ser construídos pela busca, seleção e composição de serviços locais e externos, por meio de uma adaptação para realizar a integração entre os serviços no sistema.

Um dos problemas mais desafiadores da utilização de SOA é realizar a seleção de serviços web, onde o responsável pela arquitetura do sistema de software depara-se com o desafio de montar uma configuração arquitetural com uma grande variedade de serviços web disponíveis, que possuem requisitos funcionais e não funcionais equivalentes ou similares as especificações dos serviços que compõem o sistema de software proposto.

Idealmente, serviços web são conectados e integrados uns aos outros com o intuito de gerar novos serviços, onde cada serviço executa suas funcionalidades sem nenhum tipo de dependência. Entretanto, na prática, os serviços web são desenvolvidos por fabricantes distintos e geralmente não se adaptam entre si, necessitando de esforço para adaptação, caso eles sejam

escolhidos para fazerem parte de algum sistema (Becker, 2006). Isso evidencia que a questão de incompatibilidades de integração entre serviços web deve ser tratada na etapa de seleção, pois, caso serviços web muito incompatíveis entre si sejam selecionados, o custo para adaptação será muito alto e impactará negativamente no tempo e custo do desenvolvimento. O tratamento de incompatibilidades de integração deve ser reconhecido como inevitável, representando uma tarefa crucial para a Engenharia de Software Orientada a Serviços (Becker, 2006).

Engenharia de Software Orientada a Serviços, ou simplesmente Engenharia de Serviços, é o processo de desenvolvimento de serviços para reuso em aplicações orientadas a serviço. Os engenheiros de software precisam garantir que o serviço represente uma abstração reusável que poderia ser útil em diferentes sistemas. Geralmente, eles devem projetar e desenvolver uma funcionalidade útil associada com essa abstração e assegurar que o serviço seja robusto e confiável. O serviço deve ser documentado para que possa ser descoberto e compreendido pelos usuários em potencial. Existem três estágios lógicos no processo de engenharia de serviços (Becker, 2006):

- Identificação de serviços candidatos: identificar os possíveis serviços que podem ser implementados e definir os requisitos do serviço.
- Projeto de serviços: projetar a lógica e as interfaces WSDL dos serviços.
- Implementação e implantação de serviços: implementar e testar os serviços tornando-os disponíveis para uso.

Ao se trabalhar com Engenharia de Serviços, é possível se deparar com algumas questões de incompatibilidades na integração entre os serviços, algo que prejudica o projeto, pois impede o sistema de software de alcançar uma configuração arquitetural satisfatória quando comparada com a especificação realizada para o software. A seleção de serviços web tem mostrado ser a etapa de maior complexidade ao se adotar essa abordagem para desenvolvimento de software (Erl 2007).

1.1. Motivação

Mesmo obtendo relatos de sucesso e avanços, a indústria ainda não aderiu massivamente ao reuso de software, e, consequentemente, ao reuso de componentes. Dentre as limitações apresentadas por Barzilay (2014) em relação ao reuso de software, podemos destacar a enorme dificuldade de selecionar componentes de software reusáveis. Questões como incompatibilidades de integração prejudicam alcançar a consistência entre uma arquitetura de um sistema e os componentes disponíveis no mercado, resultando em um problema persistente

na Engenharia de Software Baseada em Componentes. A seleção de componentes e consequentemente a seleção de serviços, tem se mostrado ser a etapa de maior risco no processo de desenvolvimento com a utilização de serviços web (Alferez, 2011).

De acordo com Katawut (2015), a maioria dos processos de seleção de serviços web leva em consideração apenas atributos de qualidade para avaliá-los, tais como: preço, disponibilidade, confiabilidade e tempo de resposta. No entanto, requisitos funcionais influenciam na integração entre os serviços, e por consequência, uma contribuição na qualidade geral da arquitetura do software. Quanto menor a quantidade de incompatibilidades geradas, maior a efetividade da integração entre os serviços web, consequentemente, o custo para adaptação dos serviços diminui.

Escolher os serviços web adequados para uma configuração arquitetural é uma tarefa muito importante e complexa para ser realizada de forma manual, pois não pode ser baseada apenas na experiência do arquiteto e do engenheiro de software. Como para cada serviço web especificado na arquitetura pode existir várias implementações candidatas com funcionalidades correspondentes, a quantidade de possibilidades geradas apresenta complexidade exponencial, onde a base é o número médio de serviços web candidatos e o expoente é o número de serviços web especificados na arquitetura, tornando impossível a análise das implementações dos serviços web candidatos sem nenhum auxílio computacional.

Como o espaço de busca do processo de seleção de serviços web é gigantesco e complexo para a capacidade humana, automatizar um processo de busca exaustiva, guiada por métricas de software, também não é uma tarefa trivial. Consequentemente, cabe então recorrer à abordagens como a Engenharia de Software Baseada em Buscas (SBSE - *Search-Based Software Engineering*), onde problemas da Engenharia de Software são tratados como problemas de busca para serem otimizados por algoritmos metaheurísticos, como por exemplo Algoritmo Genético (AG) (Harman, 2012).

1.2. Objetivos do Trabalho

O objetivo geral deste trabalho é selecionar de forma automática serviços web com base em métricas funcionais e estruturais, a fim de montar uma configuração arquitetural que atenda a uma especificação desejada, minimizando o impacto de adaptação, e assim reduzir o tempo e o custo do processo de desenvolvimento de um sistema de software. Para que isso se torne possível, a abordagem proposta adota métricas funcionais e estruturais, que avaliam a

similaridade entre especificações e implementações de serviços, bem como as dependências entre os mesmos.

Para alcançar o objetivo geral do trabalho, foram definidos os seguintes objetivos específicos:

- Elabor métricas funcionais e estruturais a serem aplicadas neste processo de seleção de serviços web;
- Adotar técnicas da SBSE para automatizar o processo de seleção;
- Definir metaheurística que será utilizada na implementação e validação do presente trabalho;
- Implementar processo de automatização da seleção de serviço web com a metaheurística escolhida;
- Avaliar proposta a partir de experimentos provenientes de estudos de casos.

1.3. Organização do Trabalho

O trabalho está organizado da seguinte forma. No Capítulo 2 são introduzidos alguns conceitos fundamentais relacionados à abordagem proposta, tendo como objetivo facilitar a compreensão do trabalho proposto. O Capítulo 3 apresenta os trabalhos relacionados, destacando as semelhanças e diferenças em relação ao presente trabalho, e, assim, evidenciando suas principais contribuições. O Capítulo 4 apresenta a abordagem proposta, descrevendo as métricas definidas para avaliar os serviços web candidatos, como também o algoritmo metaheurístico para buscar soluções satisfatórias. No Capítulo 5 é realizada a avaliação experimental com a aplicação de testes comparativos visando avaliar a abordagem proposta. Por fim, no Capítulo 6 são apresentadas as considerações finais, evidenciando as contribuições, limitações e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo, apresentaremos os conceitos das áreas mais relevantes presentes no trabalho e cujo conhecimento é de extrema importância para o real entendimento da abordagem proposta: *Arquitetura de Software*, com ênfase no uso de métricas para avaliar uma configuração arquitetural; *SOA* e *Web Services*, apresentando os conceitos e padrões adotados a fim de realizar a seleção e composição de serviços web e montar um sistema de software; *WSDL*, com o objetivo de situar sua importação nesse trabalho; *SOAP* e *REST*, realizando uma discussão sobre as tecnologias; *Engenharia de Software Baseada em Buscas*, com foco na utilização de metaheurísticas como estratégia para resolver problemas com grandes e complexos espaços de busca.

2.1. Arquitetura de Software

Com o passar do tempo, a atividade de desenvolver software vem se tornando cada vez mais complexa. Doravante disso, pode-se dizer que o desenvolvimento de um sistema computacional vai além dos algoritmos e estruturas de dados. A concepção e a especificação da estrutura geral do sistema surgem como um novo tipo de problema. Questões estruturais incluem uma total organização do processo, além de protocolos de comunicação, sincronização e acesso a dados; distribuição física; escalabilidade e desempenho; seleção e composição entre as alternativas de elementos do projeto (Garlan, 1994). Existe um considerável corpo de trabalho sobre este tema, incluindo linguagens utilizadas para conexão entre os módulos, sistemas que atendem a necessidades de domínios específicos e modelos formais de mecanismos para integração entre as partes do software.

A disciplina de Arquitetura de Software é muito ampla e engloba um conjunto de decisões significativas a respeito da organização de um sistema de software, nas quais podemos incluir a seleção de elementos estruturais e interfaces que compõem o sistema de software (Garlan, 1994). Um projeto de Arquitetura de Software está preocupado com a compreensão de como um sistema deve ser organizado e com a estrutura geral do sistema. Em um modelo de processo de desenvolvimento de software, o projeto arquitetural é o primeiro estágio, o qual é o elo crítico entre o projeto e a especificação, pois identificada os principais componentes

estruturais de um sistema e os relacionamentos entre eles. O resultado do projeto de arquitetura de um sistema de software é um modelo que descreve como o sistema está organizado.

A arquitetura de software de um sistema consiste na definição dos componentes de software, suas propriedades externas, e seus relacionamentos com outros softwares. O termo também se refere à documentação da arquitetura de software, que facilita a comunicação entre os envolvidos no processo de construção de um sistema de software, e, além de registrar as decisões tomadas acerca do projeto, também permite o reuso dos componentes e padrões utilizados no projeto (Shaw, 1996).

A origem da arquitetura de software como um conceito foi primeiramente identificado por Edsger Dijkstra (1968). Os estudos nessa área aumentaram significantemente de popularidade em meados dos anos 1990, com os trabalhos de pesquisa concentrando-se nos padrões de estilo de arquitetura de software, linguagens de descrição de arquitetura de software, documentação de arquitetura de software, e métodos formais (Garlan, 1996). A arquitetura de software é normalmente organizada em visões (Bass, 2012), sendo possível realizar uma analogia com os tipos de projetos utilizados na engenharia civil. A IEEE (2006) define visões como instâncias de pontos de vista, onde cada ponto de vista existe para descrever a arquitetura na perspectiva dos envolvidos no processo de desenvolvimento de um sistema de software. Alguns tipos de visões são detalhados a seguir (Kruchten, 1995).

- **Visão Lógica**: mostra as abstrações fundamentais do sistema como objetos ou classes de objetos;
- **Visão de Processo**: mostra como em tempo de execução o sistema é composto de processos interativos. Esta visão é útil para entender características não funcionais do sistema, como desempenho e disponibilidade;
- **Visão de Desenvolvimento**: mostra como o software é decomposto para o desenvolvimento, ou seja, apresenta a distribuição do software em componentes que são implementados.
- **Visão Física**: mostra o hardware do sistema e como os componentes de software são distribuídos entre os processadores.

Na prática, visões são conceitos que, quase sempre, são desenvolvidos durante o processo de projeto. Várias linguagens para descrição da arquitetura de software foram desenvolvidas, porém, há uma grande dificuldade para estabelecer um padrão para realizar uma representação do sistema.

A ideia de padrões como uma forma de apresentar, compartilhar e reusar o conhecimento sobre sistemas de software é hoje amplamente utilizada. É possível pensar em um padrão de arquitetura como uma descrição abstrata, estilizada de boas práticas, experimentadas e testadas em diferentes sistemas e ambientes. Assim, um padrão de arquitetura deve descrever uma organização do sistema bem-sucedida em sistemas anteriores. Deve incluir informações de quando o uso desse padrão é adequado, juntamente com seus pontos fortes e fracos (Schmidt, 2000). A seguir são apresentadas algumas arquiteturas amplamente utilizadas pelo mercado no processo de desenvolvimento de software.

- **Arquitetura em camadas**: essa abordagem apoia o desenvolvimento incremental de sistemas. Quando uma camada é desenvolvida, alguns dos serviços prestados por ela podem ser disponibilizados para os usuários. Enquanto sua interface não sofrer alteração, uma camada pode ser substituída por outra equivalente. Além disso, quando uma interface muda ou tem novos recursos adicionados, apenas a camada adjacente é afetada.
- **Arquitetura de repositório**: a maioria dos sistemas que usam grandes quantidades de dados é organizada em torno de um banco de dados ou repositório compartilhado. Esse modelo é, portanto, adequado para aplicações nas quais os dados são gerados por um componente e utilizados por outro.
- **Arquitetura cliente-servidor**: um sistema que segue o padrão cliente-servidor é organizado como um conjunto de serviços e clientes que acessam e usam estes serviços. Arquiteturas desse tipo são normalmente consideradas arquiteturas de sistemas distribuídos, mas o modelo lógico de serviços independentes rodando em servidores separados pode ser implementado em um único computador.
- **Arquitetura de duto e filtro**: é um modelo de organização em tempo de execução de um sistema no qual as transformações funcionais processam suas entradas e produzem saídas. Cada etapa do processamento é implementada como uma transformação. Os dados de entrada fluem por meio dessas transformações até serem convertidos em saídas. As transformações podem ser executadas sequencialmente ou em paralelo. Os dados podem ser processados por cada item de transformação ou em um único lote.
- **Arquitetura Orientada a Serviços**: é um estilo de arquitetura de software cujo princípio fundamental prega que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços (Lublinsky, 2007). Frequentemente

estes serviços são conectados através de um "barramento de serviços" que disponibiliza interfaces, ou contratos, acessíveis através de *web services* ou outra forma de comunicação entre aplicações (Krafzig, 2004). Arquitetura Orientada a Serviços é baseada nos princípios da computação distribuída e utiliza o paradigma *request/reply* para estabelecer a comunicação entre os sistemas clientes e os sistemas que implementam os serviços (Raghu, 2005). Além da perspectiva estritamente técnica, SOA também se relaciona com determinadas políticas e conjuntos de "boas práticas" que pretendem criar um processo para facilitar a tarefa de encontrar, definir e gerenciar os serviços disponibilizados (Bobby, 2006). A implementação desse tipo de arquitetura depende de uma rede de serviços de software, que, ao invés de realizar chamadas diretas para o código fonte, os serviços definem protocolos que descrevem como enviar e receber as mensagens utilizando descrições em metadados.

Uma avaliação arquitetural é de extrema importância para prever a qualidade de uma arquitetura de software antes que um sistema seja construído, como também possibilita estimar os principais ganhos na utilização do tipo de arquitetura escolhido (Kazman, 1993). O propósito dessa avaliação é identificar riscos potenciais e verificar se os requisitos de qualidade foram incorporados no projeto arquitetural. Existem na literatura várias abordagens para a avaliação de uma arquitetura de software, conforme abordado no trabalho de Oliveira (2010), podendo ser divididas em três categorias:

- **Avaliação de atributos de qualidade:** são estudos que realizam a avaliação de atributos como desempenho, reusabilidade, confiabilidade, entre outros. Segundo Clarke (2003), a avaliação dos atributos de qualidade, geralmente, é executada avaliando individualmente cada serviço web e a soma do escore qualitativo de todos os serviços é atribuída à qualidade geral da arquitetura. Para avaliar os serviços, podem ser realizadas simulações entre os serviços web, por exemplo, para mensurar o tempo de resposta na chamada de uma operação provida por determinado serviço.
- **Avaliação estrutural:** são estudos que realizam a avaliação arquitetural conforme sua estrutura em termos de métricas que analisam interfaces, assinaturas de métodos e utilização de serviços. Na perspectiva estrutural, a avaliação de incompatibilidades no que diz respeito a integração entre os serviços possui o benefício de aumentar a probabilidade de que tais inconsistências possam ser identificadas antes que o sistema seja construído, potencialmente prevenindo grande perda de tempo e esforço no desenvolvimento de um sistema de software (Egyed, 2000). As métricas definidas para

este trabalho, buscam realizar uma avaliação estrutural da arquitetura do software a ser construído.

- **Avaliação de escopo:** são estudos que realizam a análise de riscos e benefícios, adoção e melhoria de processos, estimativas, etc. A avaliação de escopo está interligada com características referentes ao processo de construção do projeto arquitetural.

A avaliação dos atributos de qualidade e estrutural são mais adequadas para serem utilizadas no processo de seleção de serviços web, visto que deve-se avaliar configurações arquiteturais candidatas. Os serviços pertencentes à arquitetura contribuem para a avaliação de atributos de qualidades da arquitetura, pois a qualidade de cada serviço reflete na qualidade geral da arquitetura. A qualidade na integração entre os serviços web contribui para a avaliação estrutural.

2.2. SOA e Serviços Web

Ao trabalhar com SOA (*Service-oriented Architecture*), são proporcionados o entendimento e a análise de toda arquitetura e funcionalidade do sistema, como por exemplo, permitindo observar o nível de dependência entre os serviços (Cheesman, 2001). Minimizar a dependência entre serviços é uma boa prática de projeto, pois resulta na diminuição da complexidade do sistema. SOA permite também que serviços web sejam utilizados como componentes reutilizáveis de software, que em contraposição ao desenvolvimento de todas as partes de um sistema, é um fator que pode levar ao aumento da qualidade e da produtividade da atividade de desenvolvimento de software. Desta forma, podemos ver o uso de SOA como algo que traz agilidade e qualidade ao desenvolvimento de sistemas de software.

A Figura 1 ilustra uma Arquitetura Orientada a Serviços. O provedor de serviços projeta, implementa e específica interfaces para os serviços web, tendo como função também, a publicação de informações sobre o serviço em um registro acessível. Já o solicitante (cliente) do serviço descobre as especificações deste serviço, e o localiza no provedor. Em seguida, o solicitante pode conectar seus sistemas a serviços específicos. Os serviços web representam um grande avanço no desenvolvimento de um sistema computacional, pois facilita a integração entre os sistemas, possibilitando o crescimento de sistemas que utilizam esta tecnologia.

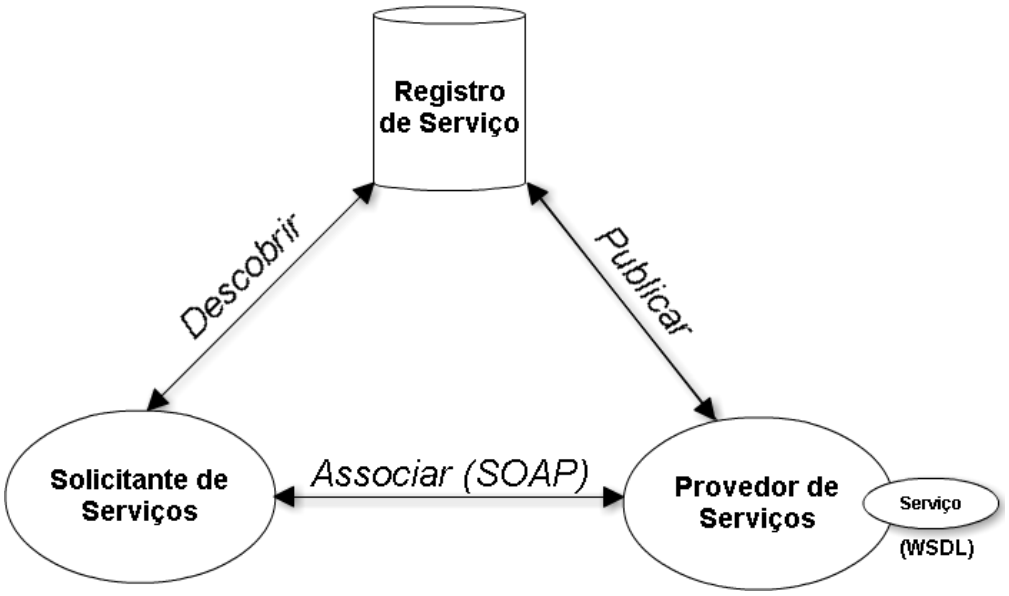

Figura 1 - Arquitetura Orientada a Serviços

Na perspectiva de *web services* (serviços web), o consórcio OASIS (2012) descreve uma Arquitetura Orientada a Serviços como um paradigma para organização e utilização de entidades distribuídas que estão sob controle de diferentes domínios. SOA não está diretamente relacionada e nem dependente de serviços web, porém é com o uso de tal tecnologia que os benefícios de SOA são melhores alcançados, pois são independentes de linguagem de programação, possuem baixo acoplamento, são reusáveis, viabiliza a construção de sistemas de software em grande escala, são acessados usando protocolos baseados em padrões da internet e são descritos em XML.

A tecnologia de serviços web surgiu com o intuito de atender a um dos principais problemas da área de tecnologia da informação, mais precisamente no processo de desenvolvimento de software, que foi e continua sendo a integração de sistemas. Esta tecnologia se desenvolveu a partir da especificação de um conjunto de padrões abertos baseados em XML, que tornam os serviços web independentes de protocolo de rede, plataforma e linguagem de programação. Os padrões que formam a base de serviços web são (Erl, 2004):

- **WSDL** (*Web Services Description Language*): um padrão baseado em XML desenvolvido pelo W3C para descrever o serviço web em termos funcionais. A WSDL é extensível para permitir a descrição dos serviços e suas mensagens independentemente do formato e protocolo de rede utilizado. A WSDL descreve os serviços disponibilizados à rede através de uma especificação em XML, e é responsável por introduzir uma gramática comum para descrever os serviços;

- **SOAP** (*Simple Object Access Protocol*): define um protocolo para invocação de uma funcionalidade remota, e, para tal, necessita representar o endereço do serviço web remoto, o nome da operação, e os parâmetros necessários. Estes dados são formatados em XML com determinadas regras e enviados para o serviço web, normalmente por protocolos de aplicação, tal como HTTP. SOAP define os componentes essenciais e opcionais das mensagens passadas entre os serviços, além de permitir que documentos XML de envio e de recepção suportem um protocolo comum de transferência de dados via rede;
- **UDDI** (*Universal Description, Discovery, and Integration*): uma iniciativa promovida originalmente pela IBM, Microsoft e Arriba, com objetivo de acelerar a interoperabilidade e utilização de serviços web, com a proposta de um registro de nomes de organizações e de descrições dos serviços. UDDI é um repositório de serviços web, onde fornecedores e consumidores podem registrar (publicar) e buscar (descobrir) serviços web. O UDDI possui três funções principais: (*i*) **publicação**: permite que uma organização divulgue seus serviços web; (*ii*) **descoberta**: permite que o cliente do serviço procure e encontre um determinado serviço web; e (*iii*) **associação**: permite que o cliente do serviço web obtenha as informações necessárias para que possa estabelecer a conexão e interagir com o serviço.

Juntas, essas especificações permitem que os serviços web interajam entre si, pois definem um formato para troca de mensagens, determinam um padrão para a especificação das interfaces, criam convenções para o mapeamento entre as mensagens e a implementação do serviço web, e definem mecanismos para publicação e busca de tais serviços.

A seguir são descritos alguns dos benefícios da utilização de serviços web (Erl, 2005):

- **Distribuição**: os sistemas de informação estão sendo utilizados em ambientes cada vez mais distribuídos, exigindo que os componentes do sistema sejam executados em máquinas fisicamente separadas.
- **Heterogeneidade**: sistemas tipicamente distribuídos são executados em ambientes heterogêneos e que muitas vezes não estão sob controle dos desenvolvedores.
- **Dinamismo**: para atender aos ambientes de negócio cada vez mais dinâmicos, os sistemas precisam ser cada vez mais flexíveis para atender frequentes mudanças.
- **Transparência**: os sistemas devem ser transparentes aos detalhes de infraestrutura de comunicação dos diferentes pontos onde executam os diversos componentes.

- **Orientação a processos**: sistemas devem atender aos processos de negócio (*workflows*) presentes no ambiente em que são utilizados.

O objetivo na adoção de serviços web é alcançar a interoperabilidade universal entre os sistemas usando os padrões da internet. Os serviços web utilizam um modelo de integração de baixo acoplamento para permitir a integração flexível de sistemas heterogêneos em uma variedade de domínios, incluindo *business-to-consumer* e *business-to-business*.

A integração de sistemas requer mais do que a capacidade de conduzir interações simples, utilizando protocolos. O modelo de interação que é diretamente suportado pela WSDL é essencialmente um modelo sem estado de interações assíncronas (os serviços podem ou não estar executando simultaneamente) ou síncronas (os serviços encontram-se executando simultaneamente) não correlacionadas (Erl, 2007).

Modelos para interações entre serviços web assumem tipicamente uma sequência de troca de mensagens *peer-to-peer*, tanto para as síncronas quanto para as assíncronas, dentro de um padrão com interações de longa duração que envolve duas ou mais partes. Para definir essas interações, é necessária uma descrição formal dos protocolos de troca de mensagens utilizados pelos serviços web. A definição de tais protocolos de troca de mensagens envolve especificar precisamente o comportamento de cada uma das partes envolvidas na troca de mensagens, sem revelar sua implementação interna.

2.2.1. WSDL

Os serviços se comunicam por meio de troca de mensagens expressas em XML, e essas mensagens são distribuídas usando protocolos da internet, como HTTP, SMTP, TCP/IP. Um serviço define o que precisa de outro, definindo seus requisitos em uma mensagem e enviando-a para o serviço parceiro. O serviço requisitado analisa a mensagem, efetua o processamento, e depois envia a resposta como uma mensagem para o serviço solicitante. Ao contrário dos componentes de software, os serviços não usam chamadas de procedimento ou métodos remotos para acessar as funcionalidade associadas a outros serviços web (Kavantzas, 2004).

Como dito anteriormente, os serviços são descritos em uma linguagem baseada em XML, chamada WSDL. Segundo Erl (2004), uma especificação WSDL define três aspectos de um serviço web, são eles: (*i*) **o que faz um serviço**: este tópico é denominado de interface, e especifica quais operações o serviço suporta e define o formato das mensagens que são enviadas e recebidas pelo serviço; (*ii*) **como ele se comunica**: também denominado de associação, realiza o mapeamento da interface abstrata para um conjunto concreto de protocolos, especificando os

detalhes técnicos de como se comunicar com um serviço web; e (*iii*) **onde o encontrar**: que descreve o local da implementação de um serviço específico.

O modelo conceitual apresentado na Figura 2 mostra os elementos de uma descrição de serviço. Cada um desses é expresso em XML e podem ser fornecidos em arquivos separados (Erl, 2004).

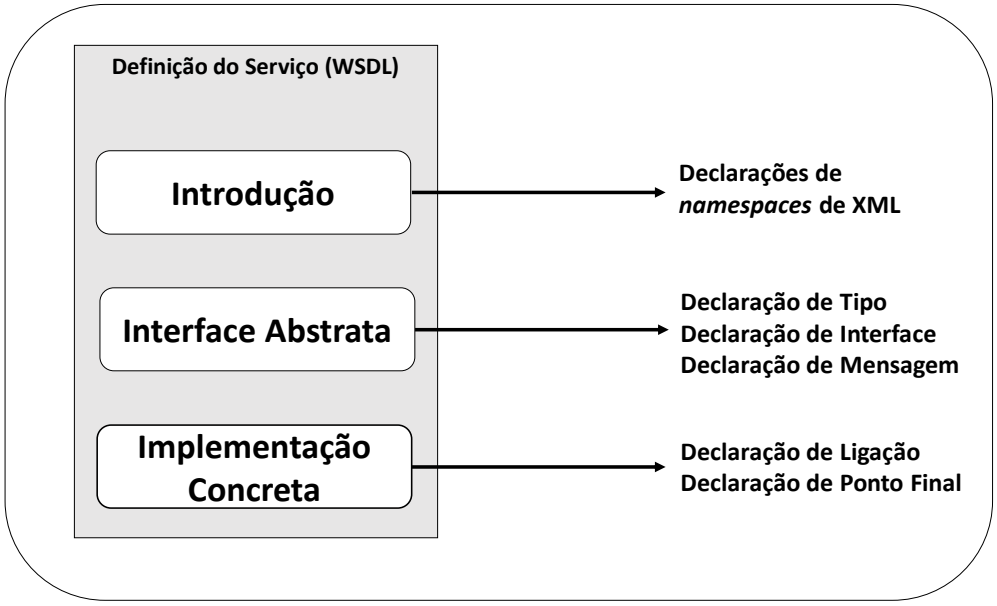

Figura 2 - Organização de uma especificação WSDL

- Uma parte introdutória que geralmente define os namespaces de XML usados e que pode incluir uma seção de documentação, fornecendo informações adicionais sobre o serviço;
- Uma descrição opcional dos tipos usados em mensagens trocadas pelo serviço;
- Uma descrição da interface do serviço, ou seja, uma descrição das operações que proveem funcionalidades para outros serviços;
- Uma descrição das mensagens de entrada e saída processadas pelo serviço;
- Uma descrição da associação utilizada pelo serviço. A mais utilizada é SOAP, no entanto, outras podem ser especificadas. A associação define como as mensagens de entrada e saída do serviço devem ser empacotadas, e especifica os protocolos de comunicação a serem utilizados.

Atualmente, as especificações WSDL raramente são escritas a mão, pois a maioria das informações de uma especificação pode ser gerada automaticamente. Não é necessário conhecer os detalhes de uma especificação para compreender os princípios de uma WSDL. Dessa forma, é descrito aqui apenas conceitos da interface abstrata. Essa é a parte de uma especificação

WSDL que equivale a interface provida de um componente de software. A Figura 3 apresenta parte da interface para um serviço simples que, dada uma data e um lugar específico, retorna os horários e preços de voos.

```
<types>
    <xs:schema targetNameSpace="http://.../weathns"
        xmls:weathns="http://.../weathns">

        <xs:element name="PlaceAndDate" type="pdrec" />
        <xs:element name="MaxMinTemp" type="mmtrec" />
        <xs:element name="InDataFault" type="ermess" />
        <xs:complexType name="pdrec">
            <xs:sequence>
                <xs:element name="cidade" type="xs:string" />
                <xs:element name="pais" type="xs:string" />
                <xs:element name="dia" type="xs:date" />
            </xs:sequence>
        </xs:complexType>
        ...
        ...
        ...
    </schema>
</types>
<interface name="weatherInfo">
    <operation name="getMaxMinTemp" pattern="wsdlns:in-out">
        <input menssageLabel="In" element="weathns:PlaceAndDate" />
        <output menssageLabel="Out" element="weathns:MaxMinTemp" />
        <output menssageLabel="Out" element="weathns:InDataFault" />
    </operation>
</interface>
```

Figura 3 - Parte de uma descrição WSDL

Na Figura 3, a primeira parte da descrição mostra o elemento e o tipo de definição usado na especificação do serviço. Isso define os elementos *PlaceAndDate*, *MaxMinTemp* e *InDataFault*. A segunda parte mostra como a interface do serviço é definida. Nesse exemplo, o serviço *weatherInfo* tem uma única operação. Tal operação tem um padrão *in-out*, o que significa que ela recebe uma mensagem de entrada e gera uma mensagem de saída. A especificação WSDL permite vários padrões de trocas de mensagens diferentes, como *in-out*, *out-only*, *in-optional-out* e *out-in*.

O grande problema com WSDL é que a definição de interface do serviço não inclui quaisquer informações sobre a semântica do serviço, suas características não funcionais, ou suas dependências em relação a outros serviços. É simplesmente uma descrição da assinatura do serviço, contendo suas operações e seus parâmetros. Ao se planejar utilizar um serviço, é necessário avaliar o que o serviço realmente faz e o que significam os diferentes campos nas mensagens de entrada e saída (Erl, 2007).

2.2.2. SOAP *x* REST

Observando as especificações atuais associadas aos serviços web, é possível identificar duas interfaces bastante utilizadas para implementação de serviços: SOAP e REST. A seguir, inicialmente, é apresentado a especificação para troca de mensagens entre serviços, denominada SOAP, e posteriormente, o modelo arquitetural que tem por base os pressupostos estabelecidos pelo modelo REST (*Representational State Transfer*) (Erl, 2012).

SOAP é um protocolo para troca de informações estruturadas, viabilizando a criação de uma plataforma descentralizada e distribuída. Ele se baseia em XML para seu formato, onde uma mensagem SOAP encapsula o conteúdo e pode ser trafegada via HTTP, JMS ou outro protocolo (Erl, 2004) .

Utilizar SOAP traz uma carga adicional não encontrada ao usar REST, mas há também inúmeras vantagens. Primeiramente, SOAP é baseado em XML e dividido em três partes: (*i*) o envelope que define o conteúdo da mensagem e informa como processá-la; (*ii*) um conjunto de regras de codificação para os tipos de dados; e (*iii*) os procedimentos de chamadas e respostas. Na requisição, o envelope é enviado por meio de um protocolo, por exemplo HTTP, e processado no destino como uma chamada de procedimento remoto (RPC - *Remote Procedure Call*). Já na resposta, o envelope retorna com as informações na forma de um documento XML. Pensando em serviços que utilizam mensagems no protocolo SOAP, o mais comum é descrever a interface do mesmo usando a linguagem WSDL.

Ao adotar SOAP como padrão para troca de menssagens, cada serviço tem um arquivo de descrição WSDL, que define suas operações, os tipos de dados que são usados nas requisições e respostas, bem como os endereços de rede do serviço (*endpoints*). Uma associação que ajuda o entendimento deste ponto é pensar nas operações como métodos de uma classe, a assinatura do método como as mensagens definidas, e o tipo de cada campo da assinatura como a definição da estrutura de dados que será usada nas mensagens (Kavantzas, 2004).

Enquanto SOAP é um protocolo bem definido para troca de mensagens entre os serviços, REST é um estilo arquitetural utilizado em projetos de aplicações web que contam com recursos nomeados por meio de uma URI (*Uniform Resource Identifier*), utilizando protocolo HTTP como mecanismo de transporte, seu cabeçalho, seus métodos e toda a infraestrutura web já bem estabelecida, reconhecida e utilizada (Erl, 2012).

REST é uma abstração arquitetural que consiste de um conjunto coordenado de restrições aplicadas a componentes, conectores e elementos de dados dentro de um sistema

distribuído (Fielding, 2000). O termo REST se refere a um conjunto de princípios arquiteturais para descrever qualquer interface web que utiliza XML, JSON, ou até mesmo texto puro, juntamente com o protocolo HTTP, sem as abstrações adicionais dos protocolos baseados em padrões de trocas de mensagem, tal como, o protocolo SOAP amplamente adotado pelos serviços web. O estilo arquitetural REST é aplicado no desenvolvimento de serviços web, podendo caracterizar os serviços como "RESTful" se eles estiverem em conformidade com algumas restrições. Entre as restrições definidas, destacam-se três (Fielding, 2000):

- **Identificação global**: todos os recursos são identificados através de um mesmo mecanismo global. Independentemente do contexto de aplicação, tecnologia ou implementação, cada recurso disponível na web é identificado utilizando um URI;
- **Interfaces uniformes**: a interação entre os agentes é feita por meio do protocolo HTTP utilizando um conjunto de métodos predefinidos: GET, POST, PUT e DELETE.
- **Interações *stateless***: o estado concreto de uma aplicação web é mantido com base no estado de um conjunto de recursos. O servidor conhece o estado dos seus recursos mas não mantém informações sobre as sessões dos clientes.

  Por outro lado,

A seguir são apresentados alguns beneficios da utilização de REST e SOAP no deenvolvimento de serviços web (Erl, 2012).

- REST se torna mais elegante, pois utiliza ao máximo o protocolo HTTP, evitando a construção de protocolos adicionais;
- REST tem o potencial de ser bem mais simples que uma implementação com WSDL e SOAP;
- Com REST é possível ter diverssas representações de um mesmo recurso. Por exemplo, uma determinada entidade pode ser representada em diferentes formatos como JSON, XML, HTML, ou texto, dependendo da requisição feita pelo cliente;
- REST possibilita a navegação entre relacionamentos de vários recursos de forma dinâmica, seguindo a usabilidade de qualquer sistema web;
- SOAP é um padrão que combinado às especificações de serviços web podem garantir questões de QoS (*Quality of Service*), segurança, transação e outras questões presentes nas integrações entre os serviços;
- Uma mensagem SOAP pode ser propagada por diferentes protocolos, o que flexibiliza bastante várias integrações.

- SOAP é um padrão que está muito maduro no mercado. Diversas ferramentas de integração e frameworks possuem várias funcionalidades para manipular as mensagens SOAP.

É importante frisar que ambas as tecnologias podem ser misturadas, combinadas e manipuladas conjuntamente. O REST é fácil de entender e extremamente acessível, porém faltam padrões e a tecnologia é considerada apenas uma abordagem arquitetural. Em comparação, o SOAP é um padrão da indústria, com protocolos bem definidos e um conjunto de regras bem estabelecidas. É possível afirmar que os casos onde REST funciona bem são (Fielding, 2000):

- **Situações em que há limitação de recursos e de largura de banda**: A estrutura de retorno é em qualquer formato definido pelo desenvolvedor e qualquer navegador pode ser usado. Isso é possível pois a abordagem REST usa o padrão de chamadas GET, PUT, POST e DELETE;
- **Operações totalmente sem-estado**: se uma operação precisa ser continuada, o REST não será a melhor opção. No entanto, se forem necessárias operações de CRUD (Criar, Ler, Atualizar e Excluir), o REST seria a melhor alternativa;
- **Situações que exigem cache**: se a informação pode ser armazenada em cache, devido à natureza da operação, esse seria um cenário adequado para a tecnologia.

Essas três situações abrangem muitas soluções, mas vale ressaltar que SOAP é bastante maduro e bem definido e vem com uma especificação completa. Enquanto REST é apenas uma abordagem. Por isso ao encontrar uma das situações abaixo, o SOAP pode ser uma ótima solução (Erl, 2007):

- **Processamento e chamada assíncronos**: se o serviço precisa de um nível garantido de confiabilidade e segurança para a troca de mensagens, então SOAP oferece padrões adicionais para esse tipo de operação como por exemplo o WSRM (*WS-Reliable Messaging*);
- **Contratos formais**: se ambos os lados (fornecedor e consumidor) têm que concordar com o formato de troca de dados, então SOAP fornece especificações rígidas para esse tipo de interação;
- **Operações *stateful***: para o caso do serviço precisar de informação contextual e gerenciamento de estado com coordenação e segurança, SOAP possui uma especificação adicional em sua estrutura que apoia essa necessidade (segurança, transações, coordenação, etc.).

Como pode ser observado, cada uma das abordagens possui sua utilidade. Ambas têm suas vantangens e desvantagens nos quesitos de segurança, mecanismos de transporte, etc. Contudo, para a abordagem proposta foi escolhido SOAP em conjuto com as especificações WSDL, tendo em vista sua maturidade, comprovados casos de sucesso, e uma grande variedade de pesquisas já realizadas nesse tema.

2.4. Engenharia de Software Baseada em Buscas

A Engenharia de Software Baseada em Buscas (SBSE – *Search-Based Software Engineering*) é um campo da Engenharia de Software que aborda técnicas e métodos que reformulam e modelam problemas que podem ser resolvidos usando técnicas de otimização baseadas em buscas metaheurísticas (Harman, 2007).

Diversas atividades realizadas no desenvolvimento de projetos de Engenharia de Software, como a seleção e composição de serviços web, são atreladas a problemas complexos, que apresentam uma gama enorme de possibilidades e são definidos como problemas NP-Completo (Freitas, 2009). Atualmente, a adoção de técnicas de otimização baseadas em buscas metaheurísticas tem se mostrado efetiva e eficiente em solucionar de forma automática problemas NP-Completo, obtendo soluções próximas da ótima ou até mesmo encontrando as soluções ótimas para os problemas.

As primeiras abordagens do conceito de Engenharia de Software Baseada em Buscas surgiram em meados da década de 1970, mas eram tratadas com pouca atenção pela comunidade acadêmica. Como a tendência é que os sistemas de software se tornem mais complexos e desafiadores com o passar do tempo, houve a necessidade de explorar novas vertentes. Esse cenário contribuiu para que esse campo de pesquisa fosse abraçado pela comunidade acadêmica, onde o interesse pela aplicação dessa técnica cresce constantemente.

Essa técnica emergente demonstra potencial para se consolidar na Engenharia de Software, uma vez que as técnicas de otimização são aplicadas com sucesso em outras disciplinas de engenharia, tais como: Engenharia Mecânica, Engenharia Elétrica, Engenharia Civil, entre outras.

Conforme dito por Harman (Harman, 2010), algoritmos de otimização são adequados para um conjunto de aplicações oriundas da Engenharia de Software, pois a essência virtual dessa disciplina permite que as buscas metaheurísticas ocorram por mapeamento ou reformulações. As características desses algoritmos aplicados na SBSE, à torna: a) **Escalável**: o desempenho dos algoritmos não demonstra decaimento significativo, pelo motivo da alta

capacidade de paralelização dessas abordagens algorítmicas; b) **Genérica**: os algoritmos são compatíveis com diversos problemas, visto que possuem métodos que facilitam a representação do domínio do problema; c) **Robusta**: tem a habilidade de lidar com fatores adversos como ruídos e dados incompletos; d) **Rica em introspecção**: pode gerar resultados além da imaginação dos engenheiros; e) **Realística**: lida com múltiplos objetivos potencialmente conflitantes do dia a dia, em que reduzir o tempo de um projeto impacta na qualidade, no escopo, entre outras características do projeto (Vergilio, 2011).

Problemas de busca complexos e passíveis de serem reformulados matematicamente estão distribuídos em diversas áreas da Engenharia de Software. Atividades como alocação de recursos, presente no campo de engenharia de requisitos, ou seleção de testes, inserido no campo de testes de software, são exemplos desses problemas. Apesar de distribuídos em vários estágios do ciclo de vida no desenvolvimento de software, a adoção de metaheurísticas não é uniforme, pois ainda existe uma concentração em algumas áreas. A Tabela 1 apresenta exemplos de problemas de busca nas principais áreas da Engenharia de Software que se aplicam técnicas de otimização (Vergilio, 2011).

Tabela 1 - Problemas de busca nas principais áreas da Engenharia de Software

Área	Exemplos de Problemas
Planejamento de projeto	Estimativa de custo
Engenharia de requisitos	Priorização de requisitos
Design de software	Melhoria de qualidade do design arquitetural
Implementação/codificação	Otimização de código-fonte
Integração	Integração de componentes de software
Teste/validação	Priorização de casos de teste
Deployment	Implantação de componentes de software
Manutenção	Refatoração automática

Para uma efetiva utilização da Engenharia de Software Baseada em Buscas, é necessário a compreensão de apenas dois ingredientes fundamentais, são eles: (*i*) a escolha da representação do problema, e (*ii*) a definição da função de *fitness*. Esta simplicidade torna a SBSE bastante atraente para ser utilizada em problemas de otimização. Com apenas esses dois ingredientes simples, a Engenharia de Software Baseada em Busca pode explorar algoritmos de otimização e obter resultados satisfatórios. Normalmente, um engenheiro de software terá uma representação adequada para o seu problema. Muitos problemas em engenharia de

software também possuem métricas de software associados, o que naturalmente formam bons candidatos iniciais para funções objetivo (Harman, 2010).

A representação do problema é altamente dependente do domínio do problema, pois cada mapeamento é exclusivo para determinado problema da Engenharia de Software (Harman, 2004). Uma metaheurística possui inicialmente caráter genérico, e então é lapidada para o problema em questão, onde o esforço intelectual é para a modelagem do problema. No entanto, o esforço humano empregado em tentar buscar a solução do problema é maior que na modelagem metaheurística do problema.

A solução encontrada por esforço humano corre o risco de aparentar ser aceitável, entretanto, após vários ciclos de iteração de projeto, pode-se perceber que a solução encontrada gera resultados desastrosos, consequentemente, resultando em investimento perdido, que é um fator inaceitável na perspectiva do mercado atual. Portanto, o esforço humano empregado deve ser focado na descrição da solução, e não na sua construção (Harman, 2010).

Para que a reformulação do problema obtenha êxito, um processo deve ser seguido. Este processo é composto por três etapas, são elas: representação do problema, função objetivo, operadores de manipulação (Burke, 2012). De certa forma, o engenheiro de software geralmente terá facilidade em representar seu problema de busca em alguma metaheurística, tendo em vista que diariamente o mesmo enfrenta situações na qual deve mapear ou simular modelos concretos em dados. Outra facilidade pode ser observada no fato de que muitos problemas da Engenharia do Software possuem uma gama de métricas de software relacionadas e que elas são potenciais candidatas para serem funções objetivo, ou mesmo se tornarem modelos para construção destas funções (Harman, 2010). Segue abaixo uma explicação de cada fase do processo para reformulação do problema.

A representação do problema consiste em transformar conceitos do problema em símbolos para tornar possível a realização de manipulações simbólicas, como na matemática (Harmam, 2004). Com isso, esta transformação gera uma solução candidata que pode ser avaliada por funções que as classificam como boas ou más soluções. Para serem computadas, as soluções candidatas são representadas como: vetores, matrizes, grafos, código binário, etc. O engenheiro de software deve escolher uma representação que chegue o mais próximo possível do problema em questão. Uma má representação causa uma perda de informação, que, por consequência, impacta nos resultados provenientes da busca.

A função objetivo é criada em termos da representação do problema que precisa ser resolvido, onde soluções boas possuem níveis satisfatórios de adequação, diferentemente de

soluções ruins (Harman, 2004). Segundo Linden (2008), elaborar tal função não é uma tarefa fácil, pois talvez sejam necessárias algumas tentativas para definir propriedades que devem ser mensuradas nas soluções candidatas, antes de encontrar uma função ideal para o problema. Partindo da comparação entre soluções candidatas, a função objetivo deve estabelecer uma escala de medida, onde uma solução coxnsiderada melhor que outra recebe um escore maior que a última. Então, a solução ótima tem que obter o escore máximo, e, consequentemente, nenhuma outra solução pode ultrapassá-la (Linden, 2008). A computação da função objetivo deve ser eficiente porque as metaheurísticas necessitam realizar várias iterações do seu algoritmo, e uma função ineficiente se torna um gargalo no algoritmo de otimização.

A busca aleatória é a forma mais simples de um algoritmo de busca que aparece com frequência na literatura da engenharia de software. No entanto, a busca aleatória não utiliza uma função para guiá-la, o que impossibilita o algoritmo de encontrar soluções globalmente ideais. Soluções de maior qualidade podem ser encontradas com o auxílio de uma metaheurística, tal como algoritmo genético. Isso se torna possível pois o algoritmo utiliza uma função objetivo ou função *fitness* que serve para maximizar ou minimizar o espaço das soluções viaveis. O escore gerado pela função objetivo tem teor numérico, onde cada solução é mapeada para um valor e a função transforma o espaço dessas soluções em uma paisagem de adequação conforme a Figura 4.

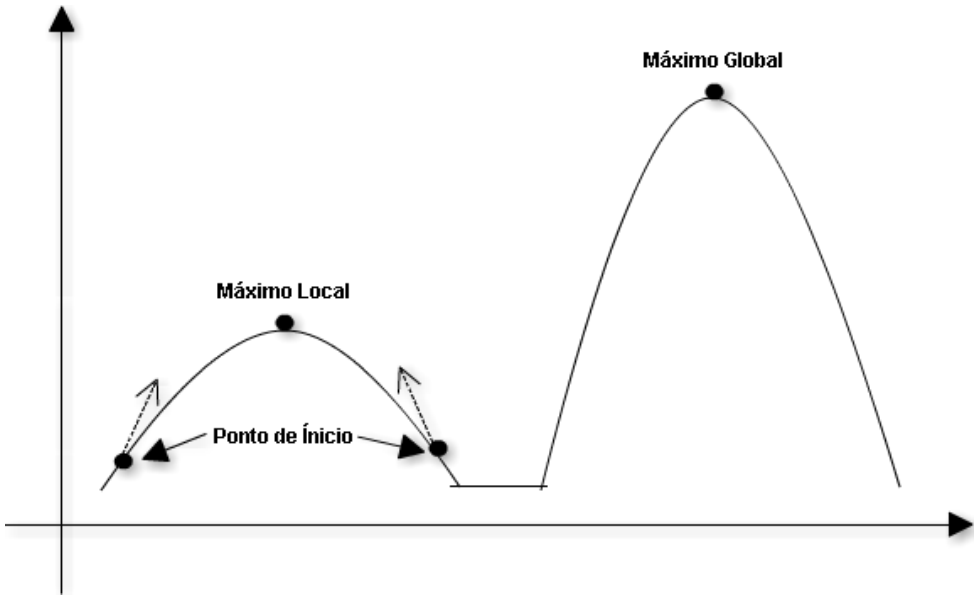

Figura 4 - Espaço de soluções em uma paisagem de adequação

Cada ponto no espaço de busca tem um valor numérico que corresponde ao grau de adequação de uma solução candidata. O máximo global representa a solução ótima, já um

máximo local é caracterizado por uma solução com uma maior pontuação, quando comparada às soluções vizinhas. Para o espaço de busca ser percorrido de forma favorável, é importante atender as seguintes propriedades:

- Não deve possuir uma paisagem aplainada, pois as soluções candidatas tornam-se muito similares, e então a busca poderá perambular sem direção definida, causando dificuldade de encontrar soluções consideravelmente melhores.
- Evitar possuir um espaço com máximos agudos, pois, caso a busca encontre uma solução que representa um máximo local agudo, ficará "presa" porque as soluções vizinhas são muito ruins em relação a esse máximo local e a busca termina por não encontrar soluções melhores.

É importante destacar que o engenheiro de software não tem conhecimento preciso da paisagem de adequação, pois ela representa todo o espaço de busca, e que apenas a busca exaustiva poderia de fato gerá-la. Então, evitar máximos agudos e uma paisagem aplainada irá depender de análises, que são realizadas em um subconjunto de soluções avaliadas pela função objetivo. Caso as análises apontem imperfeições, é requerido o ajuste da função. Mesmo com análises e ajustes, os problemas citados ainda podem persistir diante de um espaço de busca gigantesco e complexo. Portanto, é importante escolher uma metaheurística que possua estratégias para escapar dessas adversidades.

Os operadores de manipulação são capazes de realizar modificações nas soluções candidatas existentes para gerar novas possíveis soluções, com a expectativa de explorar de forma inteligente o espaço de busca (Burke, 2012). Uma escolha errada desses operadores pode restringir a diversidade das soluções encontradas, diminuindo a efetividade do algoritmo de otimização e aumentando a possibilidade de parada em ótimos locais. Diferentes técnicas de busca adotam operadores distintos, que ainda podem sofrer modificações visando uma melhor adequação ao problema que se pretende resolver.

2.4.1. Principais Metaheurísticas

Metaheurísticas são heurísticas genéricas que são mapeadas para problemas de algum domínio específico para buscar soluções aproximadas da ótima. A seguir são apresentadas as principais metaheurísticas abordadas em pesquisas no campo da Engenharia de Software Baseada em Buscas (Colanzi, 2013).

- **Algoritmos Genéticos**: método de busca evolutivo baseado no processo de seleção natural;

- **Arrefecimento Simulado**: método de busca local probabilística que melhora iterativamente a solução inicial no espaço de busca;
- **Enxame de Partículas**: técnica baseada em populações que implementa uma metáfora do comportamento social da interação entre indivíduos de um grupo;
- **Busca Tabu**: estratégia baseada em memória adaptativa que guia um método de busca local que melhora iterativamente a solução inicial do espaço de busca;
- **Colônia de Formigas**: método de busca coletivo, paralelo e dinâmico, que permite resolver problemas que podem ser reduzidos a encontrar bons caminhos em grafos.

Nenhuma busca utilizando a abordagem de metaheurística garante encontrar a solução ótima, pois seu processo de execução é não determinístico, ou seja, diferentes soluções finais podem ser apresentadas, as quais são consideradas razoavelmente boas, e, dentre elas, pode estar a solução ótima. O não determinismo pode ser justificado por fatores aleatórios inerentes às metaheurísticas, tais como soluções iniciais, parâmetros de calibração e critérios de parada. Apesar de ser impossível prever a solução final, é necessário que os resultados provenientes da metaheurística sejam melhores que uma busca puramente aleatória (Barros, 2011).

Dentre as metaheurísticas citadas acima, não existe aquela que é considerada superior à outra de forma universal, ou seja, para qualquer aspecto que se queira comparar nem sempre uma será melhor que a outra. Todas as metaheurísticas possuem o mesmo desempenho quando é realizada a média das avaliações relacionadas a infinitos problemas distintos. Dessa forma, se para certo número de problemas a metaheurística *X* proporciona melhor desempenho, também existe outro conjunto de problemas onde a metaheurística *Y* apresenta melhor eficácia (Jiang, 2015).

Segundo Wolpert (1997), um algoritmo projetado especialmente para algum problema possui melhor desempenho do que outro de cunho genérico. O problema de busca deve ser reformulado para a metaheurística que melhor absorva as propriedades e restrições do problema em questão. Quanto mais detalhes do problema a metaheurística apresentar, melhor será seu desempenho. É importante considerar que a função objetivo deve ser computacionalmente eficiente, ou seja, a metaheurística deve ser capaz de mapear o máximo de detalhes do domínio do problema, e mesmo assim manter sua eficiência computacional. Além disso, a reformulação do problema deve ser fácil de entender para gerar maior confiança na percepção das soluções (Linden, 2008).

Várias dessas metaheurísticas possuem inspiração biológica e pertencem a uma classe denominada de Algoritmos evolutivos (Back, 2000). As principais diferenças entre esses

algoritmos estão relacionadas aos procedimentos auxiliares empregados, denominados de operadores evolutivos. Cada solução é chamada, por exemplo, de *individuo*, *partícula*, *vetor*, *agente*, de acordo com a metaheurística em questão.

Ao longo dos anos diversas aplicações de metaheurísticas em problemas específicos têm sido apresentadas, tais como: problema de roteamento de veículos, problema da mochila, problema do caixeiro-viajante, problema de coloração de grafos, otimização em engenharia de software (Freitas, 2009). Portanto, esse é um campo ativo de investigação, com uma literatura considerável, uma grande comunidade de pesquisadores e usuários, e uma gama extensiva de aplicações, o que indica que trabalhos capazes de melhorar o desempenho de metaheurística costumam ser bem recebidos pela comunidade. A seguir são apresentados mais detalhes sobre a metaheurística Algoritmo Genético, tendo em vista que essa é a técnica de otimização utilizada por este trabalho.

2.5. Algoritmo Genético

No trabalho proposto por Holland (1975), algoritmo genético é uma metaheurística de busca adequada para a utilização em problemas de otimização. Algoritmo genético é parte da classe de algoritmos evolucionários e viabilizam a otimização de problemas a partir da simulação de técnicas e conceitos presentes na evolução natural, tais como seleção, cruzamento e mutação. A seguir são apresentados os requisitos para a implementação de um algoritmo genético:

- Representações das possíveis soluções do problema no formato de um código genético;
- População inicial que contenha diversidade suficiente para permitir ao algoritmo combinar características e produzir novas soluções;
- Existência de um método para medir a qualidade de uma solução potencial;
- Um procedimento de combinação de soluções para gerar novos indivíduos na população;
- Um critério de escolha das soluções que permanecerão na população ou que serão descartados;
- Um procedimento para introduzir periodicamente alterações em algumas soluções da população. Desse modo mantém-se a diversidade da população e a possibilidade de se produzir soluções inovadoras para serem avaliadas pelo critério de seleção dos mais aptos.

Nos algoritmos genéticos, uma população de indivíduos, chamados também de cromossomos, é inicialmente gerada de forma aleatória. Em seguida, é realizada a avaliação de cada indivíduo por meio de uma função objetivo, que seleciona os mais aptos para serem submetidos aos manipuladores ou operadores genéticos, tais como: cruzamento e mutação, dando origem a uma nova geração de indivíduos. Cada cromossomo representa uma possível solução para o problema a ser otimizado. O algoritmo genético busca uma solução que seja muito boa, ou a melhor do problema analisado, através da criação de uma população de indivíduos cada vez mais aptos de acordo com a função objetivo.

O ciclo do algoritmo genético é apresentado na Figura 5. Inicialmente é criada a população de indivíduos (soluções candidatas), de forma aleatória, visando evitar possíveis vícios e que a população se espalhe ao máximo no espaço de busca. A população de um algoritmo genético é o conjunto de indivíduos que estão sendo cogitados como solução e que serão usados para criar o novo conjunto de indivíduos para análise. O tamanho da população pode afetar o desempenho global e a eficiência do algoritmo. Populações muito pequenas têm grandes chances de perder a diversidade necessária para convergir a uma boa solução, pois fornecem uma pequena cobertura do espaço de busca do problema. Entretanto, se a população tiver muitos indivíduos, o algoritmo poderá perder grande parte de sua eficiência pela demora em avaliar a função objetivo de todo o conjunto a cada iteração, além de ser necessário trabalhar com maiores recursos computacionais (Linden, 2008). Em seguida, a população é avaliada pela função objetivo e são geradas as pontuações para cada indivíduo. Sempre após a avaliação, o algoritmo testa se o critério de parada foi atingido.

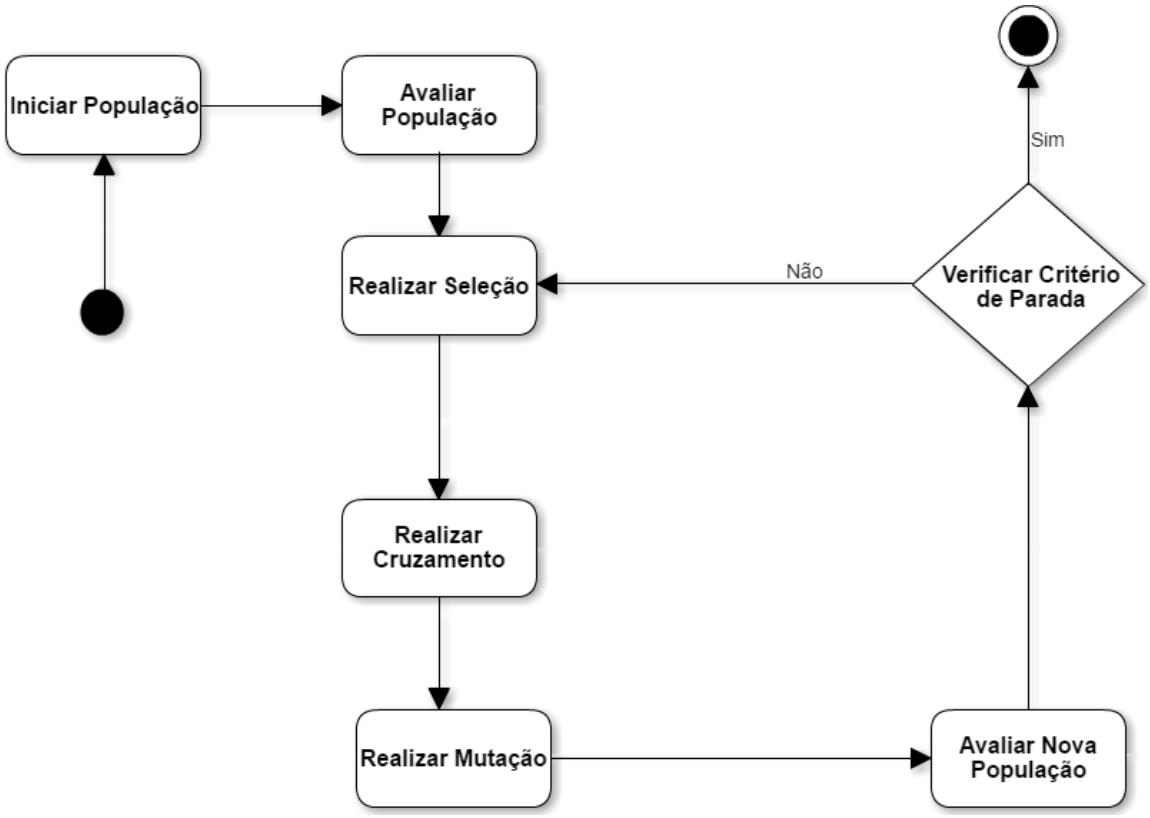

Figura 5 - Ciclo de um algoritmo genético

O passo seguinte consiste na seleção dos pais. Este passo simula o mecanismo de seleção natural, onde os mais fortes sobrevivem. Aqui os indivíduos mais aptos ou fortes são selecionados para gerar os filhos. Esse processo pode ocorrer de diversas formas, entre elas podemos citar:

- **Seleção por roleta**: apresentada por Holland (1975), usa o valor de *fitness* normalizado e disposto em uma "roleta", onde cada indivíduo da população ocupa uma "fatia" proporcional a sua aptidão normalizada. Em seguida é produzido um número aleatório entre 0 e 1. Este número representa a posição ocupada pela "agulha" da roleta. A probabilidade de cada indivíduo ser escolhido como pai nesse método de seleção é proporcional à adequação do indivíduo em relação à soma dos valores de todos os indivíduos da população. A escolha é feita aleatoriamente de acordo com essas probabilidades. Dessa forma são escolhidos como pais os mais bem adaptados, sem deixar de lado a diversidade dos menos adaptados.
- **Seleção por truncamento**: Uma população é iniciada, onde apenas os melhores *x%* da população poderão ser escolhidos como pais da próxima geração. O valor de *x* é um parâmetro do algoritmo, que pode variar entre 1% a 100%. Por exemplo, se $x = 40$, então

a seleção é feita entre os 40% melhores indivíduos e os outros 60 % são descartados. Os valores mais usuais para *x* são aqueles na faixa entre 10% e 50% (Linden, 2008). Para implementar este método, os indivíduos são ordenados de forma decrescente de acordo com sua avaliação e somente aqueles cujas posições estiverem entre 1 e a posição de corte poderão participar da seleção. A ordenação faz com que o algoritmo tenha uma complexidade mínima de tempo por rodada. Esse método causa uma convergência genética mais veloz e uma rápida perda da diversidade quando *x* é um valor pequeno.

- **Seleção por torneio**: nesta técnica, *N* indivíduos da população são escolhidos aleatoriamente. O objetivo dessa escolha é o não favorecimento a nenhum indivíduo, pois é realizada uma competição direta destes indivíduos pelo direito de ser pai, de acordo com a função objetivo de cada um. O melhor indivíduo desse torneio será selecionado para ser pai. A escolha aleatória dos indivíduos para o torneio é efetuada de modo que um indivíduo que já foi escolhido para concorrer ao torneio possa ser escolhido novamente. Como o pior indivíduo da população sempre perderá no torneio para qualquer outro indivíduo, então ele só poderá ser selecionado para ser pai se o torneio for formado apenas por esse pior indivíduo. Para o algoritmo não perder diversidade, é comum realizar o torneio por pares de indivíduos, para que os indivíduos não tão bons também tenham chance de ser pais.

Após a inicialização do algoritmo genético e seleção dos pais, é aplicado os operadores genéticos, são eles: cruzamento e mutação. O princípio básico dos operadores genéticos é transformar a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório. Os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores. Os operadores de cruzamento e de mutação têm um papel fundamental em um algoritmo genético.

O passo de cruzamento é o processo de recombinação de partes das sequências de genes entre pares de cromossomos, com o objetivo de gerar nova descendência. Esta troca de material genético garante a recombinação da população, possibilitando, assim, uma probabilidade maior de produzir indivíduos mais evoluídos que seus pais. Através do cruzamento são criados novos indivíduos misturando características de dois indivíduos pais. Esta mistura é feita tentando simular a reprodução de genes em células. Trechos das características de um indivíduo são trocados pelo trecho equivalente do outro. O resultado desta operação é um indivíduo que potencialmente combine as melhores características dos indivíduos usados como base. Alguns

tipos de cruzamento bastante utilizados são: cruzamento em um ponto de corte; cruzamento em dois pontos de corte; e cruzamento uniforme. Alguns são apresentados a seguir.

- **Cruzamento com um ponto de corte**: seleciona aleatoriamente um ponto de corte do cromossomo. Nesta técnica, dois pais de tamanho *n* são selecionados e submetidos ao processo de cruzamento, no qual um ponto de corte *p* é aleatoriamente gerado. O primeiro filho (*Filho 1*) recebe todos os genes do primeiro pai (*Pai 1*) de 1 até *p*, e todos os genes do segundo pai (*Pai 2*) de *p*+1 até *n*. O segundo filho (*Filho 2*) é gerado de forma inversa, recebendo primeiro todos os genes do segundo pai (*Pai 2*) de 1 até *p*, e todos os genes do primeiro pai (*Pai 1*) de *p*+1 até *n*. A Figura 6 exemplifica essa técnica de cruzamento.

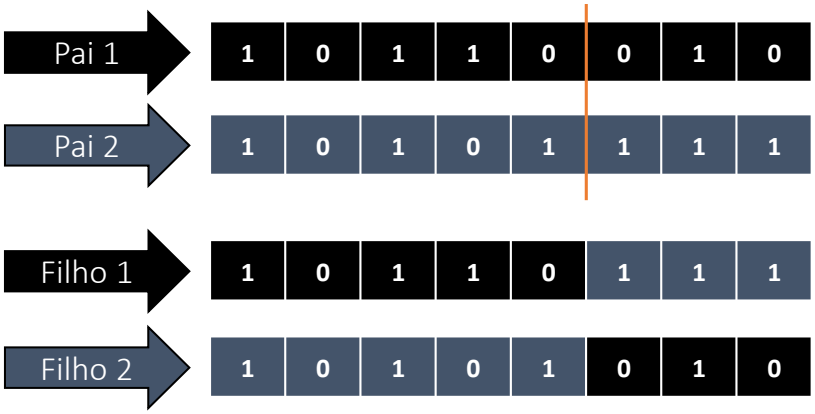

Figura 6 - Cruzamento com um ponto de corte

- **Cruzamento com dois pontos de corte**: segue a mesma ideia do cruzamento com um ponto corte, onde são selecionados dois pais de tamanho *n*, só que agora são gerados dois pontos de corte *p* e *q*. Os genes que estiverem entre (*1 e p*), e (*q e n*) sãocopiados do primeiro pai (*Pai 1*) para o primeiro filho (*Filho 1*), e, de forma análoga, do segundo pai (*Pai 2*) para o segundo filho (*Filho 2*).Já os genes situados entre (*p e q*) são copiados do primeiro pai (*Pai 1*) para o segundo filho (*Filho 2*), enquando os respectivos genes do segundo pai (*Pai 2*) são copiados para o primeiro filho (*Filho 1*). A Figura 7 exemplifica tal cruzamento.

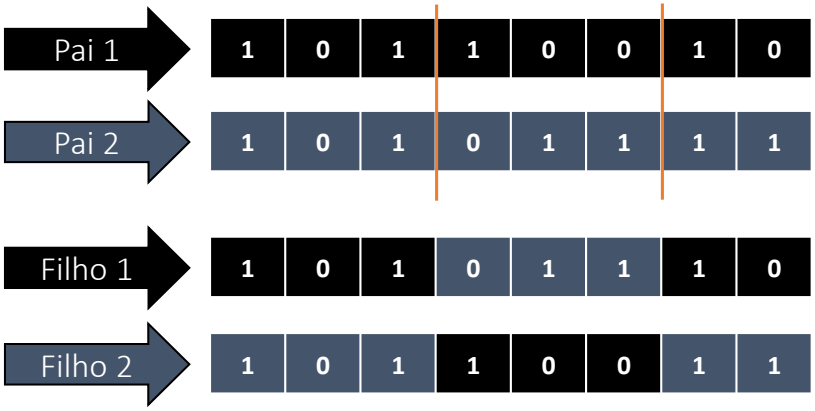

Figura 7 - Cruzamento com dois pontos de corte

- **Cruzamento uniforme**: cada gene do descendente é criado copiando o gene correspondente de um dos pais, escolhido de acordo com uma máscara de cruzamento gerada aleatoriamente. No primeiro filho (*Filho 1*), onde houver 1 na máscara de cruzamento, o gene correspondente será copiado do primeiro pai (*Pai 1*), e, onde houver 0, será copiado o gene do segundo pai (*Pai 2*). No segundo filho (*Filho 2*), o processo é repetido com os pais trocados para produzir o segundo descendente. Uma nova máscara de cruzamento é criada para cada par de pais. Um exemplo pode ser visto na Figura 8.

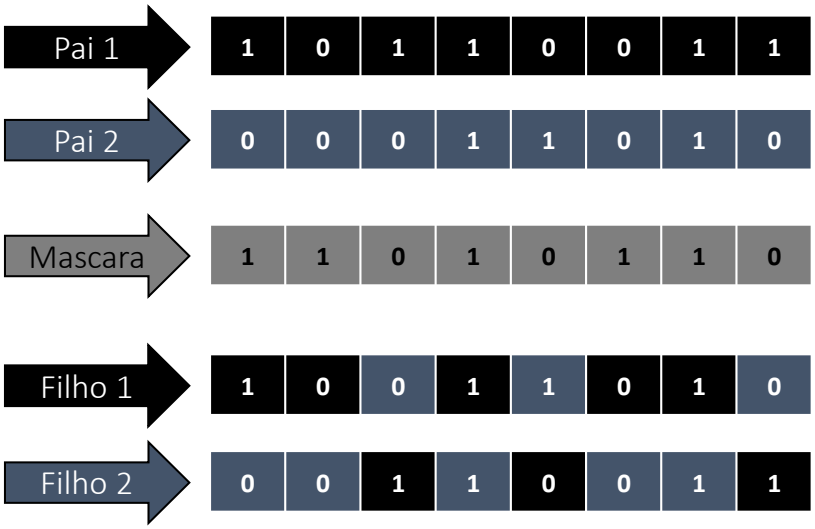

Figura 8 - Cruzamento Uniforme

Após o cruzamento, o passo seguinte é o operador de mutação, que é responsável pela introdução e manutenção da diversidade genética na população. A mutação opera sobre os indivíduos resultantes do processo de cruzamento, e com uma probabilidade predeterminada efetua algum tipo de alteração na estrutura do cromossomo. A importância deste operador reside no fato de que uma vez bem escolhido, seu modo de atuar garante que diversas alternativas

serão exploradas, mantendo um nível mínimo de abrangência na busca. Alguns exemplos do operador de mutação são representados pelas Figuras 9, 10 e 11, cujos métodos de mutação são descritos a seguir::

- **Mutação Flip**: cada gene a ser mutado recebe um valor sorteado do alfabeto válido.

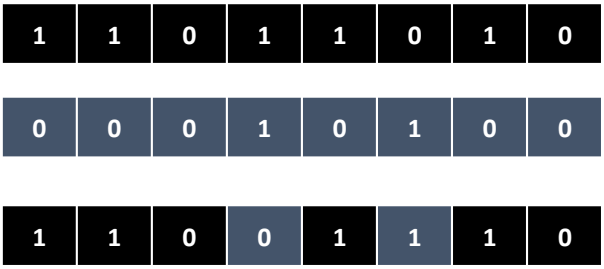

Figura 9 - Mutação Flip

- **Mutação Swap**: são sorteados *n* pares de genes, e os elementos de cada par trocam de valor entre si;

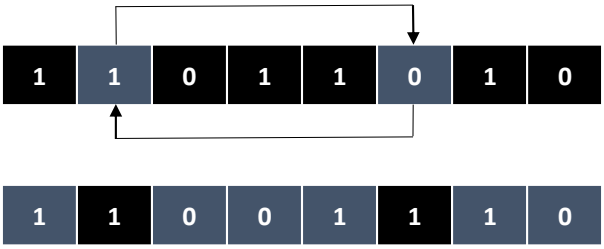

Figura 10 - Mutação Swap

- **Mutação Creep**: um valor aleatório é somado ou subtraído do valor do gene.

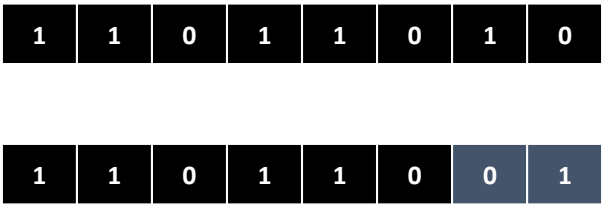

Figura 11 - Mutação Creep

Após a realização dos passos descritos anteriormente, uma nova população é gerada e deve ser avaliada, caso a condição de parada seja alcançada, o algoritmo se prepara para seu término e disponibiliza a população final. Caso contrário, a população deve passar iterativamente por todo o processo até que a condição de parada seja alcançada, e, assim, sejam encontradas algumas boas soluções, caracterizadas pela população final.

2.6. Considerações Finais

Este capítulo introduziu a revisão da literatura relativa às principais áreas de estudo envolvidas neste trabalho, tendo em vista que esse conhecimento se faz necessário para um melhor entendimento do trabalho. Arquitetura SOA com uso de serviços web implementados a partir de uma especificação WSDL e utilização do protocolo SOAP é a base teórica do problema abordado por este trabalho, que é a seleção de serviços web. Portanto, as definições de serviços web e seus elementos para desenvolver sistemas de software foram exploradas nessa seção. A fundamentação sobre SBSE, com ênfase na descrição da utilização de algoritmos de otimização como estratégia de busca, mais precisamente algoritmos genéticos, teve seu conteúdo bastante explorado, pois essa foi à técnica escolhida para resolver o problema de seleção automatizada de serviços web proposto por esse trabalho. Como a literatura é bastante ampla nessas áreas, o capítulo somente explorou os tópicos relevantes para o trabalho.

Capítulo 3

Trabalhos Relacionados

Este capítulo tem como objetivo apresentar alguns trabalhos que possuem relação com a abordagem proposta, evidenciando os pontos positivos e negativos de cada um deles e, principalmente, mostrando onde os trabalhos relacionados diferem deste trabalho de dissertação. Foram escolhidos trabalhos de seleção automatizada de serviços web que lidam com técnicas de otimização.

Seleção de serviços web é um campo de pesquisa fundamental em processos de desenvolvimento baseados em SOA. Como consequência, é possível encontrar algumas propostas na literatura (Hadjila et. al, 2011), (Lifeng et. al, 2008), (Maolin et. al, 2010), (Mojtaba et. al, 2014) e (Pegah et. al, 2012) adotando diferentes estratégias para a seleção de serviços web, a fim de reduzir o custo e o tempo de desenvolvimento de um sistema de software. Apesar de suas relevantes contribuições, em geral, tais propostas disponíveis lidam unicamente com critérios relacionados a requisitos não funcionais, mais especificamente os relacionados com a qualidade do serviço, ou atributos de QoS, incluindo tempo de resposta, preço, confiabilidade, disponibilidade, reputação e custo.

A falta de propostas que lidam diretamente com critérios relacionados aos requisitos funcionais, ou seja, a funcionalidade do serviço juntamente com propriedades estruturais, que é claramente a principal contribuição deste trabalho de dissertação, fez com que fossem buscados trabalhos que mais se assemelham com a abordagem proposta. A Tabela 2 mostra um comparativo entre os trabalhos relacionados.

Tabela 2 - Comparativo entre os trabalhos relacionados

		(Hadjilaet al., 2011)	(Lifeng et al., 2008)	(Maolin et al. 2010)	(Mojtaba et al. 2014)	(Pegah et al. 2012)	Abordagem Proposta
Tecnologias Utilizadas	WSDL	X	X		X		X
	SOAP	X	X		X		
	UDDI	X	X		X		
	Diagrama de Sequencia						X
Modelagem	BPEL	X		X	X		
	YAWL		X				
	Diagrama de Componentes						X
Métricas	Atributos de QoS	X	X	X	X	X	
	Funcional						X
	Estrutural			X			X
Requisitos Avaliados	Tempo de Execução	X	X	X	X	X	
	Disponibilidade	X	X	X	X	X	
	Preço	X	X	X	X	X	
	Reputação	X	X	X			
	Confiabilidade		X	X	X	X	
	Dependência entre Serviços		X				X
	Conflito entre Serviços		X				
	Integração entre Serviços			X			X
	Similaridade Funcional						X
Avaliação	Por Serviço	X	X	X		X	X
	Toda Arquitetura				X		X
Técnica de Seleção	Algoritmo Genético	X	X	X			X
	Método VIKOR				X		
	Busca Harmônica					X	

No artigo de Hadjila et, al. (2011), é apresentado um sistema de seleção de serviço que otimiza a composição de serviços web agregada a requisitos não funcionais com base em QoS minimizando as restrições por parte do usuário. Como exemplos de restrições do usuário, os autores citam o custo de utilizar determinado serviço, o tempo de resposta do serviço selecionado, e a disponibilidade de acesso que este serviço possui. Para realizar a tarefa de selecionar serviços web de forma automatizada, o trabalho adota uma solução com base em metaheurística, mais precisamente Algoritmo Genético. Os autores argumentam que a escolha de tal algoritmo se dá pela aptidão que esta metaheurística possui com grandes espaços de

busca. O trabalho tem o objetivo de otimizar a composição de serviços web com base no tempo de resposta, disponibilidade, preço e reputação do serviço. Alguns atributos são qualificados pelos usuários do serviço, algo que não garante uma informação real para o atributo de um determinado serviço, pois depende da disponibilidade do utilizador para realizar a avaliação. A dificuldade em elaborar métricas para medir tais atributos, é sem dúvida uma fragilidade no trabalho.

Uma abordagem para o problema de seleção e composição de serviços web com base em methaeurística é proposta por Lifeng et, al. (2008). O trabalho adota algoritmo genético como metaheurística, lidando com restrições que são derivadas das depêndencias e conflitos existentes entre as implementações dos serviços web. Com isso, a pesquisa se concentra em como selecionar uma aplicação de serviço web para cada uma das tarefas na especificação abstrata, para que as restrições encontradas na composição sejam satisfatórias ou até mesmo a melhor sobre as dependências e conflitos que podem existir entre os serviços web. Na pesquisa os autores consideram alguns atributos de QoS, são eles: tempo de resposta, preço, reputação, disponibilidade e confiabilidade. Do mesmo modo que no trabalho de Hadjila et, al. (2011), os atributos de QoS definidos podem ter sua forma de medição facilmente questionada, pois para medir o tempo de resposta de um serviço, seria necessário que a conexão de internet de todos os utilizadores fosse uniforme, algo que não acontece no mundo real. Outro atributo que pode ter sua métrica questionada, é a confiabilidade de um serviço. Ambos os trabalhos não deixam claro como realizar tais medições.

No que diz respeito a métrica de dependência entre os serviços, Lifeng et, al. (2008) diz que ao selecionar um serviço web que seja dependente de outro, esse segundo deve obrigatóriamente ser selecionado e não passará pelo filtro da metaheurística, algo que difere da abordagem proposta, que ao selecionar um serviço que possua alguma dependencia, é selecionado um outro que além de suprir a necessidade do primeiro, também atenda funcionalidades que foram especificadas. É importante resaltar que os autores não informam como as dependências do serviço são identificadas, algo que não ocorre com o trabalho proposto que utiliza diagramas de sequencia associado as operações para identificar as dependencias do serviço. Já a métrica que trata os conflitos que podem existir entre dois serviços selecionados é bastante útil para identificar quais serviços não podem se relacionar.

O trabalho de Maolin et al. (2010) tem o objetivo de selecionar uma implementação para cada serviço web com base em atributos de qualidade afim de realizar uma composição de serviços web. Para realizar a atividade de selecionar serviços os autores apresentam um

Algoritmo Genético híbrido baseado em restrição de dependência e conflito entre as implementações dos serviços. Assim como os demais trabalhos relacionados, aqui também é adotado atributos de QoS para medir a qualidade do serviço selecionado para a integração. Como expressado anteriormente, a forma como ocorre a medição de tais atributos pode ser facilmente questionada, pois não depende exclusivamente do serviço web, e sim de outros atores envolvidos no processo de selecionar e utilizar um serviço.

Assim como no artigo de Lifeng et al. (2008) e nesse trabalho de dissertaçao, Maolin et al. (2010) utiliza métricas para medir as dependências existentes entre os serviços, e as possíveis incompatibilidades que podem existir ao se integrar dois serviços web. Diferente do trabalho proposto, que ao identificar a dependência do serviço, outro serviço é submetido a análise das métricas a fim de ser selecionado, Maolin et al. (2010) afirma que caso uma implementação seja selecionada e possua dependência de outra, essa segunda também deve ser selecionada. Da mesma forma que Lifeng et al. (2008), Maolin et al. (2010) não deixa claro como as dependências do serviço são identificadas. Algo que o trabalho de dissertação se sobressai muito bem com a utilização de diagramas de sequência. Referente a integração entre os serviços, Maolin et al. (2010) mede apenas a qualidade na integração entre pares de serviços selecionados, já o trabalho proposto avalia além da integragração entre dois serviços, avalia também toda a arquitetura formada pelos serviços selecionados.

Outro trabalho relacionado escolhido, é o apresentado por Mojtaba et al. (2014). Aqui, é adotado um método com base em MCDC (*Multi-Criteria Decision Making*) como técnica para otimizar e realizar a seleção de serviços web. Da mesma forma que os demais trabalhos relacionados, este também adota atributos com base em QoS, mais precisamente: tempo de execução, disponibilidade, confiabilidade e custo. Os autores afirmam que os atributos de QoS adotados são pontuados de acordo com as preferências dos consumidores do serviço. Diferente da abordagem proposta e dos trabalhos relacionados apresentados anteriormente, o trabalho de Mojtaba et al. (2014) adota o método VIKOR na seleção de serviços. Este método é considerado um processo para tomada de decisão com base em multiplos critérios. Originalmente desenvolvido para resolver problemas de decisão em ambientes conflitantes, onde o tomador de decisão quer uma solução que seja a mais próxima da ótima.

Um fator que desperta atenção no trabalho de Mojtaba et al. (2014), é que a avaliação com base nos atributos de QoS ocorre apenas sobre a configuração arquitetural, ou seja, depois que as implementações correspondentes a cada especificação são selecionadas possibilitando a composição de serviços. Os autores afirmam que a busca e seleção de um serviço ocorre por

meio da comparação linguística da descrição da especificação de um serviço, com a descrição da implementação de um serviço candidato.

Outro ponto a ser comparado entre os trabalhos relacionados e a abordagem proposta, é a técnica utilizada para selecionar os serviços web. A técnica empregada por Mojtaba et al. (2014) se assemelha com a utilizada por esse trabalho de dissertação e demais trabalhos relacionados, em que ambas buscam encontrar soluções próximas a solução ideal, no entanto, o método VIKOR apresenta um resultado com apenas uma única solução, enquanto algoritmo genético apresenta um conjunto de boas soluções para o tomador de decisão.

Por fim, o trabalho de Pegah et al. (2012), propõe uma seleção de serviços web para obter uma composição de serviços próxima da ideal com base em um algoritmo metaheurístico. A técnica utilizada é denominada algoritmo de busca harmônica. Tal técnica é classificada como um algoritmo de otimização metaheurístico desenvolvido recentemente e sem derivações, que se inspira no processo musical em busca de um perfeito estado de harmonia. Assim como a grande maioria dos trabalhos que abordam a seleção de serviços web, o trabalho de Pegah et al. (2012) não é diferente, e adota requisitos não funcionais, mas especificamente voltados a atributos de qualidade do serviço (QoS) como métricas na etapa de seleção, tais como: preço do serviço, tempo de resposta, disponibilidade e confiabilidade. Na pesquisa, os autores tratam o problema de seleção de serviços web baseado em um QoS global, cujo objetivo é maximizar o valor de QoS do serviço web composto por meio da qualidade individual de cada serviço pertecente na composição.

Algo que difere os demais trabalhos relacionados e a abordagem proposta do trabalho de Pegah et al. (2012), é que os primeiros utilizam algoritmo genético como técnica de otimização, enquanto este último adota uma nova metaheurística denominada de busca harmônica. Essa é uma contribuição bastante interresante, pois gera uma maior diversidade nos trabalhos que propõe selecionar serviços web de forma automatizada. No entanto, as métricas utilizadas nos trabalhos relacionados, e consequentemente encontrados, lidam apenas com requisitos não funcionais, mais especificamente voltados a QoS. Algo que difere bastante da abordagem proposta que adota requisitos funcionais dos serviços, avaliando a similaridade dos serviços especificados com as respesctivas implementações, como também uma avaliação de quão boa é a integração entre os serviços selecionados, acarretando uma maior qualidade de toda configural arquitetural.

3.1 Considerações Finais

Este cápitulo apresentou diversos esforços propostos na area de seleção de serviços web. Embora algumas das soluções apresentadas compartilhem caracteristicas com o trabalho proposto, a vasta maioria dos trabalhos adotam apenas requisitos não funcionais como métricas para selecionar serviços, e nenhuma das abordagens descritas atendem de forma concisa a seleção de serviços web com base em requisitos funcionais, conforme apresentado na Tabela 2. Diferente dos trabalhos relacionados, o trabalho proposto nessa dissertação adota métrica baseada em similaridade funcional avaliando o quão similiar é o serviço candidato com o que foi especificado, além de uma métrica estrutural avaliando o quão bom é a qualidade da conexão entre pares de serviços, como também todos os serviços pertecentes a solução arquitetural. Sem dúvida essas formas de avaliar os serviços para seleção é a principal contribuição da abordagem aqui proposta.

Capítulo 4

Abordagem Proposta

O trabalho aqui proposto visa automatizar o processo de seleção de serviços web utilizando técnicas da Engenharia de Software Baseada em Buscas, cuja metaheurística é orientada por métricas funcionais que avaliam o grau de similaridade entre as especificações dos serviços web e suas respectivas implementações candidatas no que diz respeito a requisitos funcionais, e métricas estruturais que avaliam as dependências entre serviços web de uma arquitetura de software orientada a serviços. Com isso, espera-se que a seleção automatizada de serviços web seja capaz de escolher uma configuração arquitetural satisfatória baseada no projeto de arquitetura do software, que é elaborado pelo arquiteto de software. Este processo visa gerar uma solução com utilização de serviços web com o mínimo de incompatibilidades funcionais e estruturais possíveis, podendo identificar e evitar as inconsistências entre os serviços web antes que o sistema de software seja construído, e, consequentemente, minimizar o custo e o esforço para adaptação e integração das respectivas implementações selecionadas.

A abordagem de *seleção automatizada de serviços web baseada em métricas funcionais e estruturais* terá seus aspectos fundamentais detalhados neste capítulo. De início será apresentada uma visão geral do trabalho com o intuito do leitor perceber a estrutura da abordagem proposta, assim como o escopo. Em seguida serão apresentadas as etapas da abordagem proposta, detalhando a importância de cada uma neste trabalho, evidenciando seus objetivos, propriedades, e benefícios.

4.1. Visão Geral

Considerando um processo de desenvolvimento de software que adota o paradigma SOA, a etapa que antecede o processo de *Seleção de Serviços Web* é a identificação das funcionalidades e dependências dos serviços web, que recebe como artefato de entrada a descrição do projeto arquitetural (Dias, 2010). O projeto arquitetural está preocupado com a compreensão de como um sistema deve ser organizado e também com a estrutura geral desse sistema. Em um processo de desenvolvimento de software, o projeto de arquitetura é o primeiro estágio do projeto de software, sendo o elo crítico entre o projeto e a engenharia de requisitos, pois identifica os

principais componentes estruturais do sistema e os relacionamentos entre eles. O resultado do projeto de arquitetura é um modelo que descreve a tecnologia a ser utilizada, no caso deste trabalho *Web Service*, e, como o sistema está organizado em um conjunto de componentes.

A Figura 12 ilustra as etapas que compõe a abordagem proposta. Como pode ser observado, a primeira etapa é a *Identificação das Funcionalidades e Dependências*. Nesta etapa as funcionalidades dos serviços web são identificadas via especificações WSDL através do atributo *PortType*, e as dependências dos serviços web são identificadas via diagramas de sequência, por meio da troca de mensagens entre os serviços. A etapa seguinte é a *Seleção de Serviços Web*, que representa o núcleo da abordagem proposta na qual os serviços web candidatos são avaliados e, em seguida, selecionados para compor configurações arquiteturais ideais a fim de reduzir os esforços de adaptação para a integração dos serviços web. Esta etapa adota algoritmo genético como técnica de busca para realizar a seleção com base na similaridade entre a implementação do serviço candidato e o serviço web especificado, e também considerando as dependências que os serviços podem ter entre si. Vale ressaltar que várias configurações arquiteturais podem ser recomendadas, permitindo que a equipe de desenvolvimento de software possa escolher aquela que melhor satisfaz as necessidades do projeto e da organização. Depois de selecionar os serviços web, na terceira fase, chamada de *Integração do Sistema de Software*, a equipe de desenvolvimento de software pode integrar e adaptar o conjunto de serviços web incluídos na configuração arquitetural selecionada.

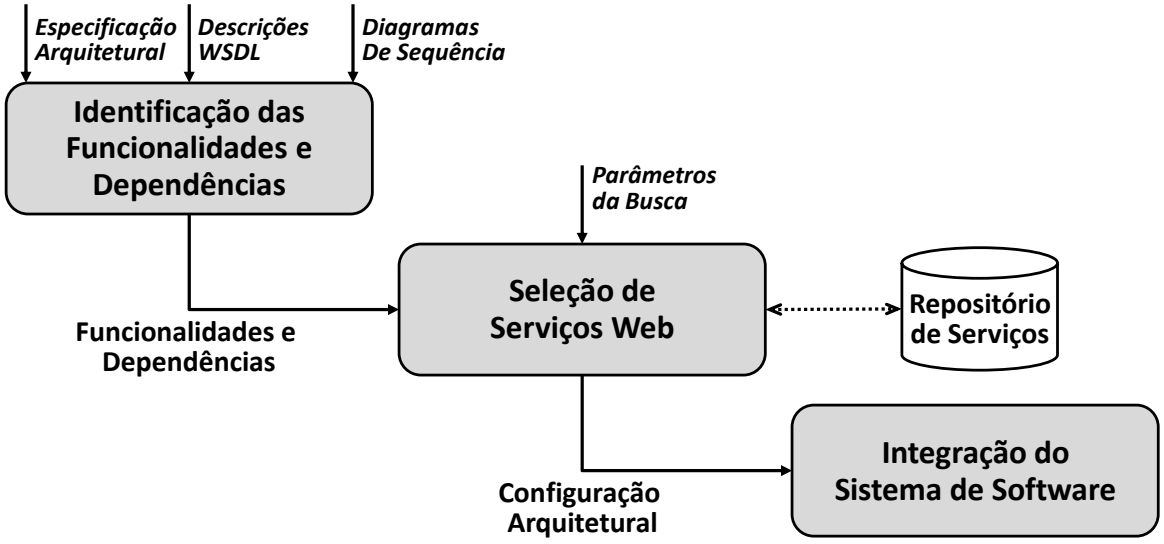

Figura 12 - Visão geral da abordagem proposta

Este trabalho tem foco nas duas primeiras fases descritas pela Figura 12. Devido a isso, a próxima subseção explica o processo de identificação das funcionalidades e dependências dos

serviços web, e, em seguida, é apresentado uma subseção descrevendo a etapa da seleção de serviços web, evidenciando sua importância na construção de um sistema de software em uma Arquitetura Orientada a Serviços, além de realizar um detalhamento das métricas utilizadas e da metaheurística adotada para realizar a seleção de serviços.

4.2. Identificar Funcionalidades e Dependências dos Serviços Web

Com o uso da tecnologia de w*eb services*, a especificação das interfaces dos serviços é realizada usando WSDL. Entretanto, a WSDL oferece apenas a identificação das funcionalidades providas pelo serviço web, ou seja, as operações que são executadas pelo serviço, restando ainda à identificação das dependências que os serviços web podem ter uns com os outros. Com o objetivo de extrair a informação de dependências de um serviço, já que essa informação não se encontra disponível na WSDL, são explorados os diagramas de sequência associados às operações providas pelos serviços. Em processos de desenvolvimento baseados em serviços, um diagrama de sequência pode ser utilizado para representar a sequência de execução de cada operação, incluindo a troca de mensagens requeridas com outros serviços web (Dias, 2010). Entende-se por mensagens as funcionalidades solicitadas ao serviço, como também as respostas das solicitações. Logo, são modelados diagramas de sequência para cada operação provida do serviço selecionado, com o objetivo de descrever a maneira como os serviços web colaboram (se conectam) entre si, registrando o comportamento dos serviços na troca de mensagens entre eles.

A Figura 13 apresenta um exemplo de especificação WSDL, que adota a linguagem XML. A especificação WSDL apresentada contém os principais atributos utilizados neste tipo de artefato, são eles: *definitions*, *types*, *menssage*, *portType*, *port*, *binding*, *operation* e *service*. A seguir serão descritos tais elementos:

- ***Definitions***: é o elemento raiz de qualquer documento WSDL. Ele contém atributos que servem para definir os *namespaces* utilizados no documento WSDL. Esse elemento e sua sintaxe podem ser visualizados na linha 2 da Figura 13.
- ***Types***: atua como um contêiner para definições de tipos de dados que estão presentes na mensagem. Exemplos deste elemento são descritos entre as linhas 5 a 18 da Figura 13.
- ***Message***: definição abstrata de dados que devem ser usados na comunicação com o serviço. Cada elemento *message* recebe um ou mais elementos *part* que formam as partes reais da mensagem. Veja exemplo entre as linhas 19 a 24 da Figura 13.

- ***PortType***: possui um conjunto de elementos *operation*, que definem as operações que podem ser executadas e as mensagens trocadas para executá-las. O elemento *PortType* está representado entre as linhas 25 a 30 da Figura 13.
- ***Operation***: descrição abstrata de uma ação suportada pelo serviço; possui um atributo *name* e um atributo opcional para especificar a ordem dos parâmetros usados nas operações. Existem quatro diferentes tipos de mensagens, são eles:
    - **Operação Unidirecional**: define uma mensagem enviada de um cliente para um serviço, e não obtém resposta;
    - **Solicitação-Resposta:** esse é o mais utilizado, permitindo que um cliente envie uma solicitação a um serviço e receba como resposta uma mensagem com o resultado da solicitação.
    - **Pedido-Resposta**: não é muito utilizada. Esse tipo define uma mensagem enviada do serviço para o cliente, resultando em uma mensagem enviada do cliente de volta para o serviço.
    - **Notificação:** o tipo de operação mais simples, onde um serviço apenas envia uma mensagem para o cliente.
- ***Binding***: mapeia os elementos *operation* de um elemento *portType* para um protocolo de transporte específico, tal como SOAP. Ele utiliza-se de um elemento de extensão SOAP chamado <*soap:binding*>, através de dois parâmetros: protocolo de transporte e o estilo da requisição, como por exemplo *document.* O elemento *binding* é exemplificado na Figura 13 entre as linhas 31 a 38.
- ***Service e Port***: definem a localização real do serviço tendo em vista que o mesmo pode conter várias portas e cada uma delas é especificada para um tipo de ligação, descrita no elemento *binding*. Estes elementos podem ser encontrados na Figura 13 entre as linhas 39 a 44.

```xml
<?xml version="1.0"?>
<definitios name="StockQuote" targetNamespace="http://example.com/stockquote.wsdl"
    xmlsn:tns="http://example.com/stockquote.wsdl" xmlns:xsd1="http://example.com/stockquote.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
<schema targetNamespace="http://example.com/stockquote.xsd" xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="PacoteTuristicoRequest">
        <complexType>
            <all> <element name="dataDestino" type="String" /> </all>
        </complexType>
    </element>
    <element name="PacoteTuristicoPreco">
        <complexType>
            <all> <element name="preco" type="float" /> </all>
        </complexType>
    </element>
    </schema>
</types>
<message name="GetPacoteTuristicoInput">
    <part name="body" element="xsd1:PacoteTuristicoRequest" />
</message>
<message name="GetPacoteTuristicoOutput">
    <part name="body" element="xsd1:PacoteTuristicoPreco" />
</message>
<portType name="StockQuotePortType">
    <operation name="GetPacoteTuristico">
        <input message="tns:GetPacoteTuristicoInput" />
        <output message="tns:GetPacoteTuristicoOutput" />
    </operation>
</portType>
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="GetPacoteTuristico">
        <soap:operation soapAction="http://example.com/GetPacoteTuristico" />
        <input> <soap:body use="literal" /> </input>
        <output> <soap:body use="literal" /> </output>
    </operation>
</binding>
<service name="StockQuoteService">
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote" />
    </port>
</service>
</definitios>
```

Figura 13 - Exemplo de especificação WSDL

O exemplo da WSDL apresentada na Figura 13 é a especificação de um serviço simples para consultar preços de pacotes turísticos. Esse serviço suporta uma única operação denominada "*GetPacoteTuristico*". Quando for feita uma requisição ao serviço, deve ser

passada a data e destino por meio do atributo "*dataDestino*", que deve ser do tipo *String*. Como resultado, a operação retorna o preço por meio do atributo "*preco*", que deve ser do tipo *float*.

Com base na especificação WSDL ilustrada na Figura 13, é possível identificar que o serviço web permite realizar uma consulta por preços de pacotes turísticos. Entretanto, para que o serviço possa obter as informações completas sobre os preços de pacotes turísticos, outros serviços devem ser invocados, tais como: consulta de passagens aéreas, consulta de reservas de hotéis, consulta de atrações turísticas, entre outras. Portanto, para que o serviço "*GetPacoteTuristico*" possa apresentar todas as informações pertinentes a um pacote turístico, ele depende e se conecta com outros serviços que oferecem as informações de consulta de passagens aéreas, reservas de hotéis, etc.

Como descrito previamente, diagramas de sequência podem ser utilizados para identificar possíveis dependências que serviços web possuem de outros, por isso, a abordagem proposta adota tais diagramas associados a cada operação provida por cada serviço, com o objetivo de identificar prováveis dependências que possam existir. A partir desse ponto, as dependências de um serviço são representadas e tratadas na abordagem proposta como interfaces requeridas de um serviço web. A Figura 14 mostra um diagrama de sequência envolvendo o serviço de consulta de pacotes turísticos, denominado "*ViagemSrv*", bem como sua dependência em relação ao serviço de reserva de passagem aérea, denominado "*VoarSrv*".

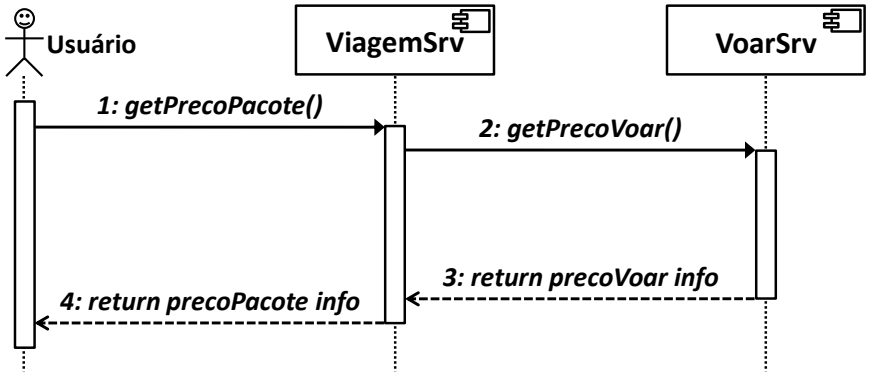

Figura 14 - Diagrama de sequência com a dependência entre serviços

Como pode ser observado na Figura 14, o serviço web que provê a funcionalidade para consulta de preços de viagens realiza uma consulta a um outro serviço com o objetivo dedeterminar o preço da passagem aérea. Com isso é possível afirmar que o serviço "*ViagemSrv*" provê a funcionalidade de consultar preço de pacotes turísticos, e que o mesmo requer uma consulta ao serviço "*VoarSrv"* para obter o preço da passagem aérea.

Para facilitar o tratamento e o entendimento das operações providas, como também das dependências entre serviços web, a abordagem proposta adota diagramas de componentes UML para representar as funcionalidades providas pelos serviços via interfaces providas e as dependências entre serviços via associações (conexões) entre interfaces requeridas pelos serviços requisitantes e as interfaces providas pelos serviços provedores. Na Figura 15 é mostrado um exemplo de descrição arquitetural usando diagramas de componentes da UML, sendo os serviços web representados por especificações de componentes e as conexões por especificações de interfaces providas e requeridas. Esse diagrama é utilizado para representar os serviços web que compõem a arquitetura do sistema de software que foi idealizado e especificado no projeto arquitetural. Com as informações obtidas nas descrições WSDL, juntamente com as informações extraídas dos diagramas de sequência, é possível gerar uma especificação como a apresentada pela Figura 15, e, com isso, visualizar de forma clara todos os serviços web participantes com suas respectivas funcionalidades e dependências.

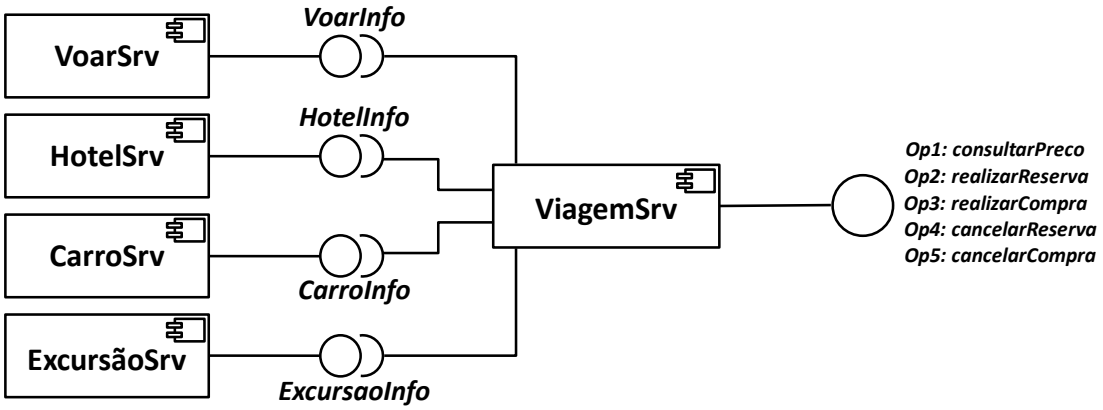

Figura 15 - Mapeamento de SOA para arquitetura baseada em componentes

O diagrama de projeto arquitetural baseado em componentes apresenta as especificações de serviços web, funcionalidades e dependências, onde todas as funcionalidades são renomeadas como interfaces providas, e todas as dependências pertencentes a uma especificação são caracterizadas como interfaces requeridas. Essas dependências são parte da definição da especificação do serviço web, e devem ser tratadas por todas as implementações do serviço web.

Após a identificação dos serviços web, de suas funcionalidades providas, e as dependências, é possível derivar as métricas funcionais e estruturais definidas na abordagem proposta, e, em seguida, realizar o processo de seleção de serviços web.

4.3. Definição das Métricas

O arquiteto de software idealiza um sistema a ser desenvolvido com base em uma arquitetura construída por serviços abstratos, ou seja, apenas com a especificação dos serviços web. Com isso espera-se que as implementações selecionadas para construir o sistema de software sejam equivalentes às especificações presentes na arquitetura. Porém, como os serviços web são desenvolvidos por diferentes fornecedores, normalmente são encontradas incompatibilidades para a montagem dos sistemas, requerendo esforço para integração e adaptação dos serviços web candidatos.

Neste contexto, para reduzir o esforço de integração e adaptação, a abordagem aqui proposta realiza a avaliação dos serviços web candidatos em duas dimensões, a saber: (*i*) *estrutural* - avaliação de quão bom é a integração entre a interface provida de um serviço com a interface requerida de outro serviço; (*ii*) *funcional* - avaliação da similaridade da implementação candidata em relação ao serviço web especificado.

Com o objetivo de dar suporte ao processo de seleção de serviços web, principalmente no que diz respeito a integração entre os serviços, foram criadas métricas estruturais para avaliar tal integração. Esse tipo de métrica avalia a configuração arquitetural do sistema, visto que a mesma é construída pela integração entre os serviços web. As integrações entre os serviços caracterizam suas dependências e determinam como estão arranjados na arquitetura. A função básica de comunicação de um serviço web com o outro se dá por meio da troca de mensagens, cujas especificações permitem extrair as interfaces providas e requeridas dos respectivos serviços como já discutido anteriormente na seção 4.2.

Numa integração entre serviços web, o ideal é que todas as operações requeridas pela interface de um serviço, sejam providas por outra interface de um outro serviço, porém, existe a possibilidade de integrações entre serviços web, onde a interface provida por um, não é exatamente igual a interface requerida por outro. Neste caso, apesar das interfaces envolvidas serem similares, a interface provida pode ter mais ou menos operações em sua especificação, como também a interface requerida, gerando inconsistências ou incompatibilidades que requerem esforços de adaptação para integração das respectivas implementações dos serviços selecionados.

Por exemplo, considere uma especificação de um serviço *S* que provê uma interface com *N* operações. Suponha também que *Si* é uma implementação similar a *S*, e por isso foi selecionada, porém, provê uma interface com *N*-1 operações. Com isso, temos uma

implementação candidata *Si* que provê uma interface com menos operações que a interface especificada pelo serviço *S*. Consequentemente, a especificação da interface de *S* proporcionará operações que não são implementadas por *Si*. O inverso dessa situação é quando a interface implementada em *Si* possui operações que não são especificadas por *S*, ou seja, a implementação *Si* prover operações adicionais, e que a priori não serão necessárias para a construção do sistema de software. Outro fato que pode ocorrer, é a interface implementada em *Si* possuir ambas as situações citadas acima, ocorrendo a redução das operações especificadas, e, ao mesmo tempo, a inclusão de operações que não foram especificadas por *S*.

Para avaliar a conexão entre dois serviços web candidatos deve-se analisar o quão bom é a integração entre eles, por meio das operações das interfaces providas por um serviço e as interfaces requeridas por outro serviço, e com isso, analisar se a qualidade dessa integração corresponde às especificações do serviço web proposto. A Tabela 3 apresenta os termos usados para identificar as interfaces providas e requeridas pelas especificações e implementações dos serviços web, o que facilita o entendimento das figuras a seguir, como também das equações para derivação das métricas.

Tabela 3 - Identificadores dos conjuntos de operações

Identificador	Descrição
RS_k	Interface Requerida da Especificação do Serviço Web
PS_j	Interface Provida da Especificação do Serviço Web
RI_k	Interface Requerida da Implementação do Serviço Web Candidato
PI_j	Interface Provida da Implementação do Serviço Web Candidato

A Figura 16 retrata a perspectiva de avaliação de uma integração entre serviços web candidatos. Tem-se duas especificações de serviços relacionados por suas interfaces providas e requeridas, representados por meio de diagramas de componentes, juntamente com as respectivas implementações candidatas, também relacionadas por suas interfaces providas e requeridas. Portanto, nesse cenário, temos quatro interfaces participantes. Como no diagrama a integração é formada através de interfaces, os elementos específicos para a avaliação são: interfaces providas e interfaces requeridas. Observe que temos as interfaces providas e requeridas tanto das especificações dos serviços quanto das respectivas implementações candidatas. Com isso, quanto maior for o número de operações pertencentes à interseção entre os serviços, melhor será a integração entre os serviços web candidatos.

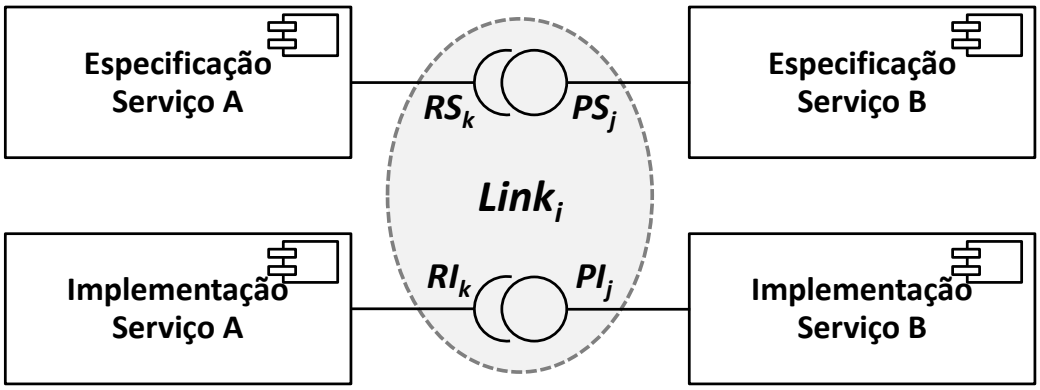

Figura 16 - Avaliação de integração entre serviços web candidatos

Cada conexão entre serviços web é avaliada conforme indicado na Equação 01, onde o numerador da divisão corresponde ao número de operações que pertencem à interseção entre todos os conjuntos envolvidos na avaliação. Tal interseção demonstra a conformidade simultânea entre a especificação e a implementação do serviço web. Já o denominador representa a união entre as operações pertencentes as interfaces requeridas na especificação e implementação. Conforme as propriedades de conjuntos, o número de elementos da interseção é menor ou igual ao da união. Também se tem o fato que o intervalo do numerador é (0, N) e o do denominador é (1, N), resultando na equação L_i possuir valores numéricos pertencentes ao intervalo (0, 1). Quanto mais próximo de 1 for o resultado, melhor será a integração, e, consequentemente, menor o esforço de adaptação dos serviços web candidatos.

$$L_i = \frac{|(RS_i \cap PS_i) \cap (RI_i \cap PI_i)|}{|(RS_i \cup RI_i)|} \quad (01)$$

Tal como pode ser observado na Equação 01, o denominador inclui apenas operações das interfaces requeridas. A razão para isso é a premissa adotada na abordagem proposta, que afirma o seguinte: operações supérfluas em interfaces providas não representam esforço de adaptação. Em outras palavras, operações ofertadas pelo prestador do serviço que não são utilizadas, não impõem esforços de adaptação no serviço solicitante.

Todas as conexões entre as implementações de serviços web candidatos são avaliadas e cada valor resultante dessa avaliação deve contribuir para gerar um único valor que represente a qualidade da estrutura da solução. Para tal, a Equação 02 considera a soma do valor de cada avaliação da Equação 01, dividida pelo número de conexões *L*, visando normalizar a avaliação estrutural da solução no intervalo (0, 1).

$$A_x = \frac{\sum_{i=1}^{L} L_i}{L} \quad (02)$$

Na Equação 02, o termo A_x avalia toda a estrutura da solução formada por implementações de serviços web candidatos, sendo representada pela relação entre o somatório das avaliações de cada conexão arquitetural e o número de conexões desta arquitetura. O número de conexões é representado pelo termo L, e o termo L_i representa a avaliação de uma conexão, calculado de acordo com a Equação 01.

Como as funcionalidades dos serviços web são descritas via interfaces, os requisitos funcionais podem ser representados por essas entidades. A arquitetura do software é construída a partir de elementos que modularizam a representação do sistema de software. Esses elementos são as especificações de serviços web, as quais possuem interfaces providas e requeridas. Com isso, uma implementação de um serviço candidato atende perfeitamente às funcionalidades especificadas se suas interfaces forem equivalentes. Partindo desse pressuposto, quanto mais similar o serviço candidato for em relação ao especificado, mais compatível será com as funcionalidades idealizadas para o sistema de software. A Figura 17 mostra a similaridade entre as especificações dos serviços web e suas respectivas implementações.

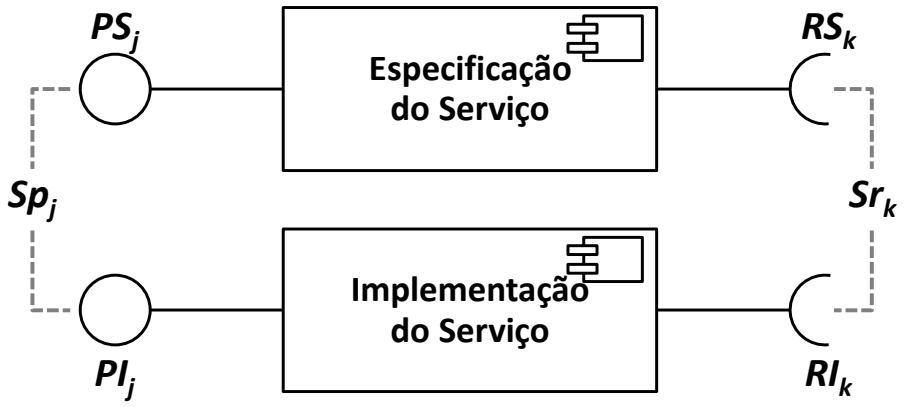

Figura 17 - Comparação por similaridade

Como as interfaces correspondentes entre a implementação do serviço e a especificação não são idênticas, há também a comparação em relação às operações dessas interfaces. A medição da similaridade entre a interface da especificação e a interface da implementação candidata é formada pelo cálculo do índice de Jaccard (Alexandre, 2013), que foi

matematicamente provado como sendo uma expressão para avaliar a similaridade entre conjuntos. Observe na equação abaixo.

$$S\ (C1, C2) = \frac{|C1 \cap C2|}{|C1 \cup C2|} \quad (03)$$

A Equação 03 tem resultado no intervalo (0, 1). À medida que os conjuntos *C1* e *C2* tendem a equivalência, a similaridade $S(C1, C2)$ tende ao valor numérico 1. Em contrapartida, na mesma proporção que um elemento possui mais ou menos elementos que o outro e também elementos distintos, o valor numérico dessa equação tende ao valor 0. Essa equação possui boa adequação com a medição de similaridade entre interfaces, pois podem ser consideradas conjuntos de funcionalidades.

Para calcular a similaridade de interfaces providas Sp_j, a abordagem proposta considera a relação cujo numerador é o número de operações da interseção entre as interfaces providas na especificação e implementação , e denominador é o número de operações da interface provida na especificação como denominador. É importante frisar mais uma vez que a abordagem proposta pressupõe que as operações supérfluas das interfaces providas não representam esforço de adaptação, por isso, o denominador da Equação 04 não considera as operações da interface provida na implementação do serviço.

$$Sp_j = \frac{|PS_j \cap PI_j|}{|PS_j|} \quad (04)$$

Na Equação 05, é obtido o nível de similaridade entre as interfaces requeridas Sr_k , onde o número de operações do numerador é formado pela interseção entre as interfaces requeridas na especificação e implementação, e o denominador é composto pelo número de operações resultante da união entre as interfaces requeridas na especificação e implementação.

$$Sr_k = \frac{|RS_k \cap RI_k|}{|RS_k \cup RI_k|} \quad (05)$$

Realizando o cálculo da similaridade de cada interface do serviço web, têm-se os valores Sp_j e Sr_k para as interfaces providas e requeridas do serviço candidato. Essas similaridades

somente são calculadas caso a interface candidata possua ao menos uma operação em comum com a interface especificada.

As equações 04 e 05 avaliam individualmente cada interface provida e requerida de um determinado serviço. Tal como definido, os valores das métricas funcionais também estão no intervalo (0, 1), sendo que, quanto mais próximo de 1 é o valor, melhor será a implementação da interface provida ou requerida no serviço candidato. Como as equações avaliam individualmente cada interface, é preciso que se tenha uma avaliação geral para o serviço, visando exibir o nível no qual o serviço web candidato atende às operações especificadas.

Como dito, é necessário obter a métrica funcional para o serviço como um todo, revelando o quão semelhante são as operações providas e requeridas na especificação e implementação do serviço. Assim, considerando todas as interfaces providas e requeridas de uma especificação, como indicado pela Equação 06, o valor da métrica funcional para o serviço é definido pela relação em que o numerador é a soma total da métrica funcional para cada interface requerida e provida, enquanto o denominador é o número total de interfaces requeridas e providas. Como definido, o valor da métrica funcional para um determinado serviço, também está no intervalo (0, 1).

$$S_i = \frac{\sum_{k=1}^{|RS \cup RI|} Sr_i + \sum_{j=1}^{|PS|} Sp_j}{|RS \cup RI| + |PS|} \quad (06)$$

Como pode ser observado na Equação 06, em termos de interfaces requeridas, a métrica funcional compreende o número de interfaces tanto na especificação como também na implementação do serviço ($|RS \cup RI|$). No entanto, em termos de interfaces providas, a métrica funcional para o serviço compreende apenas o número de interfacesna especificação do serviço ($|PS|$). É importante observar que as interfaces providas na implementação mas que não foram especificadas, não representam um esforço de adaptação extra, e assim, a Equação 06 não conta com as interfaces providas pela implementação do serviço (PI).

Semelhante à Equação 02 que representa a qualidade estrutural de uma configuração arquitetural, tem-se também uma equação que representa a qualidade funcional individual de cada implementação de serviço web que compõe a configuração arquitetural. Vale ressaltar que a qualidade funcional de cada serviço reflete na qualidade funcional da arquitetura. Portanto, a qualidade funcional da arquitetura em termos de seus serviços é representada pela Equação 07.

$$C_x = \frac{\sum_{i=1}^{S} s_i}{S} \quad (07)$$

Na Equação 07, C_x remete a avaliação geral de todos os serviços candidatos na configuração arquitetural, onde o valor da métrica funcional de toda arquitetura é definido pela relação entre a soma total da métrica funcional de cada serviço e o número total de serviços na arquitetura (*S*). Assim, o valor da métrica para a avaliação funcional da arquitetura também está no intervalo entre (0, 1). Quanto mais próximo de 1 é o valor, melhor a configuração arquitetural candidata.

4.4. Seleção de Serviços Web

No processo de desenvolvimento com SOA, a etapa de Seleção de Serviços Web tem o propósito de retornar uma configuração arquitetural que atenda aos requisitos especificados, para então ser aplicada uma técnica de integração dos serviços selecionados, denominada de orquestração e/ou coreografia de serviços, permitindo que o sistema seja construído na etapa de Integração do Sistema de Software.

Como discutido anteriormente, cada implementação não é escolhida somente porque suas funcionalidades providas e requeridas são similares ao serviço especificado, mas também a escolha depende dos outros serviços web que o mesmo irá se relacionar diretamente. Após o arquiteto de software obter a descrição da especificação de cada serviço da arquitetura do sistema, o mesmo irá realizar uma busca em Repositórios de Serviços para recuperar as descrições WSDL das implementações dos serviços candidatos, juntamente com as informações de dependências referente a outros serviços.

Referente às métricas aplicadas no processo de seleção de serviços web deste trabalho, elas são de suma importância para guiar o algoritmo de busca metaheurística, visando escolher uma solução próxima da ótima de forma automatizada. Todas as informações do problema são mapeadas para a metaheurística escolhida como o motor de seleção de serviços web. Dessa maneira, as descrições de serviços retornadas pela consulta ao repositório e as métricas fazem parte do processo de seleção. Essa abordagem configura-se como um problema da Engenharia de Software Baseada em Buscas, pois aponta para uma situação de selecionar uma configuração arquitetural aceitável, diante de inúmeras combinações de serviços que podem ser formadas, favorecendo a adoção de uma metaheurística para buscar soluções de acordo com as métricas relacionadas ao contexto do problema.

Considerando que o serviço web especificado pode ter várias implementações candidatas, a seleção de serviços web deve explorar métricas definidas para avaliar a arquitetura sob diferentes perspectivas. Neste trabalho, são exploradas as perspectivas de avaliar a integração entre os serviços web selecionados, e avaliar a similaridade entre a implementação e especificação do serviço.

A primeira perspectiva avalia a estrutura da arquitetura como descrito na Equação 02, se concentrando na integração entre os serviços. Tendo em vista que cada serviço pode se conectar com vários outros, a escolha de um serviço pode impactar na escolha de outros serviços candidatos. Cada decisão de selecionar um candidato pode impactar na escolha de outro.

Na avaliação de similaridade entre os serviços, se compara e avalia o conjunto de interfaces de uma implementação candidata com as interfaces de sua respectiva especificação. Aqui, consideramos que a especificação do serviço na arquitetura possui um conjunto de implementações candidatas, e, de acordo com as métricas que avaliam o serviço, aquela implementação mais similar em relação à especificação deve ter um valor maior para a métrica de similaridade. Então se percorre cada conjunto de implementações candidatas e pontuam-se os mais similares com maiores valores.

Com o intuito de facilitar o entendimento, um mapeamento da avaliação estrutural é ilustrado na forma de um grafo na Figura 18. De acordo com a figura, veem-se quatro conjuntos que representam quatro especificações de serviços web para uma arquitetura.

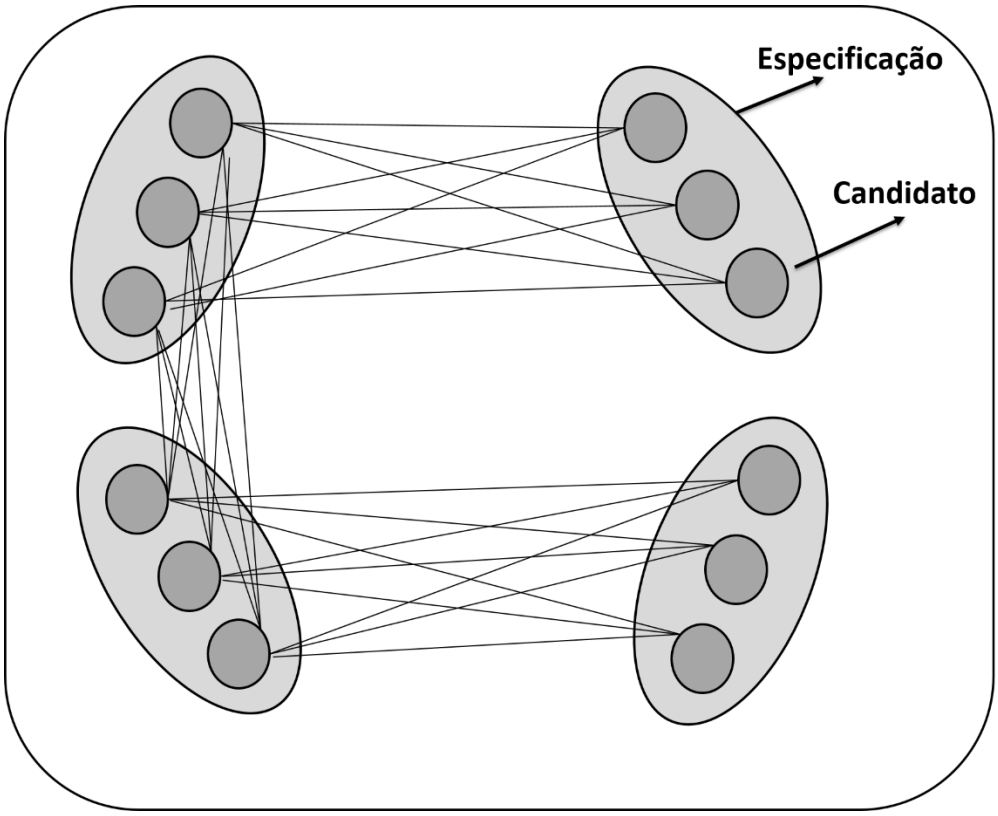

Figura 18 - Arquitetura na perspectiva de grafos

Cada conjunto corresponde a uma especificação que possui *N* implementações candidatas. Cada implementação candidata de uma especificação possui ligações via arestas com todos os candidatos das outras especificações relacionadas, exceto com os outros candidatos da própria especificação. A intenção é escolher um par de implementações candidatas para cada par de especificações relacionadas. Toda aresta possui um peso que corresponde ao valor quantitativo resultante da avaliação da métrica que analisa a conexão entre candidatos.

Este trabalho utiliza Algoritmos Genéticos (AG), uma metaheurística que busca otimizar a busca por boas soluções em grandes espaços de busca. Algoritmos Genéticos são adequados para problemas de otimização combinatória, tal como a seleção de serviços web. Fatores importantes que levaram a escolha de AG, é que o mesmo proporciona um conjunto de soluções candidatas ordenadas por meio do resultado da função objetivo, são amplamente utilizados com sucesso em problemas de difícil manipulação, possuem grande eficiência em encontrar boas soluções, e possuem bastante flexibilidade para percorrer o espaço de busca com mecanismos que minimizam as chances de encontrar em ótimos locais.

A representação genética consiste em transformar conceitos do problema em símbolos para que seja possível realizar manipulações simbólicas do algoritmo (Harman, 2012). Um

cromossomo ou indivíduo do AG representa uma solução, que é formada por um conjunto de genes. Então, deve-se construir uma representação que mapeia soluções do problema em cromossomos. Na perspectiva do problema de seleção de serviços web, uma solução corresponde a uma instância arquitetural, que é formada por implementações dos serviços web, que representam os genes do cromossomo.

O primeiro passo ao trabalhar com AG é definir como o problema será representado. Para o problema atual, em que se busca a solução mais adequada para uma arquitetura com *N* serviços, a representação pensada para o algoritmo genético utiliza um vetor (cromossomo) de *N* serviços web, onde cada posição (gene) do vetor representa uma especificação de serviço, cujos alelos (formas alternativas do gene) são implementações candidatas para tal especificação. A Figura 19 mostra a representação genética escolhida, na qual o cromossomo representa o vetor contendo uma possível solução, onde *Si* é o serviço escolhido para atender a uma determinada especificação presente na arquitetura.

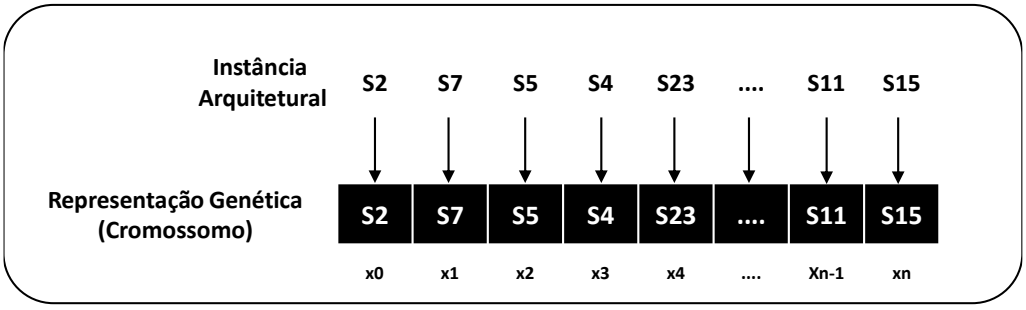

Figura 19 - Representação genética escolhida

A utilização de AG é fundamentada no processo de evolução natural por meio de seleção, onde os mais fortes sobrevivem diante dos mais fracos (Forrest, 1996). Com isso, é necessário à criação de uma função que avalie se uma solução é considerada boa ou ruim. Essa função é conhecida como função objetivo.

A Equação 08 apresenta a função objetivo do algoritmo genético proposto neste trabalho que agrupa os dois objetivos supracitados: (*i*) a avaliação estrutural quemensura a integração entre os serviços, representada pela Equação 02; e (*ii*) a avaliação de similaridade entre a especificação e implementação do serviço web, representada pela Equação 07. Tais objetivos são combinados em conjunto de modo a obter a função objetivo adotada na técnica de busca pela metaheurística. Em tal sentido, a função objetivo representada pela Equação 08 é definida como uma média ponderada normalizada das métricas funcionais e estruturais, em que os termos w_a e w_c representam seus respectivos pesos normalizados. Como pode ser observado,

o valor da função objetivo está no intervalo (0, 1), tendo o valor 1 como a melhor configuração arquitetural candidata em termos de esforço de adaptação.

$$F_x = w_a . A_x + w_c . C_x \begin{cases} 0 \leq w_a \leq 1 \\ 0 \leq w_c \leq 1 \\ w_a + w_c = 1 \end{cases} \quad (08)$$

Com o entendimento das métricas funcionais e estruturais abordadas neste trabalho, e a função objetivo utilizada pela metaheurística, em seguida é ilustrado na Figura 20 o ciclo de vida do algoritmo genético utilizado, definindo e detalhando as etapas que compõem o algoritmo proposto desde o início até o seu término, quando o critério de parada é atingido.

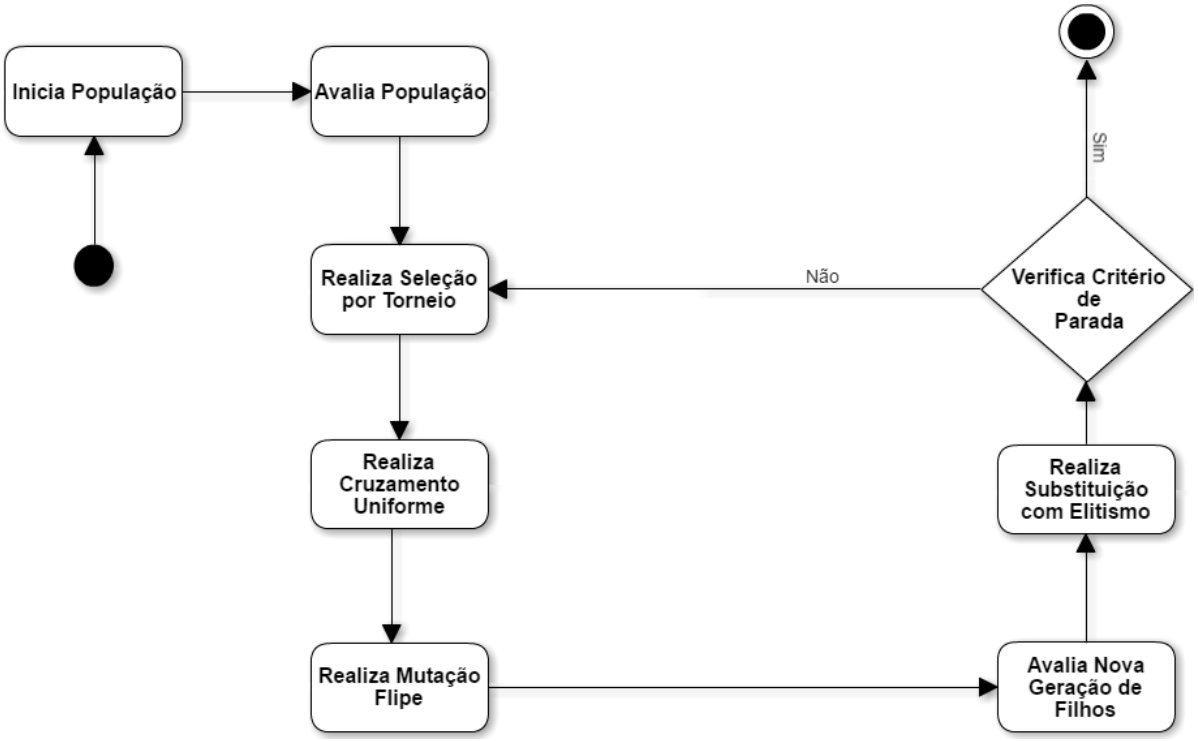

Figura 20 - Ciclo de via do AG proposto neste trabalho

Explicando o ciclo do AG proposto, inicialmente é criada a população de indivíduos de forma totalmente aleatória, visando evitar possíveis vícios e possibilitando que a população se espalhe ao máximo no espaço de busca. Em seguida, a população é avaliada pela função objetivo representada pela Equação 08, onde cada indivíduo recebe uma pontuação baseada nas métricas funcionais e estruturais definidas.

O próximo passo do algoritmo consiste na etapa de seleção de pais. Para o algoritmo genético proposto, é utilizada a técnica de seleção por torneio. Tratando da etapa de seleção do

algoritmo genético, a técnica de seleção por torneio provavelmente é a abordagem mais popular devido a sua simplicidade de implementação e eficiência (Xie, 2009). Em um problema clássico de combinatória encontrado na literatura, como é o problema do caixeiro-viajante, que serve de exemplo para testar o desempenho de novas abordagens algorítmicas de metaheurística, o método de torneio superou outros bastante utilizados, como os mencionados anteriormente (Razali, 2011). Segundo Matoušek (2008), a única desvantagem da seleção por torneio é a falta de garantia na permanência do melhor indivíduo. Porém essa deficiência é solucionada com a utilização da prática elitista aplicada pelo algoritmo genético proposto. Matoušek (2008) também afirma que a seleção por torneio possivelmente é o melhor mecanismo de seleção em algoritmo genético. É importante que a seleção dê alguma preferência para os melhores indivíduos da população, baseado na função objetivo, mas também permita que indivíduos não tão bons também sejam pais, garantindo diversidade genética e uma busca mais aprofundada no espaço de soluções (Goldberg, 1991).

Após a escolha da técnica de seleção de indivíduos que o AG utiliza, é realizada a escolha das técnicas dos operadores genéticos de cruzamento e mutação. Uma má escolha dos operadores pode restringir a diversidade genética das soluções encontradas, diminuindo a efetividade do algoritmo e aumentando a possibilidade de parada em ótimos locais.

O operador de cruzamento selecionado é denominado cruzamento uniforme. Neste operador, é gerada uma máscara binária que é utilizada para identificar quais genes dos pais serão permutados, e esse cruzamento de genes irá gerar os dois filhos. Para cada posição do vetor do cromossomo, a respectiva posição no vetor da máscara recebe aleatoriamente o valor 0 ou 1. Se a posição do vetor da máscara for 1, o primeiro filho recebe os genes do primeiro pai, enquanto o segundo filho recebe os genes do segundo pai. Se o valor for 0, o primeiro filho recebe os genes do segundo pai e o segundo filho recebe os genes do primeiro pai. Os valores binários da máscara são gerados aleatoriamente a cada recombinação entre cromossomos pais. A Figura 21 representa como é feito o processo de recombinação, onde se aplica uma máscara que será aplicada na geração de dois filhos.

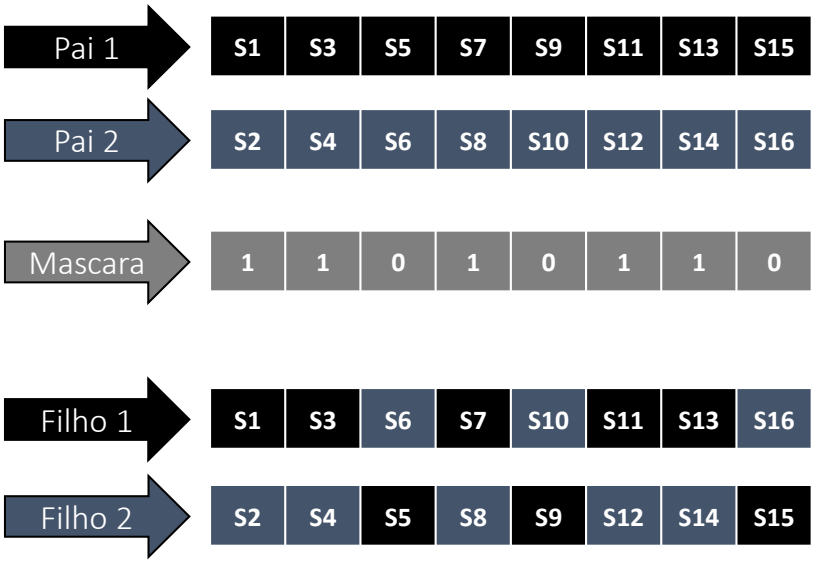

Figura 21 - Técnica de cruzamento uniforme utilizada pelo AG Proposto

O operador de cruzamento uniforme é mais propício a problemas combinatórios, porque qualquer gene pode ser herdado pelos filhos, diferentemente da abordagem de cruzamento simples em que os valores dos genes das pontas do cromossomo tendem a permanecer juntos, e, caso essa união de valores implique em uma redução de qualidade no indivíduo, a destruição dessa união ficará à mercê da mutação genética. O cruzamento uniforme possui maior chance de quebrar esquemas da estrutura do cromossomo, isso ajuda o algoritmo genético em uma melhor varredura do espaço de busca, e, se por acaso, essa técnica destruir alguma solução muito boa, a abordagem elitista irá mantê-la nos pais.

A Figura 22 apresenta o operador de mutação flip adotado por este trabalho, exemplificando que cada posição do cromossomo possui probabilidade *P* de sofrer variação em seu valor. A variação ocorre com a substituição de um gene no cromossomo por um gene candidato na população. Um dos principais benefícios da mutação, é proporcionar uma melhor diversificação na varredura do espaço de busca, permitindo escapar de máximos locais.

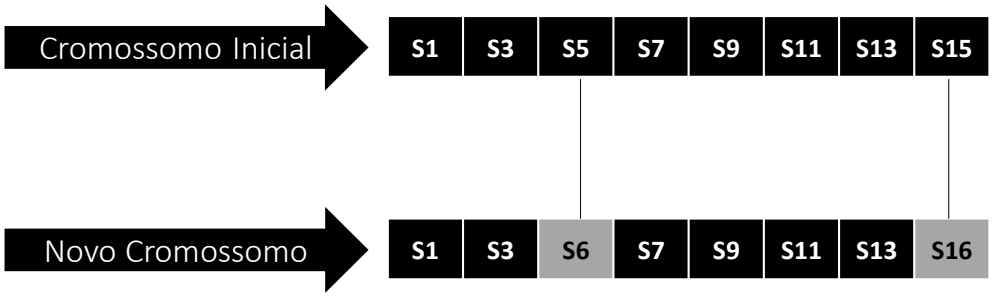

Figura 22 - Técnica de mutação flip utilizada pelo AG proposto

As últimas etapas do algoritmo proposto consite da avaliação da nova geração de filhos, e substituiação por meio da técnica elitista, que consiste em copiar os melhores indivíduos da população anterior para fazerem parte da população que está sendo formada. Esta abordagem garante que a qualidade das soluções do AG sempre melhore conforme o decorrer das iterações do algoritmo. A função que representa o melhor indivíduo da população no decorrer do tempo tem o comportamento monótono e crescente. Como, por natureza, AG não possui memória, caso a população tenha um indivíduo que representa a solução ótima, essa solução pode ser perdida sem a adoção do elitismo, pois parte da população morre a cada nova geração. Mesmo quando uma boa solução gera soluções filhas por meio dos operadores genéticos, essas novas geralmente não são soluções tão boas quanto os pais. Mediante isso, é importante proporcionar uma pequena parcela de memória para que uma boa solução encontrada não seja perdida. Segundo Linden (2008), o ato de utilizar a abordagem elitista, apesar de ser simples, usualmente colabora de forma positiva para a melhoria do desempenho do AG. As melhores soluções são mantidas dentro da população e são responsáveis por contribuir para que boas soluções sejam geradas a partir da recombinação com os melhores indivíduos da população anterior.

Após todas as etapas citadas serem concluídas, o algoritmo avalia se o critério de parada foi atingido, caso seja concretizada a condição de parada do algoritmo, é realizada a ordenação com o intuito de apresentar a população ordenada para obter um subconjunto das melhores soluções e apresentá-las ao responsável pelo desenvolvimento do sistema de software. De forma contrária, a população deve passar iterativamente por todo o processo até que a condição de parada seja satisfeita, e sejam encontradas boas soluções caracterizando a população final.

4.5. Considerações Finais

Neste capítulo, foi apresentada a abordagem desenvolvida para selecionar serviços web de maneira automatizada e guiada por métricas que avaliam os serviços candidatos por meio de sua similaridade com a especificação, como também pelo ponto de vista da integração entre os serviços web candidatos.

A primeira etapa da abordagem proposta é a Identificação das Funcionalidades e Dependências dos serviços web, que recebe como entrada três artefatos, são eles: especificação da arquitetura, descrição WSDL, e diagramas de sequência. Com esses dois últimos artefatos é possível identificar as interfaces providas e requeridas respectivamente.

A seleção de serviços web é realizada utilizando técnicas da Engenharia de Software Baseada em Buscas. Os serviços web candidatos são buscados em um repositório, quando

selecionados, são avaliados por métricas definidas com o propósito de avaliar os serviços na perspectiva estrutural da arquitetura e do próprio serviço web.

Como o problema de seleção tratado neste trabalho possui complexidade exponencial, foi utilizada a metaheurística algoritmo genético com o intuito de alcançar boas soluções e por consequência obter um sistema com qualidade, e com seu custo e tempo de integração reduzido. A função objetivo do algoritmo é baseada nas métricas definidas, onde as soluções são guiadas em função das decisões e restrições definidas nas métricas. Após o AG apontar as melhores soluções alcançadas em função de tais métricas, uma solução candidata é escolhida e enviada para etapa de Integração de Serviços Web, onde as implementações dos serviços web serão integradas e formarão realmente um sistema de software concreto. Vale relembrar que este trabalho tem seu foco nas etapas de Identificação das Funcionalidades e Dependências do Serviço, e na Seleção de Serviços Web.

Capítulo 5

Avaliação Experimental

Neste capítulo é abordada a maneira como foi construído o ambiente para executar os experimentos do presente trabalho, para com isso realizar a avaliação da abordagem proposta. O intuito desse ambiente é gerar uma base de dados para que seja processada pelo algoritmo genético proposto. A fim de comparar os resultados obtidos pela abordagem proposta, será realizada a execução de outros dois algoritmos de busca, a saber: o algoritmo de busca aleatória e o algoritmo de busca exaustiva. Essa comparação tem como objetivo evidenciar que a abordagem proposta fornece soluções com qualidade próxima ou equiparável ao algoritmo de busca exaustiva. Já referente ao algoritmo de busca aleatória, o algoritmo proposto apresenta soluções com qualidade superior, demonstrando que o esforço empregado na concepção do algoritmo resultou na melhoria qualitativa das soluções apresentadas.

De início são apresentados os parâmetros para o algoritmo genético desenvolvido nesse trabalho, e o espaço de busca explorado. Além da apresentação dos resultados obtidos por cada algoritmo, também são realizados testes comparativos como meio de evidenciar que a abordagem proposta possui desempenho melhor que a busca aleatória, e equiparável à busca exaustiva.

5.1. Ambiente Experimental

Antes de dar início ao processo de avaliação da abordagem proposta, deve-se ter um ambiente experimental propício que simule adequadamente uma etapa de seleção de serviços automatizada por algoritmo genético. A priori, a etapa *Identificação de Funcionalidades e Dependências* adota como entrada três tipos de artefatos: (*i*) a especificação da arquitetura do sistema a ser desenvolvido; (*ii*) as descrições WSDL a fim de identificar as operações providas; e (*iii*) os diagramas de sequência associados a cada operação provida pelos serviços web, com o objetivo de identificar as interfaces requeridas. Ao término, é possível obter uma arquitetura formada por especificações de componentes contendo as interfaces providas e requeridas de cada serviço web. Essa arquitetura representa a base de referência para a criação do sistema em desenvolvimento.

Da mesma forma que nos trabalhos relacionados, para os experimentos realizados nessa dissertação não foi possível a utilização de um repositório real com variações das implementações dos serviços especificados na arquitetura. Em função desta indisponibilidade e com o intuito de simplificar, optou-se por adotar a geração de candidatos de forma aleatória. Essa forma de geração não influencia na execução do algoritmo, pois cada candidato gerado é similar em ao menos uma operação ao serviço web especificado.

Os serviços web candidatos são gerados da seguinte forma: pega-se cada especificação de serviço web como referência, e cria-se variações dela a nível de interfaces e operações, ou seja: um serviço web candidato faz uma cópia da especificação que possui como referência, posteriormente esse candidato terá modificações em cada interface, adicionando novas operações e/ou retirando algumas operações existentes. No entanto, a interface sempre terá pelo menos uma operação que corresponde a sua interface de referência. O candidato também poderá receber novas interfaces e/ou perder algumas. Esse processo cria um candidato referente a uma especificação de serviços web da arquitetura, então ele é repetido *n* vezes, onde *n* é o número candidatos que se queira criar.

A Figura 23 ilustra como os serviços web candidatos são gerados a partir de uma especificação pertencente à arquitetura do sistema em construção. O candidato *Implementação A* possui uma interface provida a mais, e duas delas são similares às interfaces especificadas, contudo, a interface requerida é idêntica. Já o candidato *Implementação B* apresenta uma interface provida a menos, no entanto, as demais interfaces desse candidato são similares as especificadas. E assim por diante vão sendo geradas variações aleatórias da especificação até se obter um número *n* de candidatos.

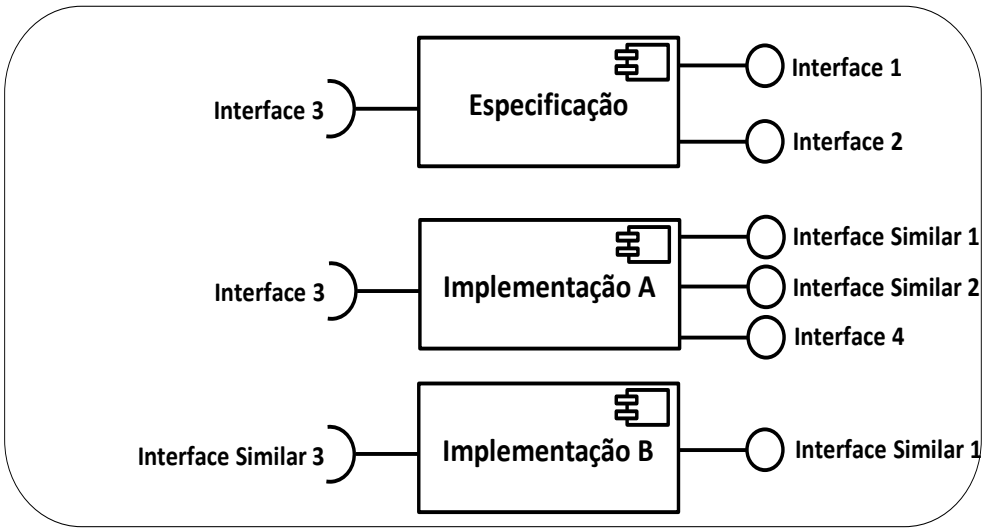

Figura 23 - Criação de serviços web candidatos

Depois que todos os candidatos são criados, chega o momento do algoritmo genético proposto ser iniciado. O primeiro passo do algoritmo é a criação da população, então os serviços web candidatos de cada especificação são selecionados para os respectivos genes, que conjuntamente formam cromossomos (soluções candidatas). O algoritmo genético executa suas várias iterações até que a condição de parada seja satisfeita. Relembrando, a condição de parada para o AG proposto é não ocorrer melhora na solução durante 100 iterações subsequentes no algoritmo. Com base no exposto a criação da massa de dados é criada para os experimentos, que têm como objetivo validar a abordagem proposta.

5.2. Avaliação

Ao implementar uma metaheurística, se faz necessário a realização do processo de avaliação da abordagem algorítmica com o intuito de obter confiança na qualidade dos resultados encontrados. É comum utilizar testes comparativos nesse contexto de avaliação, como evidenciar o quão próximo os resultados da metaheurística são em relação à solução ótima. Isso configura uma comparação entre os resultados do algoritmo em análise com os resultados de um algoritmo exaustivo que retorna à solução ótima.

Como a maior motivação da elaboração de uma metaheurística é a ausência de um algoritmo que encontre a solução ótima em tempo polinomial, a comparação deve ser feita com um espaço de busca reduzido para que um algoritmo de busca exaustiva encontre a solução ótima em um tempo viável. Se houvesse um algoritmo que encontrasse a solução ótima em tempo polinomial, não haveria necessidade de se criar uma metaheurística. Então, como deve-

se ter uma comparação entre a abordagem proposta e o algoritmo de busca exaustiva, se faz necessário adotar um espaço de busca reduzido.

Uma metaheurística também deve ser avaliada pela comparação com alternativas mais simples, visando justificar o esforço empregado na elaboração dessa metaheurística, pois se a mesma apresentar resultados equiparáveis aos de uma alternativa mais simples, não se faz necessário o esforço para sua implementação e utilização, uma vez que uma implementação mais simples já poderia apresentar soluções satisfatórias. Segundo Ali (2009), a comparação com abordagens mais simples é necessária sempre que se propõe uma abordagem metaheurística que trata algum problema da Engenharia de Software. Como a abordagem proposta utiliza algoritmos genéticos, onde o mesmo é considerado um algoritmo aleatório guiado pela abordagem da seleção natural, então comumente a alternativa mais simples que é comparada com o algoritmo genético é o algoritmo de busca aleatória. A intenção da escolha da busca aleatória para comparação se deve pelo motivo de mostrar que o algoritmo genético implementado é diferenciado de uma mera busca aleatória, pois as informações do problema inseridas nele devem guiar a busca para encontrar boas soluções, ou até mesmo a melhor solução.

Portanto, a intenção desse processo de avaliação é mostrar que a formulação do problema da Engenharia de Software tratado nesse trabalho foi executada de forma consistente, em que as soluções resultantes do algoritmo proposto devem ser próximas ou iguais da solução ótima e também devem ser melhores que soluções apresentadas por uma busca aleatória. A Figura 24 mostra a comparação da qualidade das soluções entre essas abordagens relatadas, ou seja, os resultados obtidos com a abordagem proposta devem ser melhores que aqueles obtidos com a busca aleatória, e, ao mesmo tempo, aproximados ou até mesmo iguais aos obtidos com a busca exaustiva.

Figura 24 - Comparação do desempenho qualitativo das soluções

Para comparar abordagens algorítmicas em um determinado domínio de aplicação, e apontar se uma é melhor que a outra ou se são equivalentes, se faz necessário a utilização de alguns dados estatísticos associados às soluções encontradas. Esses dados avaliam se existem evidências suficientes para afirmar que a qualidade dos resultados de um conjunto de amostras

apresenta soluções melhores ou piores quando comparados com um conjunto de amostras de outro algoritmo. A intenção de comparar as soluções encontradas é demonstrar que o algoritmo proposto sobrepõe a qualidade do algoritmo com uma abordagem mais simples, pois o esforço em construir um algoritmo mais complexo deve ser justificado por soluções com uma melhor qualidade.

É sabido que um algoritmo de busca exaustiva sempre encontrará a solução ótima, e como a metaheurística implementada neste trabalho se propõe a encontrar soluções próximas da ótima ou até mesmo a ótima, a comparação entre os dois algoritmos se faz necessária. Como se tem o conhecimento que os resultados provenientes do algoritmo de busca exaustiva sempre irão superar os resultados de qualquer outro algoritmo, resta ao algoritmo genético proposto mostrar o quão próximo da solução ótima são os resultados encontrados.

5.3. Experimentos

Nesta seção é apresentado a forma como os experimentos são realizados, evidenciando os resultados apresentados pelos algoritmos, realizando comparativos entre tais resultados, e destacando a eficiência e os ganhos do algoritmo genético proposto sobre o algoritmo de busca aleatória.

A princípio é necessário ter uma arquitetura para concretizar o ambiente experimental. A avaliação experimental desde trabalho foi realizada utilizando quatro especificações de arquiteturas diferentes, são elas: (*i*) arquitetura formada por 5 especificações de serviços web, possuindo 6 dependências entre eles; (*ii*) arquitetura formada por 10 especificações de serviços web, que possuem 12 dependências entre si; (*iii*) arquitetura com 20 especificações de serviços web, onde é possível contabilizar 25 dependências entre os serviços; e (*iv*) arquitetura com 30 especificações de serviços web, contabilizando 36 dependências entre os serviços.

Para cada uma dessas quatro arquiteturas, foi criado um diagrama UML a fim de representar cada arquitetura adotada nos experimentos. A avaliação de cada arquitetura ocorre em três cenários diferentes, onde cada especificação de serviço web possui 30 implementações candidatas, no entanto variando o número de especificações que têm implementação candidata perfeita. Os cenários para a avaliação das quatro arquiteturas são: *(i)* todas as especificações possuem candidatos perfeitos; *(ii)* metade das especificações com candidatos perfeitos; e *(iii)* ausência de candidatos perfeitos.

Como as arquiteturas possuem tamanhos diferentes, a quantidade de combinações possíveis é o número de candidatos elevado ao número de especificações na arquitetura. Com

isso, para a arquitetura com 5 serviços web, tem-se uma quantidade de 30^5 combinações possíveis. Já para a arquitetura com 10 serviços web, tem-se uma quantidade de 30^{10} combinações possíveis. Para uma arquitetura com 20 serviços web tem-se uma quantidade 30^{20}, e para 30 serviços web tem-se a quantidade de 30^{30} combinações possíveis. Portanto, a fim de buscar a solução ótima e comparar com a solução encontrada pelo algoritmo genético proposto, as arquiteturas que possuem 5 e 10 especificações de serviços web são utilizadas, pois as arquiteturas com 20 e 30 serviços web geram um número de combinações exponencial, acarretando em gigantescos espaços de busca, impossibilitando obter a solução por intermédio de um algoritmo de busca exaustiva em tempo realístico.

O algoritmo proposto por este trabalho possui o objetivo de buscar soluções satisfatórias sob a orientação de duas métricas: (*i*) métrica estrutural, onde é avaliada de acordo com a Equação 01 a integração entre os serviços, e Equação 02 toda a estrutura da solução formada pelas implementações; e (*ii*) métrica funcional, na qual de acordo com a Equação 06 avalia a similaridade entre a implementação e especificação do serviço web, e de acordo com a Equação 07 todos os serviços web candidatos na configuração arquitetural. Com isso, os experimentos são efetuados nas duas perspectivas, onde é avaliada uma função objetivo que une ambas. Como discutido anteriormente, todas as métricas estão com valores normalizados no intervalo [0, 1], ou seja, quanto mais próximo de 1 for o valor da função objetivo, melhor será a qualidade da solução. Neste trabalho, cada métrica possui peso 0,5 na função objetivo representada pela Equação 08. No entanto, é importante frisar que esse peso é parametrizável de acordo com a necessidade específica de cada projeto.

Como o algoritmo proposto soma de forma ponderada os valores produzidos pelas duas métricas, a computação de um objetivo não interfere na computação do outro. Ou seja, a execução de uma métrica não espera pela execução nem altera os valores da outra. Portanto, quando a função objetivo pondera os valores, cada um será consistente e contribuirá para seleção de serviços web na proporção que foi definida.

Conforme abordado anteriormente, as soluções encontradas pelo algoritmo genético proposto serão comparadas com o resultado da busca exaustiva, com o intuito de provar a sua eficiência. A outra comparação será realizada com o algoritmo de busca aleatória, com o intuito de evidenciar os ganhos do AG sobre uma busca meramente aleatória. Mediante isso, um dado importante a ser considerado é o número de amostras que devem ser obtidas para realizar com propriedade a comparação dos resultados apresentados pelos algoritmos. De acordo com Arcuri (2011), 1000 amostras é uma boa quantidade para realizar testes de comparação entre os

resultados dos algoritmos. Portanto, a confiabilidade dos testes realizados nesse trabalho é melhor que em outros trabalhos que utilizam metaheurísticas na Engenharia de Software, no qual na maioria das vezes se limitam a uma quantidade de 100 amostras apenas.

Para obter as 1000 amostras, o algoritmo de busca aleatória e o algoritmo genético proposto são executados 1000 vezes. Consequentemente, cada algoritmo apresenta o mesmo número de soluções. Com isso, têm-se duas populações com 1000 amostras cada, as quais serão analisadas, visando mostrar que o algoritmo genético proposto se sobrepõe ao algoritmo de busca aleatória.

Para mostrar que as soluções do algoritmo genético são próximas da solução ótima, a alternativa mais comum e prática é ter um espaço de busca reduzido para que o algoritmo de busca exaustiva obtenha a solução ótima em tempo viável, para então, comparar com as soluções do algoritmo genético proposto. Desta forma, o espaço de busca da arquitetura com 5 especificações de serviços web, que é de 30^5, e a arquitetura com 10 especificações de serviços web, que é de 30^{10} combinações possíveis, são utilizados para realizar essa comparação. É importante ressaltar que a execução do algoritmo de busca exaustiva se torna inviável para as arquiteturas com 20 e 30 especificações de serviços web, pois o espaço de busca é gigantesco, e, assim, o algoritmo de busca exaustiva não retorna a melhor solução em tempo realístico e viável.

Antes de apresentar os experimentos, é importante destacar como o algoritmo genético proposto foi parametrizado. Para cada geração do AG, a população é igual a 300 configurações arquiteturais candidatas. O critério de parada é alcançado quando o valor de aptidão da mais alta solução torna-se estável em um patamar durante 100 iterações sucessivas e já não produz melhores resultados. A seleção das soluções candidatas para produzir uma nova geração é baseada no método de torneio. Para reprodução de uma próxima geração, o método de cruzamento uniforme é adotado, em conjunto com uma taxa de mutação flip de $\frac{1}{n}$, onde *n* é a quantidade de serviços especificados.

Para o algoritmo de busca aleatória, a quantidade de possíveis soluções candidatas é baseada no seguinte critério: acrescenta-se 100 iterações sob o maior número da iteração que o algoritmo genético encontrou uma boa solução e não obteve melhora, em seguida, o resultado desta soma é multiplicado por 300. Por exemplo, caso o AG encontre uma boa solução na iteração 50, se durante as próximas 100 iterações seguidas não houver melhora, a quantidade de possíveis soluções da busca aleatória é 45.000, derivado da expressão (100+50)*300, ou seja, entre 45 mil possíveis soluções, a busca aleatória seleciona a melhor delas.

Com a finalidade de mostrar que as soluções encontradas pelo algoritmo genético proposto possuem uma eficiência considerável quando comparadas com as soluções apresentadas pela busca aleatória, é realizado o cálculo $E = \{((AG - BA)/\, BA) * 100\}$, cujo valor é apresentado em forma de percentual, representando o quão melhor é a solução encontrada pelo algoritmo genético em relação à busca aleatória.

Por fim, no restante desse capítulo são apresentados os experimentos realizados, e resultados encontrados para os estudos de caso desde trabalho. Foram realizados quatro estudos de caso, para arquiteturas de sistemas com 5, 10, 20 e 30 especificações de serviços web.

5.3.1. Arquitetura com 5 Especificações de Serviços Web

Esta seção apresenta os resultados para o primeiro experimento realizado neste trabalho. O intuito é mostrar os resultados encontrados para a arquitetura que possui uma especificação com 5 serviços web. Como dito anteriormente, de início é necessário ter uma arquitetura como modelo para concretizar o ambiente a ser utilizado no experimento, desse modo, é utilizada a especificação de um sistema real, que tem o objetivo de realizar vendas de pacotes turísticos. A especificação do sistema adotada neste experimento foi extraída do consórcio OASIS (2012). Esta especificação possui 5 serviços, contendo a descrição das operações providas por cada serviço, e o nível de dependência que os serviços possuem de outros. Baseado nessa especificação, foi desenvolvido um diagrama de componentes, conforme a Figura 25, para representar a arquitetura com as especificações de serviços, juntamente com as interfaces providas e requeridas de cada serviço web.

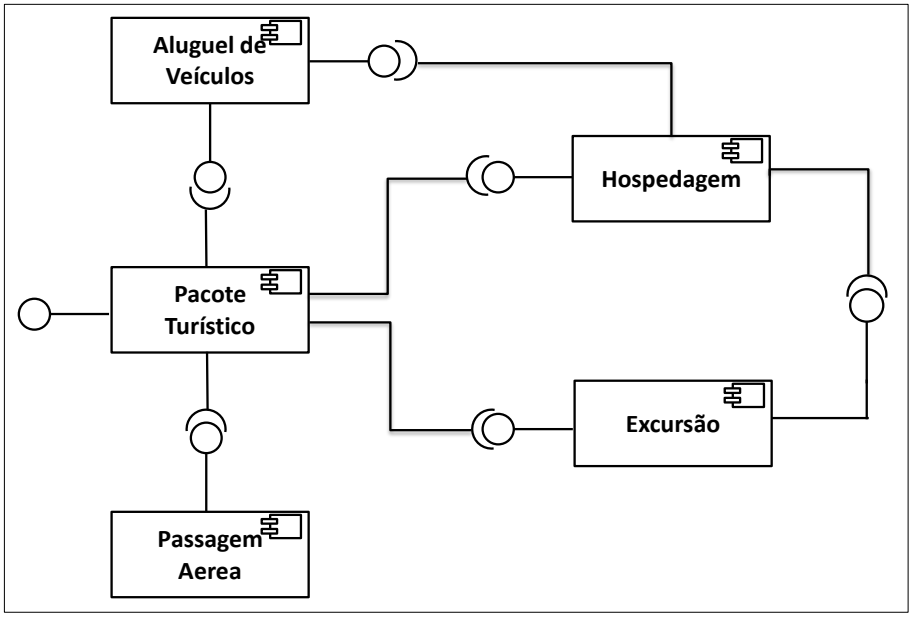

Figura 25 - Arquitetura com 5 serviços web

Para cada especificação de serviço presente na arquitetura, são disponibilizados 30 serviços web candidatos, gerando assim um espaço de busca de 30^5, o que equivale a 24.300.000 possíveis soluções. A fim de mostrar que as soluções encontradas pelo algoritmo proposto são próximas à solução ótima, a alternativa mais comum e prática, é comparar os resultados da abordagem proposta com o resultado apresentado pelo algoritmo de busca exaustiva. Logo após, a comparação é feita com um algoritmo de busca aleatória, tendo em vista que o algoritmo genético é um algoritmo aleatório guiado pelas variáveis inseridas nele. O intuito dessa comparação é mostrar que o tempo de processamento do algoritmo proposto é compensado pela qualidade da solução encontrada, quando comparado com um algoritmo puramente aleatório.

Conforme abordado anteriormente, a avaliação experimental ocorre em três cenários diferentes, onde cada especificação de serviço possui 30 implementações candidatas, no entanto variando o número de especificações de serviços web que tem implementações perfeitas, são eles: (*i*) todas as especificações de serviços web da arquitetura possuem candidatos perfeitos; (*ii*) três especificações de serviços web possuem candidatos perfeitos; e (*iii*) ausência de candidatos perfeitos para as especificações pertencentes a arquitetura.

Para cada cenário, a abordagem proposta foi comparada com a busca exaustiva e a busca aleatória. Em tal comparação, a abordagem proposta e a busca aleatória foram executadas 1000 vezes, e o valor médio da solução é calculado. Já a busca exaustiva foi executada apenas uma vez, para descobrir a solução ótima. Como mostra a Figura 26, os resultados experimentais

revelam que, em todos os cenários deste experimento, a abordagem proposta encontrou a solução ótima, que é confirmada pela busca exaustiva.

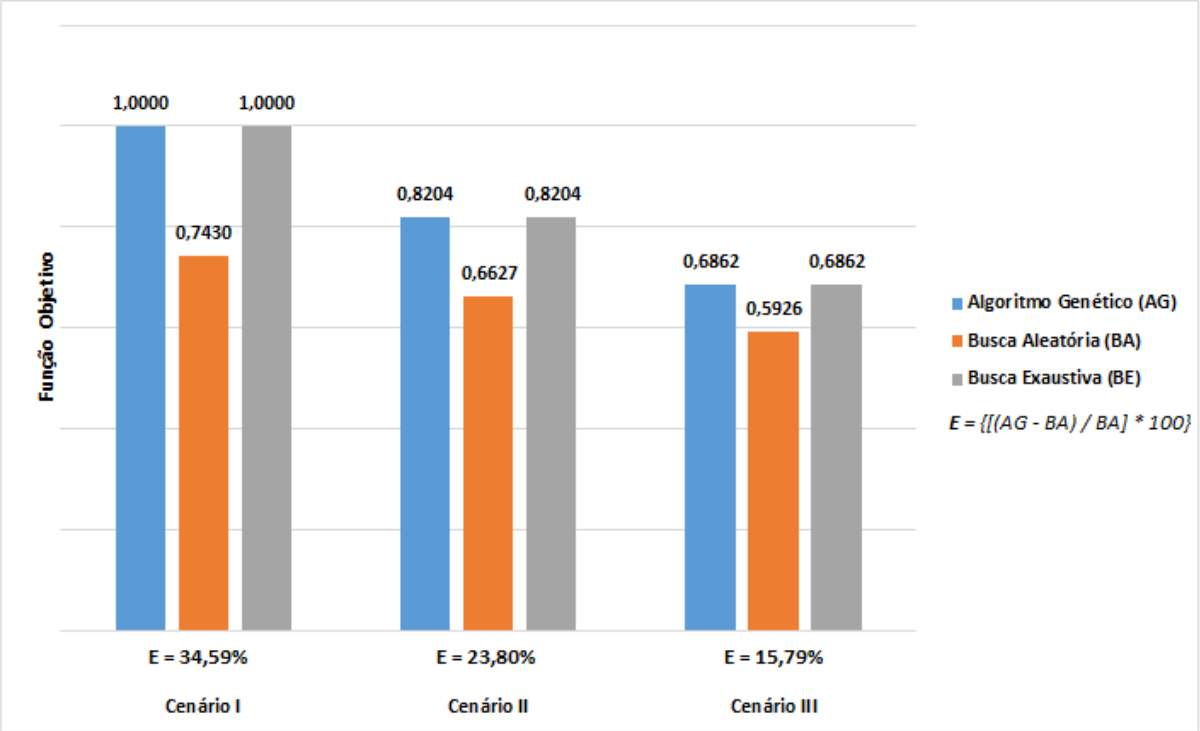

Figura 26 - Resultados experimentais para arquitetura com 5 serviços web

Com base nos resultados apresentados na Figura 26, é possível afirmar que no primeiro cenário, onde todas as especificações têm implementações perfeitas, a solução recomendada pela abordagem proposta é cerca de 34,59% mais eficiente do que o recomendado pela busca aleatória. No segundo cenário, onde apenas três especificações possuem implementações perfeitas, a eficiência da abordagem proposta em relação à busca aleatória é reduzida para aproximadamente 23,80%. Finalmente, na última situação, na qual há uma ausência de candidatos que possuem implementações perfeitas, a eficiência da abordagem proposta torna-se estável em cerca de 15,79%. É importante salientar que, no melhor dos casos, a eficiência do algoritmo proposto em relação à busca aleatória é 55,50% no primeiro caso, 38,77% no segundo caso, e 28,05% no terceiro caso.

Como outro resultado interessante, deve-se ressaltar a rápida convergência do algoritmo proposto. Isso pode ser comprovado pelo número de iterações que foi necessário para que o algoritmo encontre a melhor solução para o problema, algo que foi comprovado quando o resultado foi comparado com o da busca exaustiva. A Tabela 4 apresenta o número de iterações que o algoritmo genético encontra a melhor solução nos três cenários abordados. É importante evidenciar que para os algoritmos de busca aleatória e a busca exaustiva não é possível obter

essa informação, uma vez que esses algoritmos não realizam iteração em suas execuções, ou seja, enquanto o primeiro seleciona uma configuração arquitetural de forma aleatória, o segundo testa todas as possibilidades possíveis, retornando assim, a melhor solução para o problema.

Tabela 4 - Número de iterações do AG para uma arquitetura com 05 serviços web

	Menor Iteração	Maior Iteração	Média Iteração
Cenário I	2	8	5
Cenário II	2	7	5
Cenário III	2	11	7

O terceiro resultado a ser comparado neste experimento é o tempo de processamento da execução entre o algoritmo genético proposto e o de busca aleatória. A Tabela 5 apresenta em milissegundos os tempos de processamento dos algoritmos. Como pode ser observado, o tempo do algoritmo proposto é maior que o tempo da busca aleatória em todos os cenários, sendo em média o dobro. No entanto, a perda não é tão grande, tendo em vista que o tempo de processamento do algoritmo proposto é aceitável, e essa perda é facilmente superada pela qualidade da solução apresentada pelo algoritmo genético.

Tabela 5 - Tempo de Processamento para uma arquitetura com 5 serviços web

		Menor Processamento	Maior Processamento	Média de Processamento
Cenário I	***Algoritmo Proposto***	184	588	219
	Busca Aleatória	96	165	120
	Busca Exaustiva	-	-	329547
Cenário II	***Algoritmo Proposto***	191	365	226
	Busca Aleatória	97	169	119
	Busca Exaustiva	-	-	328155
Cenário III	***Algoritmo Proposto***	213	407	274
	Busca Aleatória	100	196	129
	Busca Exaustiva	-	-	283695

5.3.2. Arquitetura com 10 Especificações de Serviços Web

Nesta seção são apresentados os resultados encontrados para a arquitetura que possui uma especificação com 10 serviços web, e 12 dependências entre eles. Assim como no experimento anterior é necessário ter uma arquitetura como modelo para concretizar o ambiente experimental. A Figura 27 ilustra a arquitetura formada por 10 especificações de serviços web, que é baseada na Figura 25, sendo adicionados novos serviços para realização desde experimento.

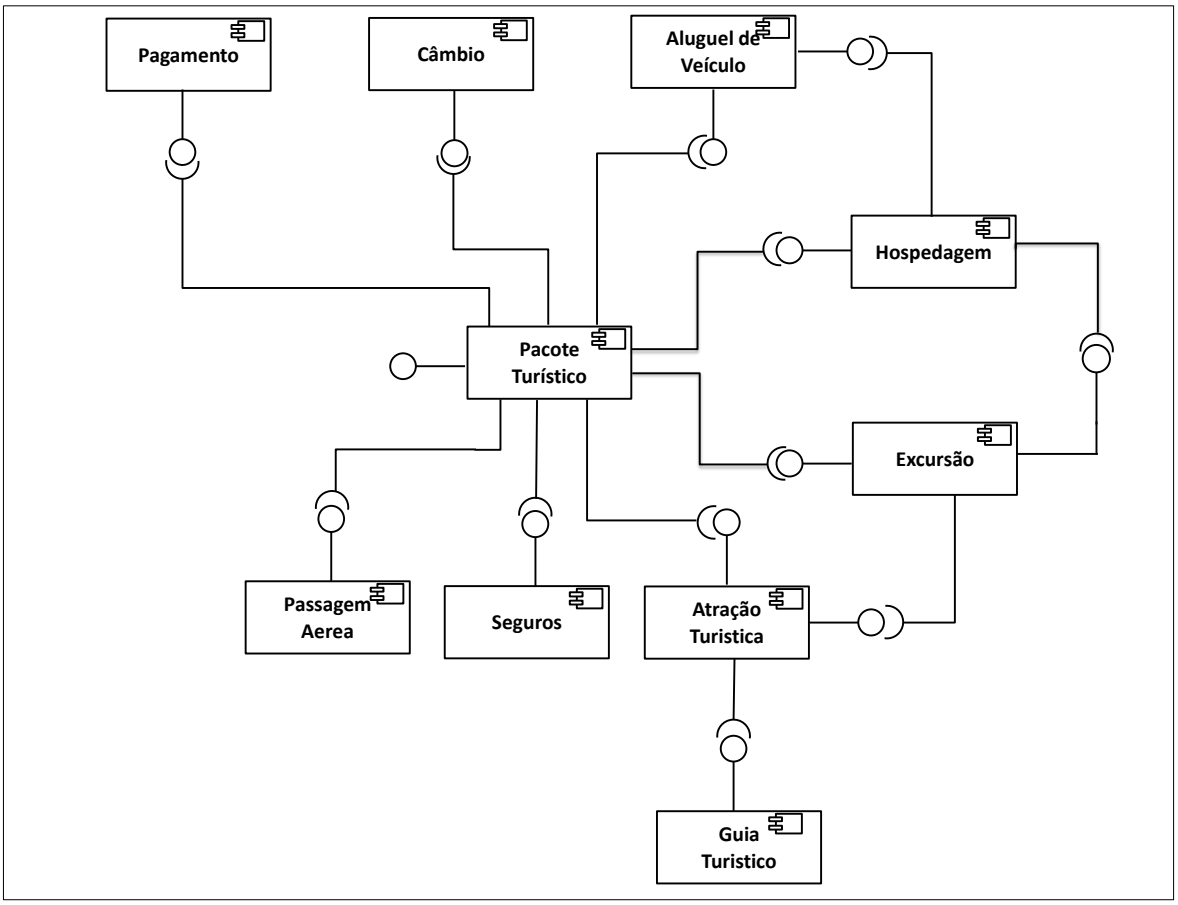

Figura 27 - Arquitetura com 10 serviços web

Neste experimento, fazendo uso de uma especificação com 10 serviços web, e 30 implementações candidatas, é gerado um espaço de busca de 30^{10}, o que equivale a 590.490.000.000.000 possíveis soluções. Este é um espaço de busca considerável para se utilizar uma metaheurística. A fim de mostrar que as soluções encontradas pela abordagem proposta são próximas à solução ótima, as soluções encontradas pelo algoritmo proposto são comparadas ao resultado apresentado pelo algoritmo de busca exaustiva. Uma seguida

comparação é feita com a busca aleatória com o objetivo de mostrar que a qualidade da solução encontrada pelo algoritmo genético proposto é bem superior.

Assim como no experimento anterior, a avaliação ocorre em três cenários diferentes, são eles: (*i*) todas as especificações na arquitetura possuem candidatos perfeitos; (*ii*) cinco especificações de serviços web possuem candidatos perfeitos; e (*iii*) ausência de candidatos perfeitos para as especificações pertencentes a arquitetura.

Apesar do espaço de busca gerado por este experimento ser bem maior que o anterior, ocorre em cada cenário a comparação entre a abordagem proposta e a busca exaustiva, como também a comparação entre a abordagem proposta e a busca aleatória. A intenção de executar o algoritmo de busca exaustiva neste problema é para encontrar a melhor solução para o problema, e assim compará-la com a solução proposta pelo algoritmo genético, e com isso evidenciar que o algoritmo proposto sempre encontra boas solução, ou até mesmo a solução ótima. Na comparação entre o algoritmo proposto e a busca aleatória, ambos foram executados 1000 vezes, e, em seguida, calculada a média do valor das soluções encontradas. Como mostra a Figura 28, os resultados experimentais revelam que a abordagem proposta encontra boas soluções, algo que é confirmada quando se realiza a comparação com a busca exaustiva.

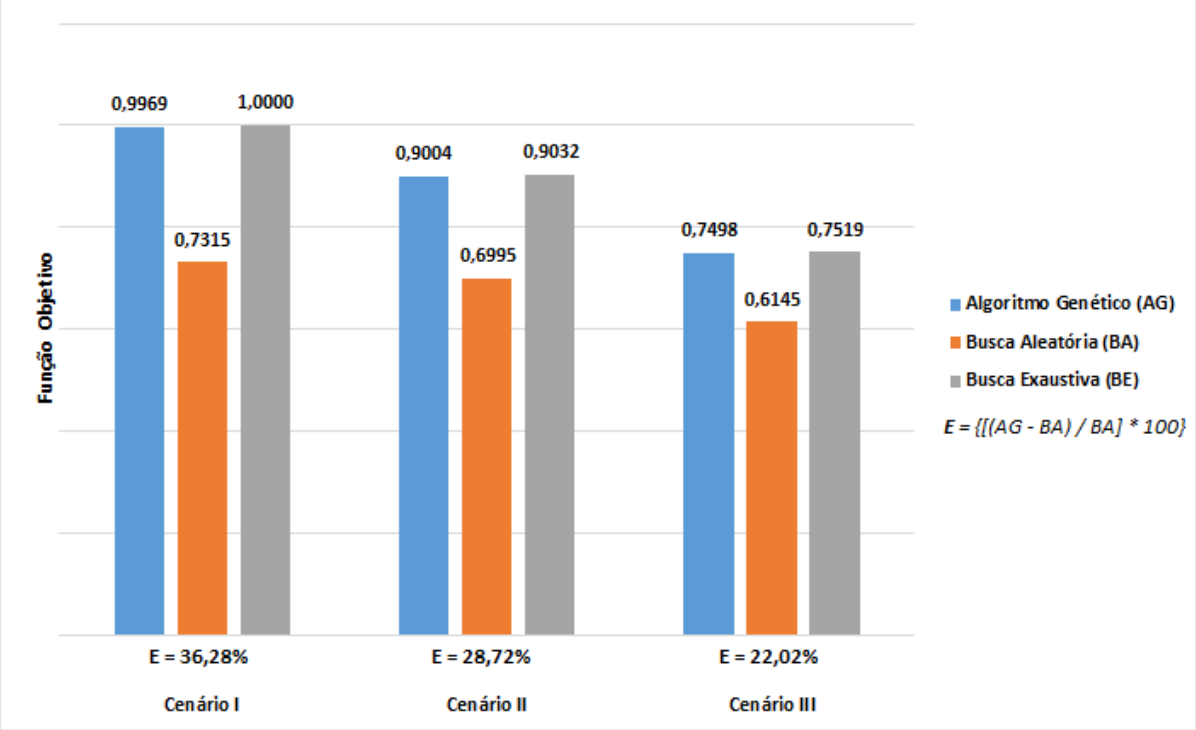

Figura 28 - Resultados experimentais para 10 serviços web

Conforme mostra a Figura 28, diferentemente do experimento anterior, aqui nem sempre o algoritmo genético proposto encontrou a melhor solução, mas a qualidade das soluções

encontradas sempre foi próxima à solução ótima. No entanto, vale ressaltar que, dentre as 1000 execuções realizadas, o algoritmo genético encontrou a solução ótima em 799 vezes no primeiro cenário, 719 vezes no segundo, e 617 vezes no terceiro cenário.

A qualidade das soluções geradas pelo algoritmo genético proposto garantiu no primeiro cenário, onde todas as especificações têm implementações perfeitas, um ganho de cerca de 36,28% em relação a solução recomendada pela busca aleatória. Já no segundo cenário, onde o experimento foi realizado com cinco especificações que possuem implementações perfeitas, a eficiência da abordagem proposta em relação a busca aleatória é aproximadamente 28,72%. No último cenário, na qual há uma ausência de candidatos que possuem implementações perfeitas, a eficiência da abordagem proposta torna-se estável em cerca de 22,02%. No melhor dos casos, a eficiência do algoritmo proposto em relação à busca aleatória é cerca de 38,92% no primeiro caso, 31,43% no segundo caso, e 25,22% no terceiro caso.

Outro resultado a ser levado em consideração é a rápida convergência do algoritmo proposto. Neste experimento, a quantidade de iterações necessárias para que o algoritmo encontre a melhor solução para o problema foi bastante baixo, levando em consideração que o número máximo de iterações é 1000. A Tabela 6 apresenta a quantidade de iterações necessárias para que o algoritmo genético proposto encontre uma boa solução em todos os cenários.

Tabela 6 - Número de iterações do AG para uma arquitetura com 10 serviços web

	Menor Iteração	Maior Iteração	Média Iteração
Cenário I	5	11	8
Cenário II	5	12	8
Cenário III	6	17	11

Por fim, assim como no experimento anterior, aqui também é comparado o tempo de processamento do algoritmo genético proposto com o algoritmo de busca aleatória. A Tabela 7 apresenta os tempos de processamento, representados em milissegundos. Como pode ser observado, o tempo do algoritmo proposto é um pouco maior que o tempo da busca aleatória em todos os cenários, porém, é uma perda irrelevante, quando comparada a qualidade das soluções apresentas pelo algoritmo genético proposto.

Tabela 7 - Tempo de Processamento para uma arquitetura com 10 serviços web

		Menor Processamento	Maior Processamento	Média de Processamento
Cenário I	***Algoritmo Proposto***	1177	3157	1279
	Busca Aleatória	765	1087	821
	Busca Exaustiva	-	-	17633538
Cenário II	***Algoritmo Proposto***	1201	1702	1309
	Busca Aleatória	764	1055	813
	Busca Exaustiva	-	-	15292023
Cenário III	***Algoritmo Proposto***	1221	1891	1313
	Busca Aleatória	779	998	834
	Busca Exaustiva	-	-	13163449

5.3.3. Arquitetura com 20 Especificações de Serviços Web

As seções anteriores apresentaram a utilização da abordagem proposta em um espaço de busca reduzido (5 especificações de serviços), e um espaço de busca ainda aceitável (10 especificações de serviços) para a utilização da busca exaustiva. Os resultados sugeridos pela abordagem proposta foram comparados com resultados da busca exaustiva e busca aleatória. Para evidenciar o ganho na qualidade da solução sugerida pelo algoritmo proposto em grandes espaços de busca, nesta seção é apresentado os resultados encontrados para a arquitetura que possui uma especificação com 20 serviços web e 24 dependências entre eles. A Figura 29 ilustra o diagrama UML que representa a arquitetura adotada no experimento.

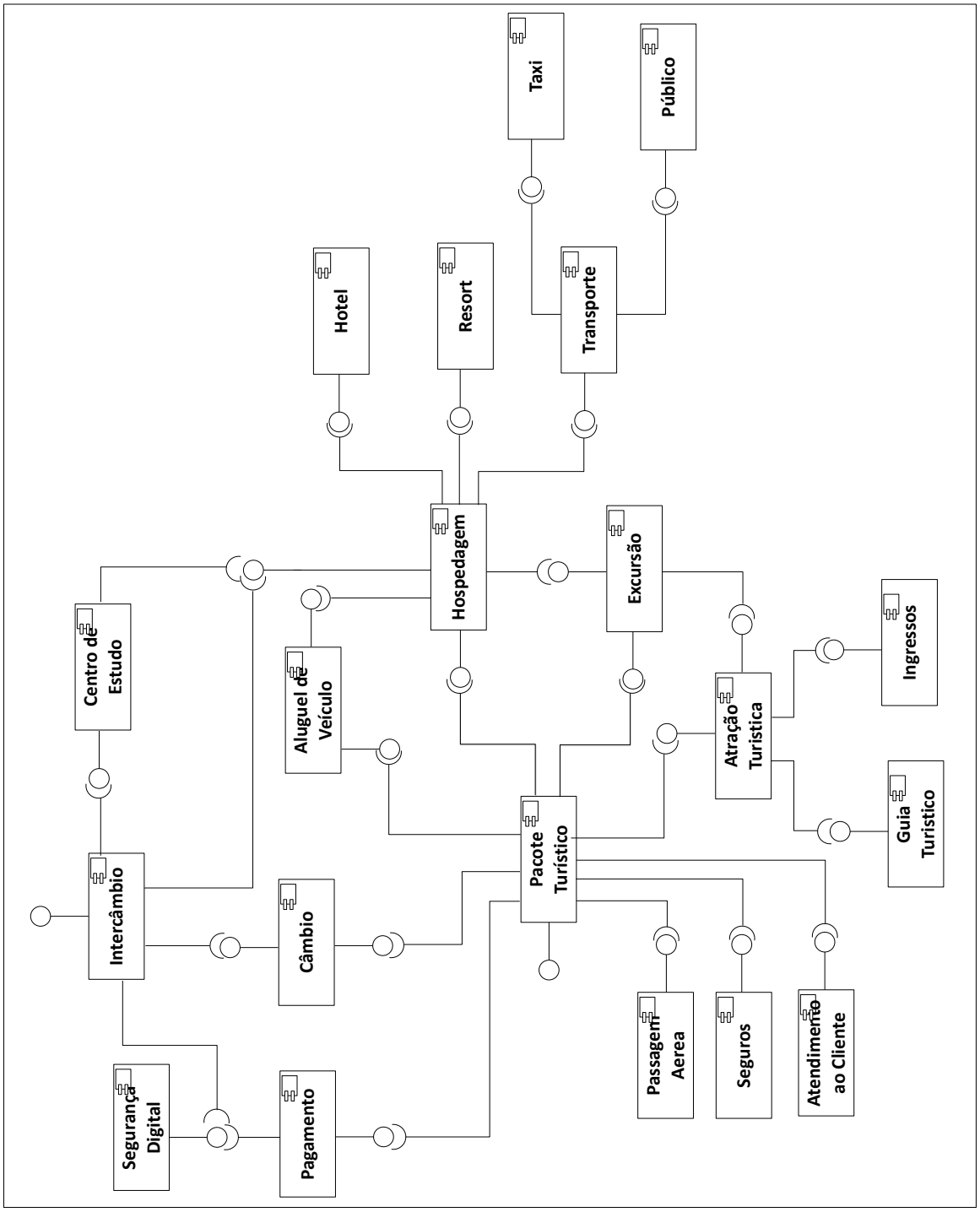

Figura 29 - Arquitetura com 20 serviços web

No experimento com 20 especificações de serviços web, com 30 implementações candidatas para cada especificação, é gerado um espaço de busca de 30^{20}. Como neste caso o espaço de busca é gigantesco, a comparação entre as soluções da abordagem proposta com a solução apontada pela busca exaustiva não será realizada, tendo em vista que executar o algoritmo de busca exaustiva para esse espaço é inviável, pois o tempo de execução do

algoritmo se torna inaceitável. Para exemplificar a inviabilidade, vamos utilizar a execução realizada para um espaço de busca de 30^{10}, tendo o conhecimento que tal execução teve a duração de um pouco mais de 4 horas, e que o espaço de busca com tamanho de 30^{20} é cerca de 590 trilhões de vezes maior que o anterior. Portanto, por analogia, realizar a execução da busca exaustiva para o espaço de busca de 30^{20} levaria cerca de 330 bilhões de anos. No entanto, a comparação com a busca aleatória é realizada.

Da mesma forma que nos outros dois experimentos anteriores, essa avaliação ocorre em três cenários diferentes, são eles: (*i*) todas as especificações de serviços da arquitetura possuem candidatos perfeitos; (*ii*) dez especificações possuem candidatos perfeitos; e (*iii*) ausência de candidatos perfeitos para as especificações pertencentes a arquitetura. Para cada cenário, a abordagem proposta foi comparada com a busca aleatória. Em tal comparação, a abordagem proposta e a busca aleatória foram executadas 1000 vezes, e o valor médio das soluções é calculado.

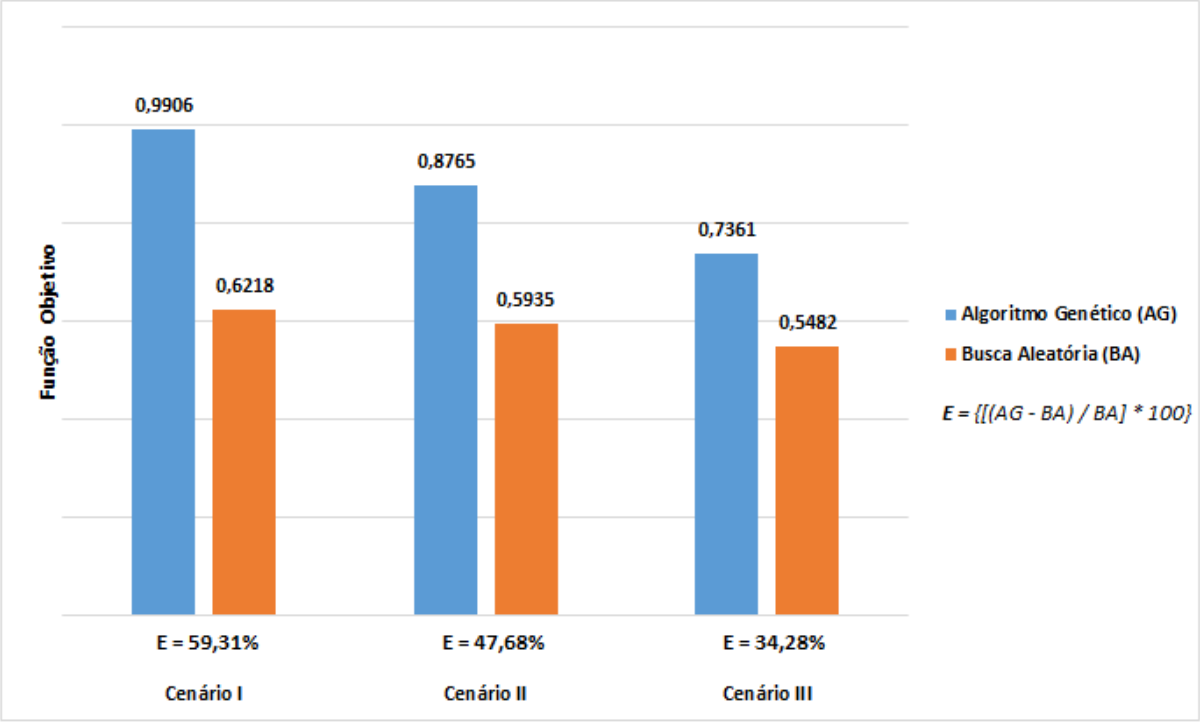

Figura 30 - Resultados experimentais para 20 serviços web

Mesmo com o espaço de busca com tamanho de 30^{20}, o valor da função objetivo apresentado na Figura 30 evidencia que o algoritmo proposto tem encontrado soluções muito boas, principalmente quando comparadas com as apresentadas pela busca aleatória. No primeiro cenário do experimento, o AG encontrou soluções com o mesmo valor de *fitness* (0,9993) em 258 das 1000 oportunidades possíveis, tendo em vista que o valor da função

objetivo possui seu valor no intervalor entre 0 e 1, é possível considerar que o algoritmo encontrou boas soluções. Já no segundo cenário, a quantidade de vezes que o algoritmo proposto encontrou boas soluções com o mesmo escore (0,8849) diminui para 116, no entanto, considerando o tamanho do espaço de busca, esse é um valor considerável, e que mostra a eficiência do algoritmo. No terceiro cenário, a abordagem proposta encontra em 102 oportunidades das 1000 possíveis soluções com o mesmo valor de *fitness* (0,7421).

Realizando um comparativo, é possível afirmar que a solução gerada pelo algoritmo genético proposto é cerca de 59,31% melhor que a solução recomendada pela busca aleatória no primeiro cenário. No segundo cenário, onde há dez especificações que possuem implementações perfeitas, a eficiência da abordagem proposta comparada com a busca aleatória é aproximadamente 47,68%. No último cenário, com a ausência de candidatos que possuem implementações perfeitas, a eficiência da abordagem proposta é cerca de 34,28%. Destacando os ganhos no melhor dos casos, a eficiência do algoritmo proposto em relação à busca aleatória é cerca de 72,65%, 59,93%, e 42,66% nos três cenários, respectivamente. Outro destaque bastante interessante neste experimento, é que a melhor solução encontrada pela busca aleatória é inferior a pior solução encontrada pela abordagem proposta. Este fato ocorre em todos os cenários.

Uma boa escolha dos operadores de mutação e cruzamento, e utilização da técnica de elitismo impacta diretamente no custo de processamento do algoritmo. O resultado desta boa escolha pode ser percebido não apenas pelo baixo tempo de processamento, mas também pela rápida convergência do algoritmo encontrar boas soluções. A Tabela 8 apresenta a quantidade média de iterações necessárias para o algoritmo genético encontrar boas soluções.

Tabela 8 - Número de iterações do AG para uma arquitetura com 20 serviços web

	Menor Iteração	Maior Iteração	Média Iteração
Cenário I	9	16	13
Cenário II	11	18	14
Cenário III	13	23	18

Como dito, o tempo de processamento do algoritmo proposto é muito baixo, tendo em vista a qualidade das soluções apresentadas. A Tabela 9 apresenta os tempos de processamento, representados em milissegundos. Como pode ser observado, o tempo do algoritmo proposto é cerca de 40% maior que o tempo da busca aleatória em todos os cenários, porém, a qualidade da solução gerada supera facilmente esse ponto.

Tabela 9 - Tempo de processamento para uma arquitetura com 20 serviços web

		Menor Processamento	Maior Processamento	Média de Processamento
Cenário I	***Algoritmo Proposto***	3046	4547	3213
	Busca Aleatória	1930	2772	2065
Cenário II	***Algoritmo Proposto***	3223	4044	3389
	Busca Aleatória	1983	2749	2090
Cenário III	***Algoritmo Proposto***	3335	4625	3607
	Busca Aleatória	2032	2652	2152

5.3.4. Arquitetura com 30 Especificações de Serviços Web

Com o objetivo de comprovar que quanto maior o espaço de busca, melhor será a eficiência da abordagem proposta quando comparada com uma busca aleatória, esta seção apresenta os resultados encontrados para a arquitetura que possui uma especificação com 30 serviços web com 36 dependências. Assim como em todos os experimentos anteriores, se faz necessário uma arquitetura como modelo para concretizar o ambiente experimental. A Figura 31 ilustra tal arquitetura em forma de um diagrama UML, contemplando as 30 especificações.

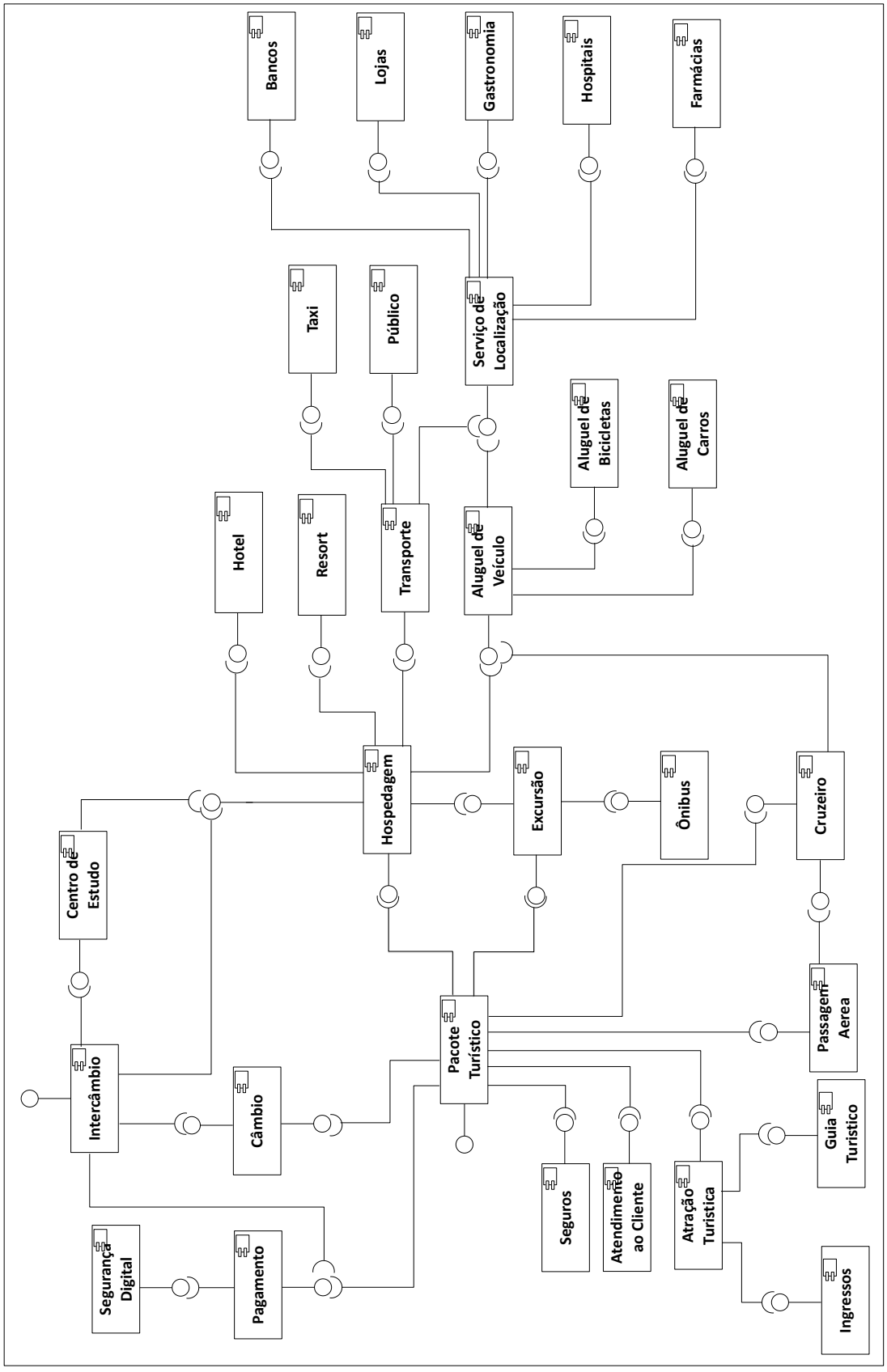

Figura 31 - Arquitetura com 30 serviços web

Para o experimento com 30 especificações de serviços web, sendo 30 implementações candidatas para cada especificação, é gerado um espaço de busca com 30^{30} possíveis soluções. Como dito anteriormente, para esse experimento, a solução apresentada pela abordagem proposta não será comparada com a solução apresentada pela busca exaustiva, tendo em vista que executar o algoritmo de busca exaustiva neste cenário é inviável e irreal, uma vez que o tempo de execução do algoritmo se torna intolerável. No entanto, a comparação com a busca aleatória será realizada para confirmar a eficiência do algoritmo genético em encontrar boas soluções em gigantescos espaços de busca.

Assim como em todos os experimentos realizados neste trabalho, a avaliação ocorre em três cenários diferentes, são eles: (*i*) todas as especificações de serviços web da arquitetura possuem candidatos perfeitos; (*ii*) quinze especificações possuem candidatos perfeitos; e (*iii*) ausência de candidatos perfeitos para as especificações. Para cada cenário, afim de realizar a comparação entre as soluções apresentadas pelo algoritmo proposto com as apresentadas pela busca aleatória, ambos os algoritmos foram executados 1000 vezes, e calculado uma média das soluções encontradas. A Figura 32 evidencia o que fora dito anteriormente, afirmando que quanto maior o espaço de busca, melhor é a eficiência da abordagem proposta em comparação com a busca aleatória.

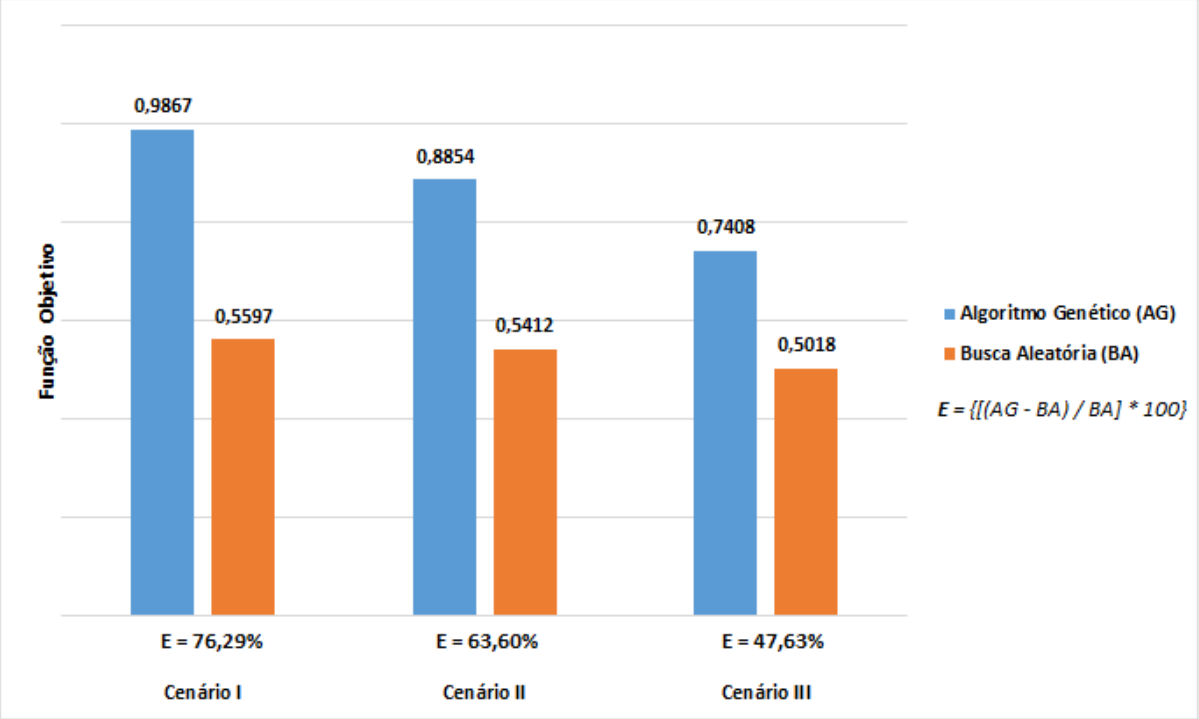

Figura 32 - Resultados experimentais para 30 serviços web

Comparando as soluções encontradas por ambos os algoritmos, no primeiro cenário do experimento, a configuração arquitetural apresentada pelo algoritmo genético proposto é, em média, 76,29% melhor que a apresentada pela busca aleatória. Uma característica do algoritmo proposto neste primeiro cenário é que em cerca de 10% (102 das 1000 possíveis) das execuções, o algoritmo encontrou soluções com o mesmo escore (0,9980). Para o segundo cenário desse experimento, onde há quinze especificações que possuem implementações perfeitas, a abordagem proposta tem uma eficiência de aproximadamente 63,60% quando comparada com a busca aleatória. Por fim, no terceiro cenário, onde existe uma ausência de candidatos que possuem implementações perfeitas, a eficiência da abordagem proposta é cerca de 47,63%. A capacidade do algoritmo proposto em encontrar boas soluções proporcionou no melhor dos casos, uma eficiência em torno de 88,04%, 74,76%, e 57,23% nos três cenários. Outra particularidade desde experimento é que a melhor solução encontrada pela busca aleatória, em todas as execuções, é inferior a pior solução encontrada pela abordagem proposta.

Como em todos os experimentos realizados neste trabalho, neste quarto experimento, o algoritmo genético se mostrou muito eficiente em encontrar boas soluções com um tempo de processamento baixo, e, mediante uma boa escolha dos operadores de mutação e cruzamento, acrescido da utilização da técnica de elitismo, uma rápida convergência para encontrar boas soluções. A Tabela 10 apresenta a média de iterações necessárias para o algoritmo genético encontrar boas soluções neste experimento.

Tabela 10 - Número de iterações do AG para uma arquitetura com 30 serviços web

	Menor Iteração	Maior Iteração	Média Iteração
Cenário I	13	20	17
Cenário II	15	23	18
Cenário III	17	30	23

Como abordado anteriormente, o tempo de processamento do algoritmo proposto é baixo. Somando essa característica a qualidade da solução apresentada, é possível afirmar que o AG é eficiente em encontrar boas soluções, ou até mesmo a solução ótima em gigantescos espaços de busca, e com um tempo de processamento aceitável. A Tabela 11 apresenta os tempos de processamento representados em milissegundos. Como pode ser observado, o tempo de processamento do algoritmo genético é em média 34% maior que o tempo da busca aleatória em todos os cenários. No entanto, essa fragilidade é facilmente superada pela qualidade da solução apresentada pela abordagem proposta.

Tabela 11 - Tempo de processamento para uma arquitetura com 30 serviços web

		Menor Processamento	Maior Processamento	Média de Processamento
Cenário I	***Algoritmo Proposto***	084	7705	5288
	Busca Aleatória	3059	5397	3488
Cenário II	***Algoritmo Proposto***	4661	6074	4986
	Busca Aleatória	3113	4058	3263
Cenário III	***Algoritmo Proposto***	5367	7273	5825
	Busca Aleatória	3193	4811	3382

5.4. Considerações Finais

As avaliações de cada um dos experimentos mostraram que a abordagem proposta encontrou sempre boas soluções, e, nos cenários com menor espaço de busca, na maioria das vezes encontrou a solução ótima. Dessa forma, tem-se a percepção de que o algoritmo genético proposto é guiado de forma consistente por sua função objetivo, e evoluem suas soluções por meio da seleção e dos operadores genéticos. Contudo, é importante verificar se as soluções encontradas pelo AG são equiparaveis a solução ótima.

Para atender esse requisito, foram realizados os experimentos com uma arquitetura com 5 e 10 especificações de serviços web. No experimento com a arquitetura com 5 serviços web, foi comprovado por meio da comparação com a busca exaustiva que a abordagem proposta sempre encontra a solução ótima. Como esse experimento possui um espaço de busca reduzido, o mesmo foi realizado para termos garantia na qualidade das soluções apresentadas pelo algoritmo proposto. Para realizarmos o experimento em um espaço de busca maior, foi utilizada uma arquitetura com 10 especificações, algo que aumentou o espaço de busca de forma exponencial. Mais uma vez, foi comprovado por meio da comparação com a busca exaustiva que a abordagem proposta apresenta boas soluções quando comparadas com a solução apresentada pela busca aleatória, chegando a encontrar a solução ótima em diversos casos.

No que diz respeito aos experimentos com 20 e 30 especificações de serviços, também foi comprovado a eficiência do algoritmo proposto em grandes espaços de busca. Outro ponto bastante interessante a ser abordado é que, à medida em que o espaço de busca aumenta,

melhora a eficiência do algoritmo genético em encontrar boas soluções, principalmente quando comparadas com a busca aleatória.

Capítulo 6

Conclusão

Neste trabalho foi apresentada uma proposta para uma Seleção Automatizada de Serviços Web Orientada por Métricas Funcionais e Estruturais. Conforme foi mencionado, o processo de seleção de serviços web é uma tarefa muito complexa para ser resolvida apenas com a experiência e intuição dos membros da equipe de desenvolvimento, sendo necessário um auxílio automatizado para minimizar o esforço de adaptação e integração nas escolhas dos serviços web que constituirão um sistema de software.

Diferentemente do que normalmente é evidenciado nas propostas relacionadas, que em geral consideram apenas atributos de qualidade relacionados com requisitos não funcionais para realizar a seleção de serviços, a abordagem proposta consiste no processo de seleção avaliando métricas funcionais e estruturais, associadas com requisitos funcionais dos serviços web. As métricas funcionais abordadas neste trabalho avaliam o quão similar é o serviço implementado do especificado. Esta similaridade é medida a nível de operações e interfaces do serviço. No que diz respeito as métricas estruturais, elas consideram características arquiteturais dos serviços, medindo o esforço de adaptação e integração entre os candidatos para compor os sistemas de software. Na proposta apresentada, a automatização desse processo de seleção é possível pela utilização de um algoritmo genético, possibilitando avaliar um problema comprovadamente complexo e com muitas soluções possíveis, e assim oferecer a equipe de desenvolvimento uma série de configurações arquiteturais para o problema em questão.

A concepção e o delineamento do trabalho utilizam como suporte à abordagem de Engenharia de Software Baseada em Buscas, onde a mesma serve como um guia para reformular problemas da Engenharia de Software em problemas de busca com complexidade não polinomial, visando obter boas soluções ou até mesmo a solução ótima. Essa abordagem é considerada relativamente recente e emergiu com a motivação de explorar novas alternativas desse campo de pesquisa, pois a tendência é que os produtos e processos de software se tornem cada vez mais complexos, que suas escalas aumentem em uma proporção cada vez maior, e as metodologias de desenvolvimento desses produtos devem se adaptar às novas fronteiras alcançadas.

Os modelos atuais de seleção de serviços web possuem a deficiência de não tratarem as relações funcionais e estruturais dos serviços, em outras palavras, somente se concentram em requisitos não funcionais e/ou atributos de QoS. A perspectiva de desenvolvimento baseado em SOA remete a utilização de serviços web já implementados para construir um sistema, serviços esses, que estão distribuídos em repositórios para serem consumidos. Como cada serviço web é desenvolvido por terceiros, com modelos e padrões próprios, resultando em inconsistências no momento da integração entre os serviços web selecionados para compor o sistema, é muito importante prever os problemas de dependência entre os serviços web, afim de não comprometer o custo e o tempo de desenvolvimento do sistema.

Após concepção de métricas funcionais e estruturais, a fim de realizar uma avaliação das configurações arquiteturais candidatas, tem-se a preocupação de observar se tais métricas realmente estão guiando o algoritmo de otimização desenvolvido para tratar o problema de seleção de serviços web, assim como mostrar que o esforço utilizado na elaboração desse algoritmo foi totalmente recompensado pela eficiência do mesmo em encontrar boas soluções, principalmente quando comparadas com resultados apresentados por abordagens mais simples, como a busca aleatória. Dessa forma, é importante realizar a avaliação do algoritmo desenvolvido para evidenciar a qualidade dos resultados apresentados.

O primeiro estudo de caso, possui um modelo de arquitetura composta por 5 especificações de serviços web. Neste experimento, o algoritmo proposto mostrou-se extremamente eficiente, pois encontrou em todos os casos a solução ótima, algo que foi comprovado quando teve suas soluções comparadas com a busca exaustiva. No segundo estudo de caso, a arquitetura é composta por 10 especificações de serviços web. Aqui, o algoritmo proposto desempenhou certa equivalência qualitativa em comparação com um algoritmo de busca exaustiva, e quando comparadas as soluções com a busca aleatória, mostrou eficiência superior.

Para o terceiro estudo de caso, foi elaborada uma arquitetura com 20 especificações de serviços web. Assim como nos estudos de casos anteriores, os experimentos realizados neste estudo de caso mostraram que a abordagem proposta é bastante eficiente, algo que pode ser constatado pela qualidade das soluções apresentas. Por fim, foi realizado o quarto e último estudo de caso com uma arquitetura que possui 30 especificações de serviços web. Neste experimento, a abordagem proposta mostrou toda sua eficiência em apresentar boas soluções em espaços de busca gigantescos. Neste último experimento, foi evidenciado, por meio de comparativos das soluções, que o algoritmo genético proposto é bem mais eficiente que o

algoritmo de busca aleatória para encontrar boas soluções à medida que o espaço de busca aumenta.

6.1. Limitações

A falta da realização de alguns testes estatísticos pode ser considerada como uma limitação para o presente trabalho. A aplicação desse tipo de teste serve para confirmar a eficiência do algoritmo proposto em encontrar boas soluções quando comparadas com as apresentadas pela busca aleatória. Outra limitação do trabalho é a falta da indicação de uma ferramenta que auxilie a equipe de desenvolvimento em buscar por serviços candidatos em um repositório de serviços. Neste ponto, é importando frisar que a falta de um repositório real é sem dúvida uma outra limitação do trabalho.

Como os experimentos realizados neste trabalho não foram realizados com base em arquiteturas de sistemas reias, exceto o experimento com a arquitetura com 5 serviços, outro limitação do trabalho é a não utilização de arquiteturas de sistemas reias para realização de tais experimentos.

6.2. Trabalhos Futuros

Como um trabalho futuro interessante, é importante a realização de alguns testes estatísticos para dar uma maior credibilidade aos resultados encontrados, onde se utiliza conceitos estatísticos para aceitar ou rejeitar uma hipótese nula. A hipótese nula geralmente condiz com a asserção de que não há diferenças entre dois algoritmos, e o teste indicará se deve aceitar ou rejeitar tal hipótese. A intenção desse teste é demonstrar que o algoritmo proposto sobrepõe a qualidade do algoritmo de busca aleatória. Outro teste a ser realizado é o cálculo do intervalo de confiança, que é uma forma de calcular a probabilidade que um evento ocorra dentro de um determinado intervalo. Esse cálculo é feito a partir de amostras de uma população.

Seria importante e prática a criação de uma ferramenta que auxilie a equipe de desenvolvimento em buscar serviços web candidatos em repositórios de serviços a partir do diagrama de arquitetura produzido no projeto arquitetural. Com isso, após as concepções das especificações de serviços no diagrama de arquitetura, as consultas seriam construídas automaticamente na linguagem estabelecida pelo serviço de busca, e assim, poupar o esforço ou até o esquecimento do integrante da equipe de desenvolvimento em consultar algum serviço web candidato específico. A entrada para essa ferramenta seria um conjunto de especificações

de serviços web pertencentes à arquitetura do software a ser construído, e a saída seria os resultados de tais consultas com as descrições WSDL e os modelos de diagramas de sequência. Em seguida, tais artefatos são processados por uma ferramenta, na qual identifica todas as informações (interfaces providas e requeridas) dos candidatos, e as disponibiliza por meio de um arquivo (XML ou CSV) para serem processadas pela metaheurística desenvolvida neste trabalho.

O primeiro experimento realizado neste trabalho, conta com uma especificação de serviços real, no entanto as implementações candidatas, foram geradas de forma aleatória conforme descrito. Baseado nisso, um trabalho futuro a ser realizado, é a realização de testes em um ambiente real, onde todas as especificações e implementações candidatas são reais.

Conforme discutido nos trabalhos relacionados, a maioria dos trabalhos que tratam a seleção de serviços, o fazem baseado em requisitos não funcionais e atributos de QoS. Um trabalho futuro seria a seleção de serviços que adote todas as métricas discutidas neste trabalho, ou seja, uma seleção baseada em métricas funcionais e estruturais, adotando atributos de QoS. Esse sem dúvida é um trabalho bastante interessante a ser desenvolvido, tendo em vista que tais métricas se complementam e se relacionam em um processo de desenvolvimento com serviços web.

O ambiente desenvolvido neste trabalho de dissertação utiliza serviços web baseado em SOAP, um trabalho futuro bastante proveitoso, seria a adaptação da atual arquitetura para um modelo arquitetural baseado em REST, visando que tal modelo vem se tornando tendência no mercado para desenvolvimento de sistemas de software com base em serviços web. Para utilizar os conceitos da presente proposta, se faz necessário apenas um processo para identificar as interfaces requeridas do serviço, já que a interface provida é possível ser recuperada por meio da especificação do serviço e sua API. As demais etapas do processo de seleção de serviços seguem da mesma forma como apresentado nesse trabalho.

Outro ponto bastante interessante a ser desenvolvido futuramente, seria a aplicabilidade do conceito de Ontologias no domínio da Engenharia de Software, mais precisamente na seleção de serviços web, tendo em vista que a sintaxe utilizada por uma especificação de serviço pode ser diferente da sintaxe utilizada por uma implementação de serviço candidato, mesmo quando ambas possuem a mesma semântica, ou seja, possuem assinaturas de métodos diferentes, no entanto possuem o mesmo comportamento funcional.

Por fim, o presente trabalho buscou investigar novos caminhos para melhorar a qualidade do processo de seleção de serviços web, como realizar uma avaliação estrutural da

arquitetura do software a ser desenvolvido, como também uma avaliação de similaridade entre a especificação e a implementação do serviço candidato. A utilização da Engenharia de Software Baseada em Buscas mostrou que é um caminho a se seguir e realizar pesquisas mais a fundo, já que essa abordagem ainda é considerada emergente.

Referências

Alexandre J. M. **Seleção Automatizada de Componentes de Software Orientada por Métricas Estruturais e Informações de Reúso**, 2013. Dissertação de Mestrado - Universidade Federal da Paraíba.

Ali S.; Briand L.; Hemmati H.; Panesar W. R. **A systematic review of the application and empirical investigation of search-based test-case generation**. IEEE Transactions on Software Engineering (TSE), 2009.

Arcuri A; Briand L. **A practical guide for using statistical tests to assess randomized algorithms in software engineering**. Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), 2011. p. 1-10.

Back T.; Ogel D.; Michalewicz Z. **Evolutionary Computation 1 Basic Algorithms and Operators**. Bristol and Philadelphia: Institute of Phisics Publishing, 2000.

Barros M.; Arilo C. D. **A Survey of Empirical Investigations on SSBSE Papers**. International Symposium on Search Based Software Engineering, SSBSE 2011.

Barzilay O.; Urquhart C. **Understanding reuse of software examples: A case study of prejudice in a community of practice**, 2014. Information and Software Technology 1613–1628.

Becker S.; Brogi A.; Gorton I.; Overhage S.; Romanovsky A.; Tivoli M. **Towards an Engineering Approach to Component Adaptation**. Springer-Verlang, 2006.

Bobby W. **Introduction to SOA governance**, 2006.

Burke E.; Harman M.; Clark J. A.; Yao X. **Dynamic Adaptive Search Based Software Engineering. Empirical Software Engineering and Measurement (ESEM)**, ACM-IEEE International Symposium, 2012.

Cheesman J.; Daniels J. **UML Components: A Simple Process for Specifying Component-Based Software**. Boston, 2001.

Clarke J; Dolado J. J.; Harman M.; Hierons R.; Jones B.; Lumkin M.; Mitchell B.; Mancoridis S.; Rees K.; Roper M.; Shepperd M. **Reformulating software engineering as a search problem**. IEEE Proceedings – Software, 2003. Vol. 150, Iss. 3, p. 161-175.

Colanzia T. E.; Vergilio S. R.; Guez W. K.; Pozoa A. **Search Based Software Engineering: Review and analysis of the field in Brazil**, 2013. Journal of Systems and Software Volume 86, Issue 4, p. 970–984.

Dan A.; Johnson R. D.; Carrato T. **SOA Service Reuse by Design**, 2008.

DeRose S.; Maler E.; Orchard D. **XML Linking Language (XLink) Version 1.0 W3C Recommendation**. World Wide Web Consortium, 2001.

DIAS J.; ALMEIDA E.; MEIRA S. **The Analysis Activity in a Systematic SOA-based Architecture Process**. 2010. 14th IEEE International Enterprise Distributed Object Computing Conference Workshops.

Dijkstra E. **The Structure of the 'THE' Multiprogramming System**. 1968. Communications of the ACM 11, no. 5: 341-346.

Dirk K.; Karl B.; Dirk S. **Enterprise SOA. Service-Oriented Architecture Best Practices 1 ed. Prentice Hall (S.l.)**, 2004.

Egyed A.; Medvidovic N.; Gacek C. **Component-Based Perspective on Software Mismatch Detection and Resolution**, 2000. IEE Proceedings – Software, vol. 147, No. 6, p. 225–236.

Erl T SOA with REST: **Principles, Patterns & Constraints for Building Enterprise Solutions with REST**, 2012. 1st Edition.

Erl T. **Service-Oriented Architecture (SOA): Concepts, Technology, and Design**, 2005.

Erl T. **Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services**, 2004.

Erl T. **SOA: Principles of Service Design 1st Edition**, 2007

Fielding R. **Architectural styles and the design of networked-based software architectures**, 2000. Dept. of Information and Computer Science, University of California.

Fielding R.; Taylor N. **Principle Design of the Modern Web Architecture ACM Transactions on Internet Technology**, 2002 p. 115–150.

Forrest S. **Genetic algorithms**, 1996. ACM Computing Surveys.

Freitas F. G.; Maia C. L. B.; Coutinho D. P.; Campos G. A. L.; Souza J. T. **Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de Literatura**, 2009. II Congresso Tecnológico Infobrasil.

Garlan D.; Shaw M. **An Introduction to Software Architecture**, 1994. Advances in Software Engineering.

Germn H.; **Systematic Reuse of Web Services through Software Product Line Engineering**, 2011. Ninth IEEE European Conference on Web Services

Goldberg D.; Deb K. **A Comparative Analysis of Selection Schemes Used in Genetic Algorithms**, 1991. Foundations of Genetic Algorithms, Morgan Kaufmann, USA.

Griss P.; Jacobson I. **Software Reuse**, 1997.

Hadjila F.; Mohammed A.; Dali Y. **QoS-aware Service Selection Based on Genetic Algorithm**, 2011. 16th International Conference, Hong Kong, China.

Harman M. **Search based software engineering**, 2006. 6th International Conference on Computational Science, p. 740-747.

Harman M. **The Current State and Future of Search Based Software Engineering**, 2007. Proceedings of the Future of Software Engineering.

Harman M. **Why the Virtual Nature of Software Makes It Ideal for Search Based Optimization**, 2010. Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering, Vol. 6013, p. 1-12.

Harman M.; Clark J. **Metrics are fitness functions too**, 2004. In 10th International Software Metrics Symposium (METRICS 2004), p. 58–69. IEEE Computer Society Press.

Harman M.; Mansouri A.; Zhang Y. **Search-based software engineering: Trends, techniques and applications**, 2012. Journal ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 45 Issue 1, Article No. 11.

Holland J. H. **Adaptation in Natural and Artificial Systems**, 1975. University of Michigan Press, USA.

IEEE **Introduction to the Special Issue on Software Architecture**, 2006.

Jiang H.; Ren Z.; Li X.; Lai X. **Transformed Search Based Software Engineering: A New Paradigm of SBSE**. 7th International Symposium, SSBSE 2015, Bergamo, Italy, September 5-7, 2015.

Katawut K.; Sarun I. **QoS Attributes of Web Services: A Systematic Review and Classification**, 2015. Journal of Advanced Management Science Vol. 3.

Kavantzas N.; Burdett D.; Ritzinger G. **Web Service Choreography Description Language**, 2004.

Kazman R.; Bass L.; Abowd G.; Webb M. **Analyzing the Properties of User Interface Software**, 1993.

Kruchten P. **The view model of software architecture**, 1995. IEEE Software, v12, n. 6, p 42-50.

Len B.; Paul C.; Rick K. **Software Architecture in Practice**, 2012. 3º Edition.

Levelock C.; Vandermerwe S.; Lewis B. **Services Marketing. Englewood Cliffs, NJ: Pretice Hall**, 1996.

Lifeng A.; Maolin T.; George S.; Brisbane.; **A Penalty-based Genetic Algorithm for QoS-AwareWeb Service Composition with Inter-Service Dependencies and Conflicts**, 2008.

Linden R. **Algoritmos Genéticos: Uma Importante Ferramenta da Inteligência Computacional**, 2008. 2ª Edição.

Lublinsky B. **Defining SOA as an architectural style**, 2007.

Maolin T.; Lifeng A. **A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition**, 2010. In: Proceeding of the 2010 World Congress on Computational Intelligence.

Matoušek R. **Genetic Algorithm and Advanced Tournament Selection Concept**, 2008. Nature Inspired Cooperative Strategies for Optimization, Studies in Computational Intelligence, Volume 236.

Mojtaba K.; Ali J.; Wan M.; Suhaimi I. **An Approach for Web Service Selection Based on Confidence Level of Decision Maker**, Discover a Faster Path to Publishing in a High-Quality journal, 2014.

Newcomer E.; Lomow G. **Understanding SOA with Web Services**. 2005.

OASIS **Reference Architecture Foundation for Service Oriented Architecture Version 1.0**, 2012. Committee Specification 1.

Oliveira J.; Maldonado J.; Gimenes I. **Uma Revisão Sistemática sobre Avaliação de Linha de Produto de Software**, 2010.

Pegah M.; Jafar H.; Touraj V. **Application of Social Harmony Search Algorithm on Composite Web Service Selection based on Quality Attributes**, 6ª International Conference on Genetic and Evolutionary Computing, 2012.

Raghu R. **What is service-oriented architecture**, 2005.

Razali N. M.; Geraghty J. **Genetic Algorithm Performance with Different Selection Strategies in Solving TSP**, 2011. Proceedings of the World Congress on Engineering, Vol II.

Schmidt D.; Stal M.; Rohnert H.; Buschmann F. **Pattern-Oriented Software Architecture**, 2000. Patterns for Concurrent and Networked Objects. Nova York: John Wiley & Sons.

Shaw M.; Garlan D. **Software Architecture: Pespectives on an Emerging Discipline**, 1996.

Tuner M.; Budgen D.; Brereton P. **Turning Software into a Service. IEEE computer**, 2003. v36, p. 38-45.

Vergilio S. R.; Colanzi E.; Trinidad A.; Klewerton W. **Search Based Software Engineering: A Review from the Brazilian Symposium on Software Engineering**, 2011. 25th Brazilian Symposium on Software Engineering.

Wolpert D. H.; Macready W. G. **No Free Lunch Theorems for Optimization**, 1997. IEEE Transactions on Evolutionary Computation, Vol. 1.

Xie H.; Zhang M. **Sampling Issues of Tournament Selection in Genetic Programming**, 2009. Technical Report Series, School of Engineering and Computer Science, Victoria University of Wellington, New Zealand.