PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

AURÉLIO DA SILVA GRANDE

UM FRAMEWORK DE APOIO À INSTANCIAÇÃO DE TÉCNICAS DE SELEÇÃO DE TECNOLOGIAS DE SOFTWARE BASEADAS EM ESTRATÉGIAS DE BUSCA

Manaus-AM
Março de 2013

AURÉLIO DA SILVA GRANDE

UM FRAMEWORK DE APOIO À INSTANCIAÇÃO DE TÉCNICAS DE SELEÇÃO DE TECNOLOGIAS DE SOFTWARE BASEADAS EM ESTRATÉGIAS DE BUSCA

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas, como parte dos requisitos necessários para a obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Arilo Cláudio Dias Neto

Manaus-AM
Março de 2013

AURÉLIO DA SILVA GRANDE

UM FRAMEWORK DE APOIO À INSTANCIAÇÃO DE TÉCNICAS DE SELEÇÃO DE TECNOLOGIAS DE SOFTWARE BASEADAS EM ESTRATÉGIAS DE BUSCA

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas, como parte dos requisitos necessários para a obtenção do título de Mestre em Informática.

Banca Examinadora

_____

Prof. Dr. Arilo Cláudio Dias Neto (Orientador)
Universidade Federal do Amazonas (UFAM)

_____

Prof. Dra. Rosiane de Freitas Rodrigues
Universidade Federal do Amazonas (UFAM)

_____

Prof. Dr. Márcio de Oliveira Barros
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

Manaus
Março de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Grande, Aurélio da Silva.

Um framework de apoio à instanciação de técnicas de seleção de tecnologias de software baseadas em estratégias de busca / Aurélio da Silva Grande. Manaus: IComp da UFAM, 2013.

XVI, 121 f.: il.; 29,7 cm.

Orientador: Arilo Cláudio Dias Neto

Dissertação (mestrado) – Universidade Federal do Amazonas. Instituto de Computação, Manaus, BR – AM, 2013.

Referências Bibliográficas: p. 106-117.

1. Seleção de Tecnologia. 2. Meta-Heurística 3. SBSE. 4. Framework. I. Dias-Neto, Arilo Cláudio. II. Universidade Federal do Amazonas, Programa de Pós-graduação em Informática. III. Um framework de apoio à instanciação de técnicas de seleção de tecnologias de software baseadas em estratégias de busca.

A Deus em primeiro lugar, a minha família pelo apoio em todos os momentos.

Agradecimentos

Em primeiro lugar a Deus, pois sem ele nada é possível.

Aos meus queridos e amados pais, Aurélio Neto e Delza Grande, que sempre acreditaram em mim, investiram em minha educação e hoje estão colhendo os frutos que foram plantados no passado.

Ao professor Arilo Cláudio pela excelente orientação e incentivo em todos os momentos, seja na orientação da pesquisa ou quando em momentos de dificuldade na qual era necessário uma palavra de um amigo.

À minha irmã Jácy Alice que sempre me apoiou e torceu por mim.

Ao meu amor, Ruth Lopes que sempre esteve presente e teve muita compreensão nos momentos mais difíceis.

Aos professores da UFAM, em especial a professora Rosiane de Freitas Rodrigues pela parceria nos artigos e também pelas palavras de incentivos e motivação.

Aos profissionais do Instituto Nokia de Tecnologia (INdT) por proporcionarem o crescimento profissional através da parceria com a UFAM.

À minha prima Sandra Aparecida Grandi que infelizmente nos deixou há pouco tempo e que era uma pessoa adorável de quem gostava bastante e sempre quis o meu melhor.

Aos meus colegas do grupo ExperTS, com os quais participamos nas atividades do laboratório, nas aulas e trabalhos das disciplinas.

Ao INCT-SEC pelo apoio financeiro nas participações em eventos científicos.

E a todos que me ajudaram direta ou indiretamente, todos vocês estão guardados com muito carinho.

Resumo da Dissertação apresentada à UFAM/AM como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática (M.Sc.)

UM FRAMEWORK DE APOIO À INSTANCIAÇÃO DE TÉCNICAS DE SELEÇÃO DE TECNOLOGIAS DE SOFTWARE *BASEADAS EM ESTRATÉGIAS DE BUSCA*

Aurélio da Silva Grande

Março / 2013

Orientador: Arilo Claudio Dias Neto

A qualidade de um projeto de software está diretamente relacionada com as decisões tomadas durante suas diversas fases, pois decisões equivocadas podem causar danos significativos no projeto. Entre as decisões de um engenheiro de software, pode ser citada a escolha de tecnologias a serem aplicadas em um projeto de software. Geralmente estas decisões são tomadas levando-se em conta a experiência dos profissionais envolvidos nas tarefas. Assim, deixa-se de explorar outras soluções mais adequadas para tal cenário, algo que uma abordagem científica de apoio a tal seleção poderia oferecer.

O trabalho apresenta um framework para instanciação de técnicas de seleção de tecnologias de software baseado em estratégias de busca. Para isso, o Problema de Seleção de Tecnologias de Software, do inglês *Software Technologies Selection Problem* (STSP) foi modelado como um problema de otimização combinatória (Conjunto Dominante Mínimo) com o objetivo de atender diferentes cenários reais de Engenharia de Software.

O framework proposto para STSP foi idealizado como um mecanismo de apoio aos engenheiros de software que possuiriam dificuldades em usar outros frameworks de otimização genéricos durante um projeto de software, devido a prazos curtos e recursos limitados. Tal framework foi desenvolvido para ser integrado com os principais frameworks de meta-heurística de otimização identificados na literatura técnica, como JMetal e Opt4J, que implementam um grande número de meta-heurísticas.

Para analisar a viabilidade da modelagem proposta para o STSP e do framework desenvolvido, foram realizados dois estudos de casos em problemas de otimização do mundo real: (i) seleção de técnicas de teste baseado em modelos; (ii) seleção de técnicas de elicitação de requisitos para sistemas embarcados. Os estudos foram realizados utilizando diferentes meta-heurísticas. Os resultados indicam sua viabilidade de apoio à seleção de tecnologias de software.

Palavra Chave: Seleção de Tecnologia, Meta-Heurísticas, SBSE, Framework.

Abstract of Thesis presented to UFAM/AM as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A FRAMEWORK TO SUPPORT INSTANTIATION OF SOFTWARE TECHNOLOGIES SELECTION TECHNIQUES USING SEARCH-BASED STRATEGIES

Aurélio da Silva Grande

March / 2013

Advisor: Arilo Claudio Dias Neto

The quality of software design is directly related to the decisions taken during its execution, because wrong decisions may cause significant damage in a software project. Among the decisions to be performed by a software engineer, an important one would be the selection of technologies to be applied to software projects. Usually, these decisions are made taking into account the experience of the professionals involved in the task. Thus, we limit the exploration of other possibilities that could be most appropriate for such scenario, what could be offered by a scientific approach to support this decision making.

This thesis presents a framework for instantiation of software technologies selection techniques by using search-based strategies. For this, the Software Technologies Selection Problem (STSP) was modeled as a combinatorial optimization problem (Minimum Dominanting Set) with the purpose of to attend different and real scenarios of Software Engineering.

The proposed framework for the STSP was idealized as a mechanism to support software engineers who are not able to use other generic optimization frameworks in a software project due to tight deadlines and limited resources. It was designed to be integrated with the main meta-heuristic optimization frameworks, such as JMetal and Opt4J, that implement a large number of meta-heuristics.

To analyze the feasibility of the proposed modeling and the developed framework, two case studies were conducted in complex and real optimization problems: (i) selection of model-based testing techniques; (ii) selection of requirements elicitation technique for critical embedded systems. The studies were performed using different meta-heuristics and their results indicate their feasibility to support the selection of software technologies.

Keywords: Technology Selection, Metaheuristics, SBSE, Framework.

ÍNDICE

LISTA DE ABREVIATURAS E SIGLAS

- *AADL – Architecture Analysis and Design Language*
- ACM – *Association for Computing Machinery*
- *ACME – An Architecture Description Interchange Language*
- ACO – *Ant Colony Optimization*
- *ACOL – A Model Annotation Language*
- *ASDL – Architecture Structure Description Language*
- AVM – *Alternating Variable Method*
- *CCM – Common Object Request Broker Architecture Component Model*
- cGA – *cellular Genetic Algorithm*
- *CREST – Centre for Research on Evolution, Search and Testing*
- DDest – Dimensão “Destino”
- DObj – Dimensão “Objetivo”
- *DRTSADL – Distributed/Embedded Real-Time System ADL*
- DSour – Dimensão “Fonte”
- *EAST-ADL – Electronic Architecture and* Software *Tools ADL*
- EC – *Evolutionary Computation*
- EX – Extensíveis
- *FSM – Finite State Machines*
- GAs – *Genetic Algorithms*
- GD – *Generational Distance*
- GQM – *Goal-Question-Metric*
- HV – *Hypervolume*
- IGD – *Inverse Generational Distance*
- INCT-SEC – Instituto Nacional de Ciência e Tecnologia – Sistemas Embarcados Críticos
- MA – *Memetic Algorithms*
- *MARTE – UML Profile for Modeling and Analysis of Real-time and Embedded Systems*
- MBTTSP – *Model-Based Testing Techniques Selection Problem*
- MCDM – *Multiple Criteria Decision Making*
- MD – Meta-heurísticas disponíveis
- *MetaH – Avionics Architecture Description Language*
- MIC – Mecanismo para instanciar cenários
- MOEAs – *Multiobjective Evolutionary Algorithms*

- *NCES – Net Condition/Event Systems*
- NRP – *Next Release Problem*
- NSGA-II – *Nondominated Sorting Genetic Algorithm* - II
- *PADL – Process Algebraic Architectural Description Language*
- PMBOK – *Project Management Body Of Knowledge*
- PSO – *Particle Swarm Optimizer*
- *REAL – Requirement Enforcement Analysis Language*
- RR – Representação dos resultados
- SA – *Simulated Annealing*
- *SAG* – Software *Architecture Graph*
- *SavModEffort* – Esforço de Modelagem Salvo
- SBIA – Simpósio Brasileiro De Inteligência Artificial
- SBSE – *Search-Based* Software *Engineering*
- SBX – *Simulated Binary Crossover*
- *SDL – Specification and Description Language*
- SSBSE – *International Conference on Search-Based Software Engineering*
- SPEA2 – *Strength Pareto Evolutionary Algorithm* 2
- STSP – Software *Technologies Selection Problem*
- *SWP_Cov* – Cobertura de Projeto de Software
- *SysML – Systems Modeling Language*
- UDRE – Uso de dados reais em experimentos
- *UML – Unified Modeling Language*
- *UML-RT – UML Profile for Real-time Systems*
- V&V – Verificação e Validação

LISTA DE FIGURA

LISTA DE TABELAS

CAPÍTULO 1: INTRODUÇÃO

Neste capítulo serão apresentados o contexto do trabalho, o que motivou esta pesquisa e a questão de investigação. Serão também apresentados os seus objetivos, a metodologia de pesquisa adotada, o histórico deste trabalho e a organização deste texto.

1.1. Contextualização e Motivação

A seleção de tecnologias de software aplicada em atividades específicas de desenvolvimento de software consiste em uma tarefa importante que pode afetar diretamente a qualidade do produto final. Esta tomada de decisão pode estar relacionada ao gerenciamento de projetos (BIRK, 1997; ARANDA et al., 2006), levantamento de requisitos (MAIDEN e RUGG, 1996), o projeto de arquitetura, garantia de qualidade (VEGAS e BASILI, 2005; WOJCICKI e STROOPER, 2007; DIAS-NETO e TRAVASSOS, 2009; VICTOR e UPADHYAY, 2011), dentre outras fases do desenvolvimento de um software. Em alguns casos, o número de tecnologias disponíveis para ser aplicadas em um projeto de software pode ser elevado, o que poderia dificultar a análise de todas as possíveis opções por um engenheiro de software, principalmente quando há outras restrições, tais como tempo, custo, recursos e qualidade em um projeto de software. Este cenário se caracteriza como um tema de pesquisa bastante atual a ser tratado na área de Engenharia de Software, pois envolve diretamente tomada de decisões importantes a ser realizada por engenheiros de software.

Em outra perspectiva, nos últimos dez anos está crescendo a adoção de técnicas baseadas em busca como mecanismo para apoiar a tomada de decisões em cenários de engenharia de software nos quais o número de opções disponíveis é muito grande e difícil de ser analisadas manualmente por engenheiros de software, considerando as restrições de projeto de software. Esta área foi intitulada como Engenharia de Software Baseada em Busca (em inglês *Search-Based* Software *Engineering* – SBSE) (HARMAN e JONES, 2001).

Integrando estas duas perspectivas, o problema da seleção de tecnologias para projetos de software atende aos dois requisitos apresentados por HARMAN e JONES (2001) para ser incluído no conjunto de problemas de SBSE:

(1) A escolha da representação do problema;
(2) A definição da função de adequação (*fitness*).

Em (DIAS-NETO et al., 2011), os autores apresentam uma modelagem para o problema de seleção de técnicas de testes baseada em modelos para projetos de software, um subproblema de seleção de tecnologias de software, como um problema de grafo conhecido como o problema de conjunto dominante mínimo (GAREY e JOHNSON, 1979; BONDY e MURTY, 2008). No entanto, observa-se que este modelo poderia ser aplicado para a seleção de tecnologias em diversos cenários de engenharia de software (por exemplo: técnicas de teste, os métodos de levantamento de requisitos, metodologias de desenvolvimento de software, entre outros) a partir da representação de características comuns nesses cenários e a especialização desta modelagem para representar as características específicas de cada cenário.

Com base neste contexto, este trabalho propõe a modelagem do problema de seleção de tecnologias de software (em inglês, Software *Technologies Selection Problem* – STSP), em geral, como um problema de otimização combinatória. A partir desta modelagem, foi desenvolvido um framework, uma infraestrutura computacional, para instanciar sistemas de recomendação que apoiam a seleção de tecnologias de software a partir da aplicação de técnicas SBSE. Os benefícios esperados com este framework são:

- Apoiar a tomada de decisão dos engenheiros de software a respeito da seleção de tecnologias de software para projetos de software reais;
- Apoiar a investigação na área SBSE por pesquisadores interessados em avaliar novas abordagens de apoio à seleção de tecnologias de software através da instanciação do framework proposto para algum cenário.

Assim, o framework pode contribuir para avaliar as estratégias propostas que implementam diferentes algoritmos baseados em busca (meta-heurísticas) usando instâncias reais obtidas da literatura técnica, o que reduziria as ameaças à validade destas avaliações, quando comparado com avaliações através de exemplos fictícios.

1.2. Descrição do Problema

A seleção das tecnologias (por exemplo: processos, técnicas, métodos, ferramentas) para projetos de software tem sido relatada desde 1991 (BASILI, 1991), e desde então uma série de abordagens foram desenvolvidas para diferentes atividades no processo de desenvolvimento de software. De acordo com VEGAS e BASILI (2005), a seleção de tecnologias de software representa uma tarefa complexa que pode influenciar diretamente sobre a eficácia do processo e a qualidade do produto final. O principal desafio desta tarefa é entender e decidir quais tecnologias de software são mais adequadas para uma atividade específica de desenvolvimento de software.

O STSP pode ser definido como a composição de dois aspectos:

(1) A completude do conjunto de tecnologias de software que irá ser utilizado na tarefa de seleção, e;
(2) A identificação apropriada de características que representam simultaneamente tecnologias de software e ambiente de projeto de software.

Vários aspectos que fazem a seleção de tecnologias de software trabalhosa para um projeto de software, em diferentes atividades de engenharia de software, podem ser observados, tais como: a informação disponível sobre as tecnologias distribuídas em diferentes fontes (por exemplo: livros, jornais, repositórios), a falta de diretrizes para apoiar a seleção, e pouco conhecimento científico (evidência) sobre as tecnologias e seu uso em projetos de software anteriores (VEGAS e BASILI, 2005). Além disso, cada cenário de engenharia de software pode apresentar características específicas para a tarefa de seleção de tecnologias. Por exemplo, para alguma tarefa do processo de desenvolvimento a escolha de uma tecnologia de software tem de ser única, isto é, somente uma tecnologia de software é suficiente, pois a combinação de tecnologias não iria introduzir melhores resultados para o processo de desenvolvimento de software (por exemplo, a seleção de ferramenta de modelagem). Esta situação pode tornar a seleção de tecnologias de software mais simples de ser conduzida e gerenciada. No entanto, para outras tarefas (por exemplo: teste de software) a combinação de duas ou mais tecnologias de software pode introduzir uma melhoria para eficácia do processo e, consequentemente, a qualidade do produto final, mas pode introduzir um esforço adicional. Portanto, uma análise de viabilidade durante o processo de seleção é necessária.

1.3. Questão de Pesquisa

As questões de pesquisa a serem exploradas neste trabalho são as seguintes:

- Com o uso de recursos de otimização combinatória, meta-heurísticas, para o processo de seleção de tecnologias de software, seria possível aumentar a qualidade dos resultados obtidos, melhorando soluções conhecidas e alcançando soluções até então não exploradas no espaço de soluções existentes, dentro de um tempo computacional aceitável?
- Tal solução apoiaria o engenheiro de software na tomada de decisão, proporcionando maior validade à análise da seleção das tecnologias sugeridas?

1.4. Objetivos

Nesta seção serão apresentados os objetivos desta pesquisa.

1.4.1. Objetivo Geral

Propor um framework para apoiar a tomada de decisão na seleção de tecnologias de software no qual seja possível instanciar infraestruturas computacionais em diversos cenários da Engenharia de Software utilizando meta-heurísticas baseadas em estratégias de busca.

1.4.2. Objetivos Específicos

- Fornecer um corpo de conhecimento sobre tecnologias de software para dois cenários de Engenharia de Software: técnicas de teste baseado em modelos e técnicas de modelagem arquitetural para sistemas embarcados;
- Prover um framework que possibilite a instanciação de ambientes de apoio à seleção de tecnologias de software usando meta-heurísticas
- Prover um framework que possibilite a sua integração com outros frameworks de apoio à otimização combinatória disponíveis na literatura técnica, permitindo o uso e adição de meta-heurísticas provenientes destes frameworks;
- Apoiar na avaliação experimental de abordagens de seleção de tecnologias de software propostas, a partir de sua configuração no framework proposto contendo técnicas reais obtidas da literatura técnica.

1.5. Justificativa

Várias abordagens têm sido propostas na literatura técnica para suportar a seleção de tecnologias de software. O apoio prestado por essas abordagens varia desde a caracterização genérica de tecnologias de software com vista à sua reutilização em projetos de software (BASILI, 1991; BIRK, 1997) até abordagens para apoiar a seleção de tecnologias de software para uma atividade específica, tais como levantamento de requisitos (MAIDEN e RUGG, 1996) e Teste de Software (VEGAS e BASILI, 2005; WOJCICKI e STROOPER, 2007; DIAS-NETO e TRAVASSOS, 2009; VICTOR e UPADHYAY, 2011).

Normalmente, essas abordagens de seleção usam como estratégia criar um catálogo de tecnologias de software que será apresentado a engenheiros de software para tomar decisões levando em consideração quais delas aplicar em projetos de software (BASILI, 1991; BIRK, 1997; MAIDEN e RUGG, 1996; VEGAS e BASILI, 2005; WOJCICKI e STROOPER, 2007). Nos últimos anos, algumas propostas de abordagens de seleção usaram algoritmos de otimização combinatória para apoiar o processo de seleção,

funcionando como um sistema de recomendação utilizando múltiplos critérios a tomada de decisão (DIAS-NETO et al., 2011; VICTOR e UPADHYAY, 2011).

No processo de caracterização de cenários de engenharia de software para seleção de tecnologias, alguns aspectos são comuns para todas as decisões independentemente de qual atividade de engenharia de software esta tomada de decisão é obrigatória ou específica para cada atividade. A Figura 1 descreve alguns destes aspectos divididos de acordo com a sua categoria.

Aspectos Comuns

- Comparação de características de tecnologias de software disponíveis em relação as cacterísticas dos projetos.
- Análise das habilidades necessárias para utilizar as tecnologias de software disponíveis e as competências obtidas pela equipe do projeto de software.
- Medir o esforço / custo / tempo para usar as tecnologias de software como instrumento para avaliar a sua adequação ao cronograma do projeto de software e orçamento.
- As tecnologias de software podem ser aplicadas em um projeto de software original ou em uma carteira de projetos de software.

Aspectos Específicos

- Cada atividade de engenharia de software lida com atributos específicos de caracterização. Por exemplo, "nível de testes necessários no projeto de software" é um atributo importante para a caracterização da seleção da técnica de teste, mas não é importante para selecionar técnicas de inspeção.
- Cada atividade de engenharia de software pode considerar uma medida diferente (representado em unidades diferentes) para o esforço e custo.
- A medida de eficiência pode ser específica para cada atividade. Por exemplo, uma técnica de teste é avaliada pelo número de falhas reveladas, enquanto em uma técnica de inspeção é avaliado o total de defeitos detectados por minuto.
- Algumas atividades permitem apenas a seleção individual de tecnologias de software, mas outras permitem a seleção combinada (mais de uma tecnologia utilizada em um projeto para apoiar a mesma atividade).

Figura 1. Aspectos comuns X Específicos de seleção de tecnologia de Software.

Assim, um desafio para um framework de apoio a STSP seria fornecer as características comuns de seleção necessárias em qualquer atividade de desenvolvimento de software e ainda viabilizar as especializações necessárias para cada atividade.

1.6. Metodologia de Pesquisa

Para o desenvolvimento deste trabalho de pesquisa adotou-se a seguinte metodologia de pesquisa (Figura 2):

1.6.1. Fase de Concepção

- **C1)** Revisão da literatura sobre meta-heurísticas aplicadas a problemas baseados em busca.
- **C2)** Estudo sobre a utilização de ferramentas que já implementam essas meta-heurísticas.
- **C3)** Implementação do Framework.
- **C4)** Integração do Framework desenvolvido a uma ferramenta que implementa meta-heurísticas.

1.6.2. Fase de Avaliação:

- **A1)** Criar um corpo de conhecimento para o cenário de Teste de Software Baseado em Modelos;
- **A2)** Realizar o estudo com os cenários definidos na fase anterior.
- **A3)** Analisar os resultados das diferentes meta-heurísticas levando em consideração critérios mono-objetivos e multiobjetivos.
- **A4)** Criar um corpo de conhecimento para o cenário de Modelagem Arquitetural para Sistemas Embarcados;
- **A5)** Realizar o estudo com o cenário definido na fase anterior.
- **A6)** Avaliar a viabilidade do framework em possibilitar a configuração da abordagem de seleção para o cenário descrito nos passos A4 e A5.
- **A7)** Otimizar o framework a partir dos resultados do estudo.

1.7. Organização do Trabalho

Este trabalho está organizado em cinco capítulos, incluído este primeiro capítulo de introdução, que apresentou a motivação, problema, objetivos e metodologia e o contexto da pesquisa. A organização do texto deste trabalho segue a estrutura abaixo:

- **Capítulo 2: "Fundamentação Teórica":** Descreve o problema de seleção de tecnologias situando-o na classificação realizada por HARMAN et al. (2009) como problemas de gestão, além de fornecer os principais conceitos relacionados à

pesquisa como meta-heurísticas, e, por fim, apresenta os trabalhos relacionados detalhando os frameworks de otimização que implementam meta-heurísticas.

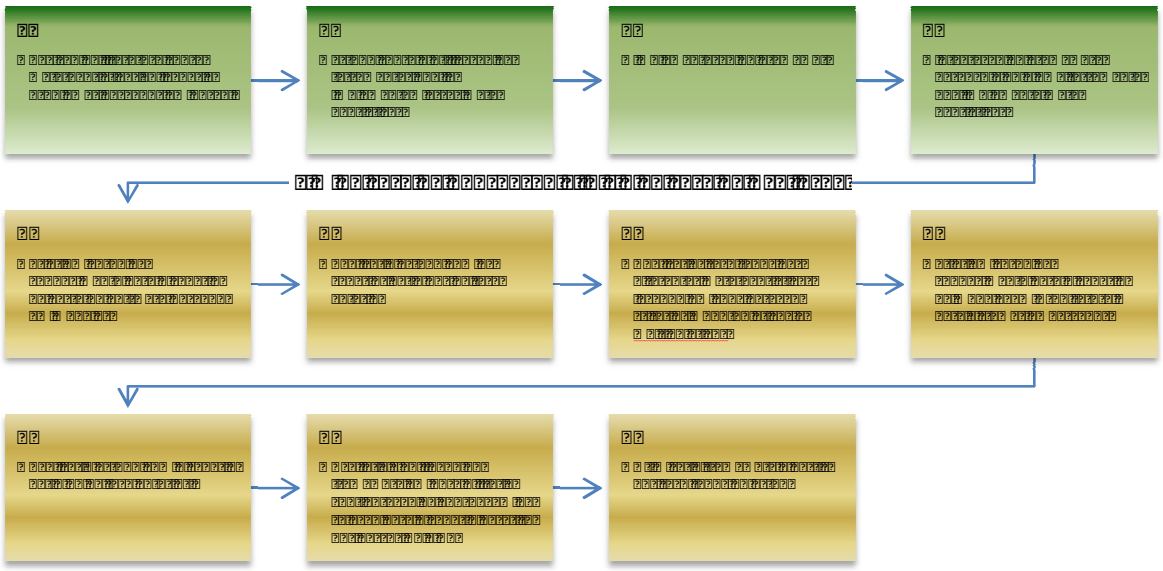


Figura 2. Atividades que compuseram a metodologia seguida neste trabalho.

- **Capítulo 3: "Framework de Apoio à Instanciação de Técnicas de Seleção de Tecnologias de Software":** Descreve o problema de seleção de tecnologias como problema de otimização combinatória bem como a modelagem dele como problema do conjunto dominante mínimo. Além disso, a estrutura do framework proposto é definida e os primeiros experimentos que serviram como prova de conceito são realizados. Por fim, a solução implementada é descrita realçando as principais etapas de utilização do framework.
- **Capítulo 4: "Estudos de Caso da Aplicação do Framework Desenvolvido":** Com o objetivo de validar o framework desenvolvido são realizados dois estudos de caso para dois cenários diferentes. Para cada um deles é formado um corpo de conhecimentos contendo técnicas encontradas na literatura técnica com a finalidade de executar experimentos utilizando algumas das principais meta-heurísticas que foram integradas ao framework.
- **Capítulo 5: Conclusões e Trabalhos Futuros:** Apresenta as conclusões desta dissertação, suas contribuições e trabalhos futuros que fornecem a direção para que seja dada continuidade a esta pesquisa.

CAPÍTULO 2: FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos relacionados ao tema abordado nesta pesquisa e os trabalhos relacionados a este que está sendo proposto.

2.1. Introdução

A Engenharia de Software é formada por diversas subáreas que possuem alguns problemas que podem ser modelados como problemas de otimização, como definição de técnicas de engenharia de requisitos, seleção de casos de testes, otimização de código, dentre outros (DE FREITAS et al., 2009). Esses problemas podem ser tratados de forma computacional com algoritmos de busca, especialmente meta-heurísticas. A junção dessas duas áreas deu início a uma nova área de pesquisa chamada Engenharia de Software Baseada em Busca (em inglês, *Search-Based* Software *Engineering* – SBSE) (HARMAN e JONES, 2001).

Embora o interesse em SBSE tenha crescido recentemente, como indicado na Figura 3, contendo trabalhos obtidos do repositório de publicações sobre SBSE[1], o uso de otimização em engenharia de software remonta aos anos 1970's (HARMAN et al., 2009). Uma das primeiras tentativas do uso de otimização no problema reportado por MILLER e SPOONER (1976) na área de testes de software. Este trabalho usou maximização numérica como técnica para geração de dados de testes em ponto flutuante para execuções de programas.

O termo SBSE foi usado pela primeira vez na literatura por HARMAN e JONES (2001) no qual foi sugerido que o termo "otimização baseada em busca" poderia ser usado em todo o espectro de atividades de Engenharia de Software. Esse trabalho foi publicado em forma de manifesto em favor da SBSE, pois também deve ser notado que anos atrás CHANG (1994) promoveu a utilização mais generalizada do termo Computação Evolutiva em Engenharia de Software.

A motivação para o surgimento dessa nova área foi devido a não ser possível utilizar normas e metodologias tradicionais de suporte ao desenvolvimento de software em determinados tipos de problemas que se caracterizam por serem intrinsecamente complexos e envolve a seleção de uma única solução em conjunto grande de possíveis

[1] http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html

soluções. Esses problemas têm como objetivo otimizar (maximizar ou minimizar) uma função ou um grupo de funções de satisfação que possuem variáveis sujeitas a restrições de acordo com o problema. O uso de algoritmos exatos, aqueles que determinam a melhor solução caso exista, pode ser necessário para instâncias pequenas de alguns problemas deste tipo. No entanto, problemas de otimização em engenharia de software geralmente não se enquadram nessas características, sendo o mais recomendado o uso de meta-heurísticas (GLOVER, 1986).

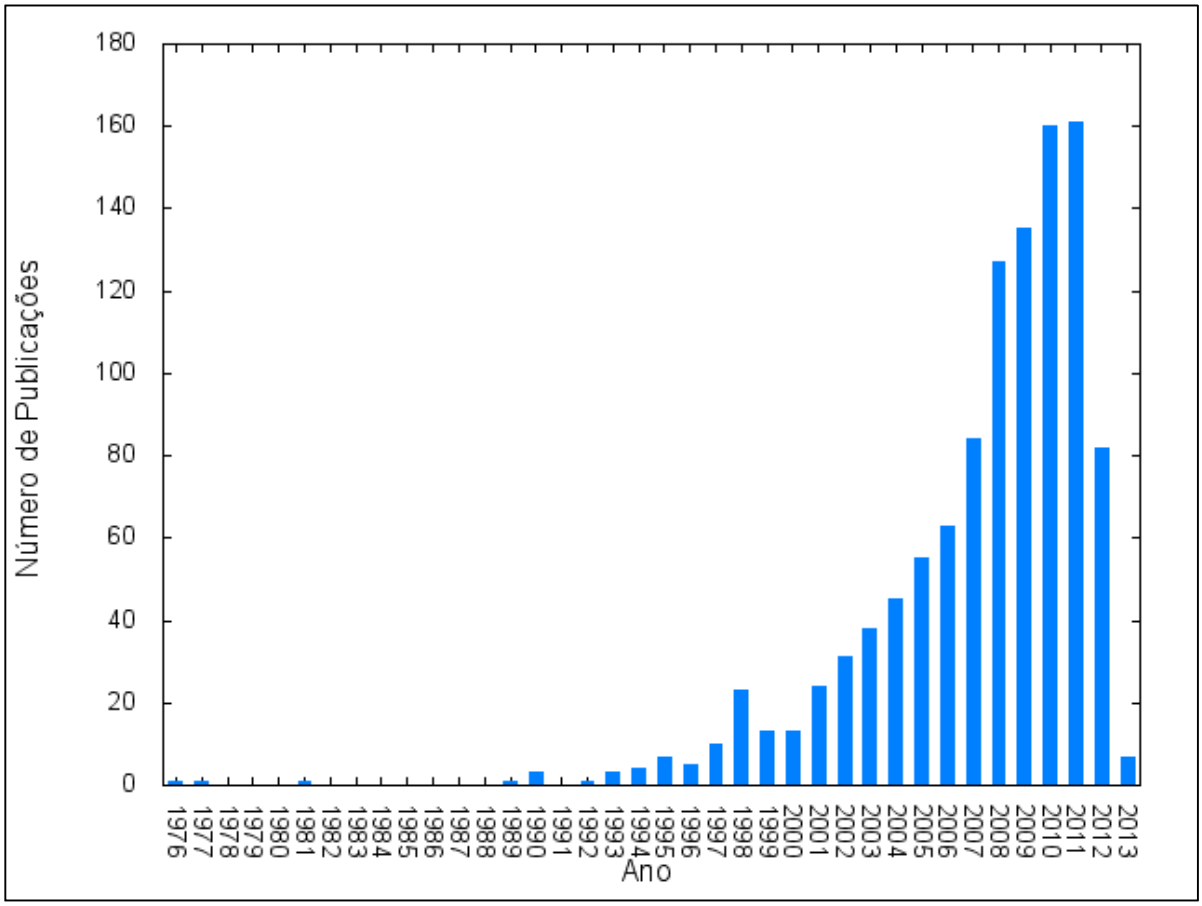


Figura 3. Número de publicações sobre SBSE (Fonte: CREST[2]).

Segundo pesquisa realizada por HARMAN et al. (2009), os algoritmos de otimização baseadas em busca mais adotados em SBSE são técnicas de busca globais, como algoritmos genéticos e arrefecimento simulado (em inglês, *simulated annealing),* e técnicas de busca local, como subida de encosta (em inglês, *hill climbing)*. Os autores também propuseram um esquema de classificação, baseada nas publicações sobre SBSE, das áreas de Engenharia de Software nas quais são usadas meta-heurísticas.

[2] http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html

Em HARMAN et al. (2009) é descrita uma análise de diversos trabalhos de SBSE e suas categorizações de acordo com um esquema de classificação das atividades de Engenharia de Software baseado no sistema de classificação da ACM (*Association for Computing Machinery*), apresentando sete categorias principais: Especificação de requisitos, Técnicas e ferramentas de Modelagem; Verificação de Programa/Software e Checagem de Modelos; Testes e Depuração; Manutenção, distribuição e aperfeiçoamento; Métricas e Gestão, além de outras com poucos trabalhos publicados. Dessas sete, a que responde pela maioria das publicações com 56% é categoria de Testes e Depuração, conforme a Figura 4 contendo trabalhos obtidos do repositório de publicações sobre SBSE.

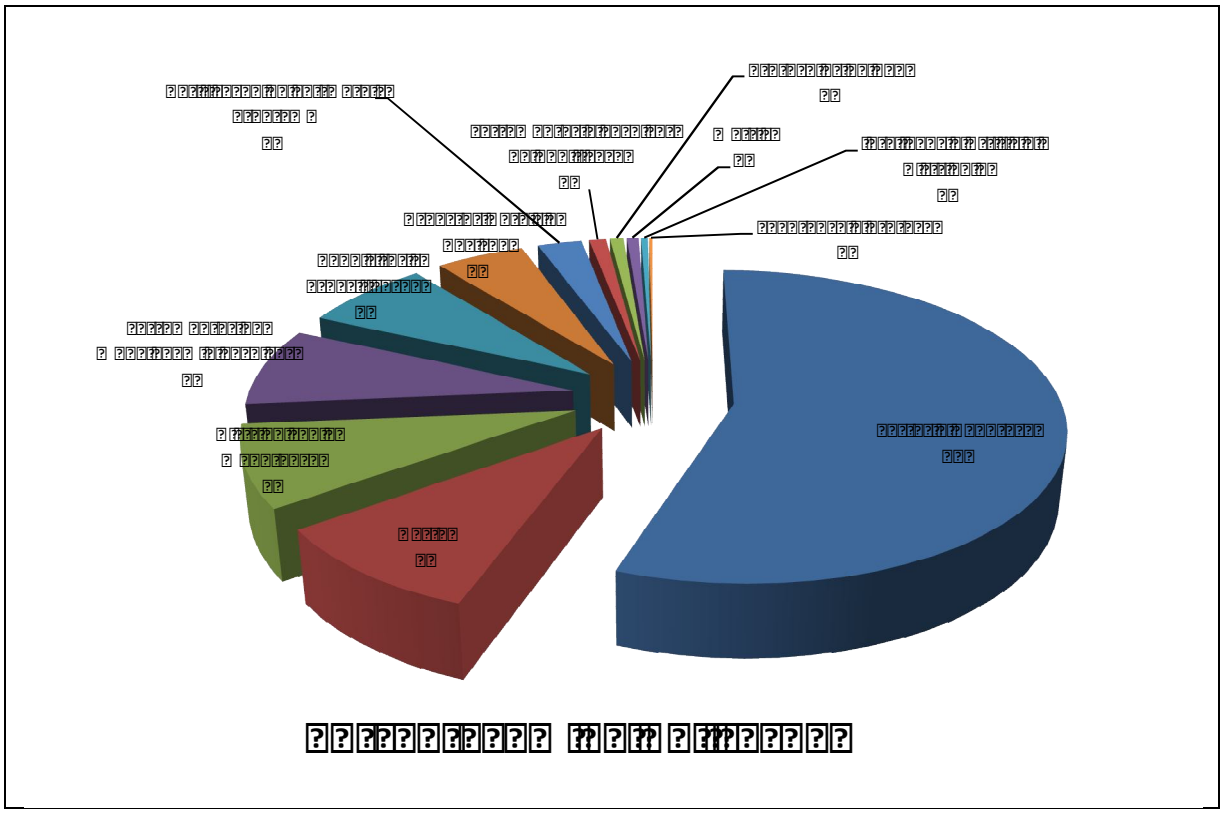


Figura 4. Espectro de publicações em SBSE e suas áreas de aplicação (Fonte: CREST).

- **SBSE em Especificação de Requisitos**

Engenharia de Requisitos é uma das mais importantes áreas no processo de Engenharia de Software (CHENG e ATLEE, 2007). Nesta área, SBSE é aplicada para otimizar a escolha de requisitos, a priorização de requisitos e o relacionamento entre requisitos e implementação. Um dos mais conhecidos problemas de busca foi formulado por BAGNALL et al. (2001) e chamado de "*Next Release Problem (NRP)*", no qual a ideia é encontrar o conjunto ideal de requisitos que balanceie pedidos de clientes, restrições de recursos e interdependência de requisitos.

Diversos autores aplicaram meta-heurísticas para esse problema ou subproblemas derivados desses, como FEATHER e MENZIES (2002) que aplicaram Arrefecimento Simulado, e GREER e RUHE (2004) que propuseram um algoritmo evolutivo. ZHANG et al. (2007), assim como SALIU e RUHE (2007), propuseram uma abordagem multiobjetivo. JIANG et al. (2010) utilizaram um algoritmo híbrido de otimização por colônia de formigas (ACO). FREITAS et al. (2011) utilizaram técnicas de otimização exata para solucionar o problema. XUAN et al. (2012) propuseram uma abordagem baseada em algoritmo multi-nível, em inglês, *Backbone based Multilevel Algorithm* (BMA) para endereçar o NRP aplicando reduções do problema e refinamentos multi-níveis para construir um melhor conjunto de soluções que maximiza o lucro dos clientes.

ZHANG et al. (2013) realizaram uma avaliação empírica de gerenciamento de Interação de requisitos através do uso do algoritmo evolutivo multiobjetivo NSGAII (DEB et al., 2002) que foi usado para produzir dados de referência para cada conjunto de dados, a fim de determinar quantas soluções na frente de Pareto não conseguem atender cinco diferentes restrições de interação de requisitos.

- **SBSE em Técnicas e Ferramentas de Modelagem**

HARMAN et al. (2009) sumarizam os trabalhos sobre Técnicas e Ferramentas de Modelagem ao problema de reconstruir limites de módulos depois da implementação utilizando como critério coesão e acoplamento. Em outro trabalho, SIMONS e PARMEE (2008) propuseram um algoritmo evolutivo multiobjetivo para modelagem de software orientado a objetos. O'KEEFFE e Ó'CINNÉIDE (2004) converteram o problema de modelagem de software orientado a objetos em um problema de otimização usando Arrefecimento Simulado.

Outra vertente de trabalhos em SBSE está relacionada ao uso de padrões de projetos em modelagens. Por exemplo, RÄIHÄ et al. (2008) propuseram uma abordagem baseada em algoritmo evolutivo para sintetizar automaticamente arquiteturas de software consistindo em diversos padrões de projeto. SIMONS e PARMEE (2009) realizaram uma investigação empírica sobre estratégias de busca para seleção de modelos conceituais de engenharia de software.

RÄIHÄ (2010) realizou uma revisão bibliográfica sobre estratégias de buscas aplicadas a problema de modelagem de software. BARROS (2011) examinou a eficácia do uso da qualidade de modularização (MQ) como uma função objetivo no contexto do problema de agrupamento de módulos de software. COLANZI e VERGILIO (2012) realizaram casos de estudo sobre arquiteturas de linha de produtos utilizando algoritmos evolutivos multiobjetivos.

Trabalhos sobre essa área têm crescido nos últimos anos com novos e interessantes problemas de modelagem em Engenharia de Software e esses são apenas alguns trabalhos encontrados na literatura técnica.

- **SBSE em Verificação de Programa / Software e Checagem de Modelos**

GODEFROID (1997) foi um dos primeiros autores a aplicar estratégias de busca para explorar o espaço de estados usados em checagem de modelos. Devido o espaço de estados ser muito grande para ser explorado, foi utilizada identificação isomórfica de subgrafos e buscar por exemplos contrários. ALBA e CHICANO (2007) e CHICANO e ALBA (2008) utilizaram a meta-heurística de Otimização por Colônia de Formigas para explorar esses espaços de estados em busca de contra exemplos.

Outros autores também exploraram a relação entre SBSE e checagem de modelos, como JOHNSON (2007) usou verificação de modelo para medir a aptidão na evolução de máquinas de estado finito, enquanto KATZ e PELED (2008) forneceram uma verificação de modelo baseada em uma abordagem de programação genética combinada com especificações assertivas *Hoare–logic–style* para a verificação e síntese a partir da especificação.

LEFTICARU et al. (2009) propuseram uma estratégia automatizada projeto do modelo, através da integração de algoritmos genéticos com a verificação do modelo. KATZ e PELED (2010) aplicaram a verificação de modelo deles baseada em uma abordagem de programação genética para viabilizar duas tarefas: encontrar uma instância de um protocolo que é suspeito de ter defeito, e corrigir o erro automaticamente.

- **SBSE em Testes e Depuração**

Teste de Software é usado para medir a qualidade do software desenvolvido com relação a tópicos de correção, completude e segurança, além de requisitos não funcionais como capacidade, confiabilidade, eficiência, portabilidade, manutenibilidade, compatibilidade e usabilidade. Por ser uma das primeiras áreas na qual foi utilizada SBSE, é a que possui mais publicações a respeito.

Muitos autores apontam que o trabalho publicado por MILLER e SPOONER (1976) foi um dos primeiros a combinar os resultados reais de execuções de programas com uma técnica de busca. Seu método foi originalmente concebido para a geração de dados de teste em ponto flutuante. No entanto, segundo MCMINN (2004), os princípios usados na ocasião possuem mais aplicações O testador seleciona um caminho através do programa e então produz uma versão em linha reta, contendo apenas aquele caminho.

Outros autores pioneiros na área são XANTHAKIS et al. (1992) que aplicaram algoritmos genéticos para geração de dados de testes para cobertura de estrutura, SCHOENAUER e XANTHAKIS (1993) que trabalharam no desenvolvimento de técnicas aperfeiçoadas para tratamento de restrições em algoritmo genético, usando geração de dados de testes baseados em busca como exemplo, seguido por DAVIES et al. (1994) para geração de casos de testes utilizando uma abordagem de algoritmo genético, bem como FERGUSON e KOREL (1996) utilizando uma abordagem de busca local.

HAAS et al. (2005) propuseram uma variação de busca local originalmente proposto por KOREL (1990) chamado "*Alternating Variable Method (AVM)*" sendo mais eficiente do que técnicas baseadas em população. Outros trabalhos são encontrados na literatura para testes estruturais, baseados em modelos, de mutação, de temporalidade, de exceção, de regressão, de configuração e integração, de estresse, de integração, dentre outras categorias de testes de software.

Recentemente, FARZAT e BARROS (2010) apresentaram uma abordagem heurística para selecionar casos de teste que poderiam apoiar defeitos críticos com o objetivo de maximizar a cobertura e diversidade da atividade de teste sob uma rigorosa restrição de tempo e dada à prioridade dos recursos que foram alterados; BAUERSFELD et al. (2011) apresenta uma abordagem para encontrar sequencias de entrada para interfaces de usuários gráficas usando otimização por colônia de formigas e relativamente nova métrica chamada pilhas de chamadas máximas para uso dentro da função de adequação;

YOO e HARMAN (2012) usaram meta-heurísticas para geração de novos dados de teste a partir de dados de teste existentes; VOS et al. (2013) estudaram o uso de testes funcionais evolutivos na indústria através de dois estudos de caso e os resultados indicam que o teste funcional evolutivo em um ambiente industrial é escalável e aplicável.

- **Manutenção, distribuição e aperfeiçoamento**

Manutenção de Software é o processo de melhoria e otimização do software implantado, bem como correção de eventuais defeitos identificados após a sua entrega. A maioria de trabalhos de SBSE nesta área é aplicada a duas vertentes: modularização e refatoração. Para o primeiro tópico, MANCORIDIS et al. (1998) foram os primeiros autores a aplicar SBSE sobre modularização de software, no qual seu trabalho inicial utilizou o algoritmo subida de encosta (*Hill Climbing*) para módulos de *clustering* para maximizar coesão e minimizar o acoplamento, que resultou na ferramenta *Bunch* (MANCORIDIS et al., 1999) que implementa *clustering* de módulos de software.

Outros autores forneceram abordagens para o problema sugerido por MANCORIDIS et al. (1998). HARMAN et al. (2002) estudaram o efeito de atribuir uma granularidade de modularização especial como parte da função de adequação (*fitness*), enquanto MAHDAVI et al. (2003) mostraram que a combinação dos resultados de múltiplas subidas de encosta pode melhorar os resultados para uma simples subida de encosta com algoritmos genéticos.

Com relação ao tópico refatoração, esta técnica consiste em mudar o programa alterando sua estrutura sem alterar a semântica. REFORMAT et al. (2007) exploraram as aplicações de clones de software e apresentaram um método para geração automática de clones utilizando programação genética. HOSTE e EECKHOUT (2008) utilizaram uma abordagem multiobjetivo para buscar o espaço de opções de compilações que controlam os níveis de otimização do compilador *GCC*.

Recentemente, JENSEN e CHENG (2010) apresentam uma abordagem que suporta a composição de mudanças no projeto e faz a introdução de padrões de projeto como mecanismo do processo de refatoração. LEE et al. (2011) aplicaram um algoritmo genético (GA) para gerar a melhor agenda de refatoração dentro de um prazo razoável para lidar com este problema. GHAITH e Ó'CINNÉIDE (2012) apresentaram uma abordagem para a melhoria automatizada de segurança de software utilizando refatoração baseada em busca.

- **SBSE em Métricas**

A ideia da aplicação de SBSE na área de métricas é saber se as métricas propostas atendem ou não à condição de representação, além de verificar se os valores atribuídos para a medição de software refletem o relacionamento pertencente ao mundo real. Geralmente as métricas são tratadas como funções de *fitness*, na qual é possível avaliar se a métrica realmente captura as propriedades de interesse produzindo sequências de soluções cada vez melhores. Caso contrário, teria sido encontrado um bom contraexemplo (HARMAN e CLARK, 2004).

VIVANCO e JIN (2008) descreve um trabalho que usa otimização para selecionar, a partir de um conjunto de métricas, o melhor conjunto que captura as propriedades de interesse de um modo previsível. LANGE e MANCORIDIS (2007) aborda como identificar o estilo de programação, aplicando algoritmos genéticos para identificar desenvolvedores de software baseados em métricas de estilo.

KPODJEDO et al. (2009) utilizaram técnicas baseadas em busca – Chidamber, Kemerer, novas métricas definidas para Rhino, Java ECMA – para definir a contabilidade de métricas de software para o papel que uma classe desempenha no diagrama de

classes e para a sua evolução ao longo do tempo e os resultados preliminares mostram que as novas métricas são favoravelmente comparadas com métricas orientadas a objetos tradicionais.

- **SBSE em Gestão de projetos**

Gerenciamento de Projetos em Engenharia de Software se preocupa com o gerenciamento de atividades complexas conduzidas em diferentes estágios do ciclo de vida do software, procurando otimizar tanto os processos quanto o produto construído. Os problemas mais estudados na área de SBSE são alocação de recursos e tarefas, escalonamento e estimativas de custo-esforço.

Para citar alguns trabalhos em planejamento de projetos, CHANG (1994) descreve o primeiro uso de SBSE em problemas de Gerenciamento de Software, no qual introduz a abordagem de Gerenciamento de Projetos de Software Net (SPMNet) para programação de projetos e alocação de recursos, avaliando SPMNet com dados do projeto simulados. ANTONIOL et al. (*2005*) aplicaram algoritmos genéticos, subida de encosta, arrefecimento simulado, para o problema de alocação de pessoal para pacotes de trabalho. Outro tópico com bastante publicação é o de estimativas de software, onde BURGESS e LEFLEY (2001) reportaram resultados da aplicação de programação genética na estimativa de custos.

FIGUEIREDO e BARROS (2011) utilizaram otimização combinatória para seleção e priorização de carteira balanceada de Projetos de Software. TOMASI et al. (2012) propuseram uma técnica de redução de processo com base na otimização multiobjetivo, para minimizar ao mesmo tempo a complexidade do processo, não-conformidades, e perda de conteúdo de negócios, que permite melhorar o modelo de processo de compreensibilidade, diminuindo sua complexidade estrutural. CHEN e ZHANG (2013) desenvolveu uma abordagem com um agendador baseado em eventos (EBS) e algoritmo de otimização por colônia de formigas (ACO) para desenvolver um modelo flexível e eficaz para o planejamento de projetos de software.

O problema de seleção de tecnologias de software, tema deste trabalho, se encaixa nesse tópico e na próxima seção serão mostrados trabalhos relacionados que utilizaram ou não conceitos de SBSE para este problema.

2.2. Seleção de Tecnologias de Software

Seleção de tecnologias de software é um assunto que remonta a 1991, quando BASILI e ROMBACH (1991) publicaram uma proposta de abordagem de apoio à seleção e reuso de diferentes tipos de tecnologias de software. Por se tratar de uma tarefa complexa, esta tarefa pode influenciar na efetividade do processo e na qualidade do produto final

(VEGAS e BASILI, 2005). Segundo BERTOLINO (2004), a escolha das técnicas mais adequadas é tema de muitas pesquisas, e o fator que pode facilitar o processo de seleção é a quantidade de informações que se possui para apoiar a tomada de decisão.

Segundo DIAS-NETO e TRAVASSOS (2009), uma boa tomada de decisões está relacionada às especificidades de cada tarefa de um projeto de software, bem como vários aspectos como: conhecimento técnico e habilidade da equipe do projeto a respeito dessas tecnologias e sobre o domínio do problema, cronograma, esforço, custos, aspectos políticos da organização, dentre outros.

No contexto de software, o fato de existir uma gama de tecnologias disponíveis para apoiar o desenvolvimento de um projeto dificulta a tomada de decisão. Muitas vezes, a solução mais adequada não é escolhida devido a fatores como: informações sobre as tecnologias estarem distribuídas em diversas fontes; carência de diretrizes para apoiar a seleção de tecnologia em um determinado projeto; e pouco conhecimento sobre o uso de tecnologias em projetos de software passados (VEGAS e BASILI,2005).

Além disso, o problema de seleção de tecnologias pode ser visto como dois problemas: completude do conjunto de tecnologias na qual a seleção será baseada e identificação apropriada das características que representam tanto as tecnologias quanto o ambiente do projeto de software.

O primeiro se refere à composição inicial das tecnologias quanto à adequação a um dado aspecto do projeto a ser desenvolvido. O segundo trata da escolha de uma composição que seja mais adequada às características do projeto. Essa escolha, muitas vezes, reflete a experiência do responsável e o perfil da equipe, pode variar desde a escolha de uma única tecnologia, mas em alguns cenários a combinação de tecnologias pode ser a melhor opção, no entanto, isso aumenta a dificuldade e complexidade da escolha.

A seguir serão apresentadas algumas abordagens que apoiam a seleção de tecnologias obtidas em (DIAS-NETO e TRAVASSOS, 2009) e outras mais recentes. Essas abordagens são: para seleção de tecnologias em geral, técnicas de apoio à identificação de requisitos e técnicas de testes:

- **Seleção de Tecnologias de Software em Geral:**
  - (BASILI e ROMBACH, 1991): abordagem que utiliza esquemas de caracterização de tecnologias de software baseados em modelos para registrar o uso de tais tecnologias em projetos, viabilizando seu reuso em futuros projetos a partir da consulta desses esquemas previamente adotados;

- (BIRK, 1997): caracterização genérica de tecnologias de software baseadas na modelagem de domínio de uma aplicação utilizando o paradigma GQM – *Goal-Question-Metric* (BASILI *et al.,* 1994). Essa abordagem se torna difícil de aplicar devido a não apresentar atributos específicos de caracterização.

- **Seleção de Técnicas de Identificação de Requisitos:**
  - (MAIDEN e RUGG, 1996): abordagem para seleção de técnicas de identificação de requisitos baseada em questões sobre características sobre pontos fortes e fracos de cada técnica agrupados em conjuntos;
  - (ARANDA *at al.,* 2006): abordagem contextualizada para o cenário de desenvolvimento globalizado de software aplicado em atividades baseadas na comunicação utilizando estratégias oriundas da psicologia cognitiva em um processo distribuído entre grupo de usuários e desenvolvedores.

- **Seleção de Técnicas de Testes:**
  - (VEGAS e BASILI, 2005): abordagem para apoio a seleção de técnicas de testes chamada Esquema de Caracterização, no qual os atributos das técnicas são armazenados em um repositório e para cada projeto é gerado um catálogo de técnicas de testes para apoiar a tomada de decisão. Foi aplicado em testes de software apesar da generalidade dos atributos da caracterização. No entanto, não possibilita a combinação de técnicas;
  - (WOJCICKI e STROOPER, 2007): abordagem de seleção de técnicas de Verificação e Validação (V&V) que avalia a combinação das técnicas com o objetivo de maximizar a completude das técnicas através dos defeitos/falhas revelados por cada técnica e minimizar o esforço. Possui três atividades principais: Pré-Seleção, Maximizar Completude e Minimizar Esforço. É utilizada uma matriz para relacionar os defeitos/falhas revelados por cada técnica e o esforço;
  - (DIAS-NETO e TRAVASSOS, 2009): seleção de técnicas baseadas em modelos, chamada *Porantim*, desenvolvida utilizando a infraestrutura para apoiar o planejamento e controle de testes de software *Maraká (*DIAS-NETO e TRAVASSOS, 2010*)* que consiste em três etapas: caracterização do projeto de software e seus atributos, aplicação do algoritmo de recomendação *Porantim* para a combinação de técnicas de teste baseado em modelos, e por último a escolha de uma solução na listagem de técnicas que mais se adequam ao projeto. Sobre essa mesma infraestrutura foi implementado outro algoritmo de recomendação chamado *Porantim-Opt* (DIAS-NETO et al*.,* 2011).

- (VICTOR e APADHUAY, 2011): seleção da melhor técnica de teste em qualquer fase do ciclo de vida de desenvolvimento do software como problema Tomada de Decisão por Múltiplos Critérios (em inglês, *Multiple Criteria Decision Making* – MCDM) e usa a técnica de preferência de ordem por Similaridade com a Solução Ideal (TOPSIS), juntamente com o Processo de Análise Hierárquica (AHP) para resolvê-lo. AHP é utilizado para determinar os pesos dos critérios. Os pesos calculados usando AHP são usados em TOPSIS, uma técnica para resolver problemas de decisão e é utilizada para calcular as vantagens relativas das alternativas. O método baseia-se no conceito de TOPSIS no qual a melhor alternativa é escolhida se está a uma distância mais curta a partir da solução ideal positiva e é o mais distante da solução ideal negativa.

Dentre essas abordagens apresentadas, apenas as duas últimas (*Porantim-Opt* e MCDM) utilizam conceitos de otimização aplicados em problemas de Engenharia de Software, mas especificamente em cenários de Testes de Software.

2.3. Meta-Heurísticas

Algoritmos de aproximação foram formalmente introduzidos na década de 1960 para gerar soluções próximas do que seria considerada a solução ótima em problemas de otimização que não poderiam ser resolvidos eficientemente pelas técnicas computacionais disponíveis naquela época (GRAHAM, 1966). Com o advento da teoria de NP-Completude no início dos anos 1970, a área se tornou mais proeminente. Da mesma forma, a necessidade de gerar soluções perto do ótimo para problemas de otimização NP-Difícil se tornou a mais importante forma de lidar com a intratabilidade computacional (GAREY e JOHNSON, 1979), pois para alguns problemas as soluções perto do ótimo podem ser geradas rapidamente, porém para outros problemas isso é considerado mais difícil.

Em face disso, nos anos 1980 outras abordagens baseadas em análises probabilísticas e algoritmos randômicos tornaram-se populares. Novas técnicas para resolver problemas de programação linear surgiram e tiveram um grande crescimento nos anos 1990. Essas metodologias têm sido referenciadas como meta-heurísticas e incluem têmpera simulada ou arrefecimento simulado *(Simulated Annealing* – SA), Otimização por Colônia de Formigas (*Ant Colony Optimization – ACO),* Computação Evolucionária *(Evolutionary Computation – EC)* e Algoritmos Meméticos (*Memetic Algorithms – MA)*, dentre outras (GONZALEZ, 2007).

Meta-heurísticas representam uma família de técnicas de otimização aproximadas que fornecem soluções aceitáveis em um tempo razoável para a solução de problemas

difíceis e complexos da ciência e da engenharia (TALBI, 2009). São abordagens utilizadas quando não se conhece muito bem o que seria a solução ótima, devido ao espaço de soluções disponíveis ser muito abrangente (LUKE, 2011). Além disso, elas são muito utilizadas em problemas de otimização multiobjetivos, nos quais geralmente não existe uma única solução melhor que as outras, mas sim um conjunto de soluções não-dominadas, chamadas de soluções de compromisso (*tradeoff*).

2.3.1. Algoritmos Evolutivos para Problemas Multiobjetivos

Algoritmos baseados em população geralmente possuem abordagens genéticas, pois várias soluções podem ser mantidas simultaneamente durante todo o processo (RIFKI e ONO, 2012). Consequentemente, algoritmos genéticos (*genetic algorithms* – GAs) são capazes de lidar com problemas de otimização multiobjetivos, onde as soluções diferentes podem ser encontradas com base no conceito de não-dominância ou conceito de otimalidade, introduzida por EDGEWORTH (1881) e depois generalizada por PARETO (1896), chamada de Edgeworth-Pareto ótimo ou, simplesmente, Pareto-ótimo.

A solução é chamada não-dominada, se não for dominada por qualquer outra solução com ilustrado na Figura 5, na qual o ponto B, assim como, qualquer ponto na região hachurada correspondendo a uma solução factível domina a solução representada pelo ponto A para um problema de minimização. Neste contexto, GAs são chamados de algoritmos evolutivos multiobjetivos (*Multiobjective Evolutionary Algorithms* – MOEAs). A principal vantagem de MOEAs é a capacidade de construir, em apenas uma única execução, um conjunto de soluções de Pareto, que é o conjunto de todas as possíveis soluções não-dominadas.

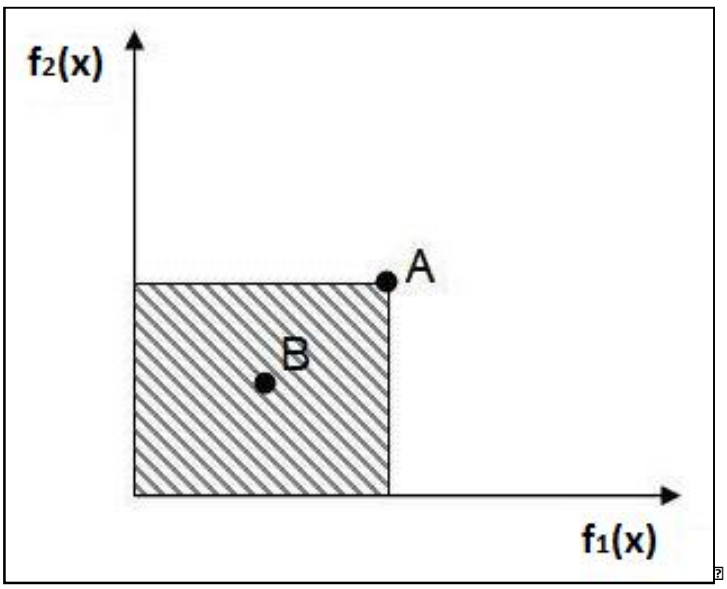


Figura 5. Representação de Dominância (AZUMA, 2011).

Formalmente temos a seguinte definição de dominância (Figura 6):

Dado um problema do tipo *[mini/max$_x$ $f_1(x)$, ..., $f_n(x)$]*, tal que *N* é o número de objetivos e *F* é o conjunto de soluções viáveis. Sem perder a generalidade, assumimos que o problema é um problema de minimização f_k e N. Nós dizemos que $x \in F$ domina $y \neq x \in F$ $\prec$ $f_1(x) \leq f_1(y) \wedge f_2(x) \leq f_2(y) \vee f_1(x) \leq f_1(y) \wedge f_2(x) \leq f_2(y)$

Figura 6. Definição Formal de Dominância.

A Figura 7 mostra graficamente o conceito da Fronteira de Pareto. Neste exemplo, trata-se de uma otimização por minimização para um problema hipotético. Por isso, as soluções mais próximas da origem dos eixos XY representam as soluções não-dominadas, pois nenhuma outra solução possui valores menores para os dois objetivos simultaneamente. As soluções S1, S2 e S3 representam Fronteira de Pareto enquanto S4 e S5 são soluções dominadas.

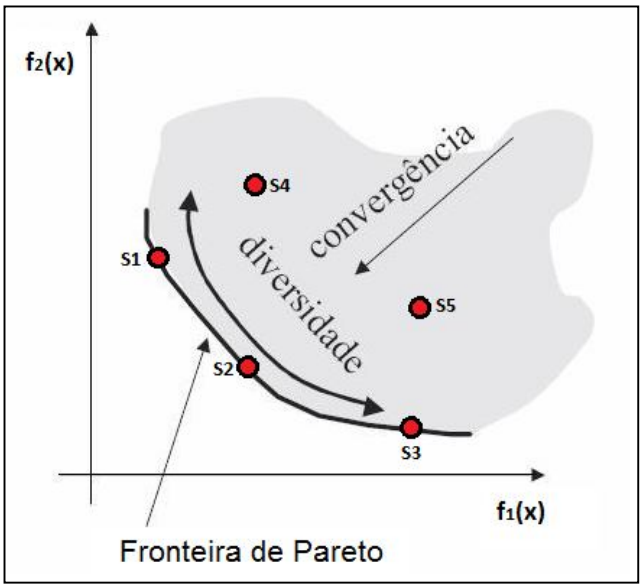


Figura 7. Representação da Fronteira de Pareto.

A seguir serão detalhados três algoritmos para problemas de otimização multiobjetivos utilizados nesse trabalho e que possuem implementações nos frameworks de meta-heurísticas pesquisados. Em geral, essas abordagens são algoritmos genéticos que combinam mecanismos de busca local em suas estratégias de geração de soluções.

2.3.1.1. NSGA-II

NSGA-II (*Nondominated Sorting Genetic Algorithm - II*) é um dos algoritmos mais eficientes proposto por Deb et al. (2002), pois possui complexidade $O(MN^2)$, onde M é o

número de objetivos e N é o tamanho da população. Este foi uma evolução do algoritmo NSGA proposto por SRINIVAS e DEB (1995) que, assim como outros algoritmos genéticos de ordenação de soluções não-dominadas como SPEA (KNOWLES e CORNE, 1999) e PAES (ZITZLER, 1999), possuem complexidade de $O(MN^3)$.

A primeira versão do algoritmo utiliza a função de compartilhamento responsável por manter a diversidade sustentável da população através da apropriada associação do parâmetro de compartilhamento σ_{share}. Esse parâmetro denota o maior valor da métrica de distância entre duas soluções que compartilham mutuamente valores de aptidão. Essa abordagem possui dois problemas: o desempenho do algoritmo está associado ao valor do parâmetro, e cada solução precisa ser comparada às outras soluções da população, o que acrescenta uma complexidade de $O(N^2)$ à essa abordagem.

Para contornar isso, na nova abordagem, a função de compartilhamento foi substituída pela métrica de estimação de densidade (uma média entre dois pontos ao longo de cada objetivo) e o operador de comparação de população $\prec_n$ (guia o processo de seleção em várias etapas em busca da Fronteira de Pareto), calculado para o conjunto de todas as soluções não-dominadas $\mathcal{X}$. Esse operador possui dois atributos: classificação de não-dominância (*nondomination rank*) i_{rank} e distância de aglomeração (*crowding distance*) $i_{distance}$.

Inicialmente, a população pai P_0 é criada randomicamente. A população é ordenada baseada na não-dominância. Cada solução é associada a um *fitness* ou *rank* equivalente ao nível de não-dominância dela (1 para o melhor nível, 2 para o próximo melhor nível, e assim por diante). Assim, a minimização de *fitness* é assumida. Primeiramente, os operadores de seleção *binary tournament*, recombinação e mutação são usados para criar a população *offspring* Q_0 de tamanho N.

O algoritmo executa o pseudocódigo definido na Figura 8 por *t* gerações. Primeiramente, a população combinada $R_t = P_t \cup Q_t$ é formada. A população R_t tem tamanho $2N$. Então a população R_t é ordenada de acordo com a não-dominância. Como todos os membros das populações anteriores e atual estão incluídos em R_t, o elitismo é garantido. Agora, soluções pertencentes ao melhor conjunto de não-dominadas $\mathcal{F}_1$ são as melhores soluções na população combinada e precisam ser enfatizadas mais do que qualquer outra solução da população combinada. Se o tamanho de $\mathcal{F}_1$ é menor que N, definitivamente são escolhidos todos os membros de $\mathcal{F}_1$ para a nova população P_{t+1}. Os membros remanescentes de P_{t+1} são escolhidos a partir das subsequentes frentes não-dominadas de acordo com seus *rankings*. Assim, soluções do conjunto $\mathcal{F}_2$ são escolhidas

na sequência, seguidas pelas soluções de $\mathcal{F}_3$, e assim por diante. Este procedimento é repetido até que nenhum conjunto possa mais ser acomodado.

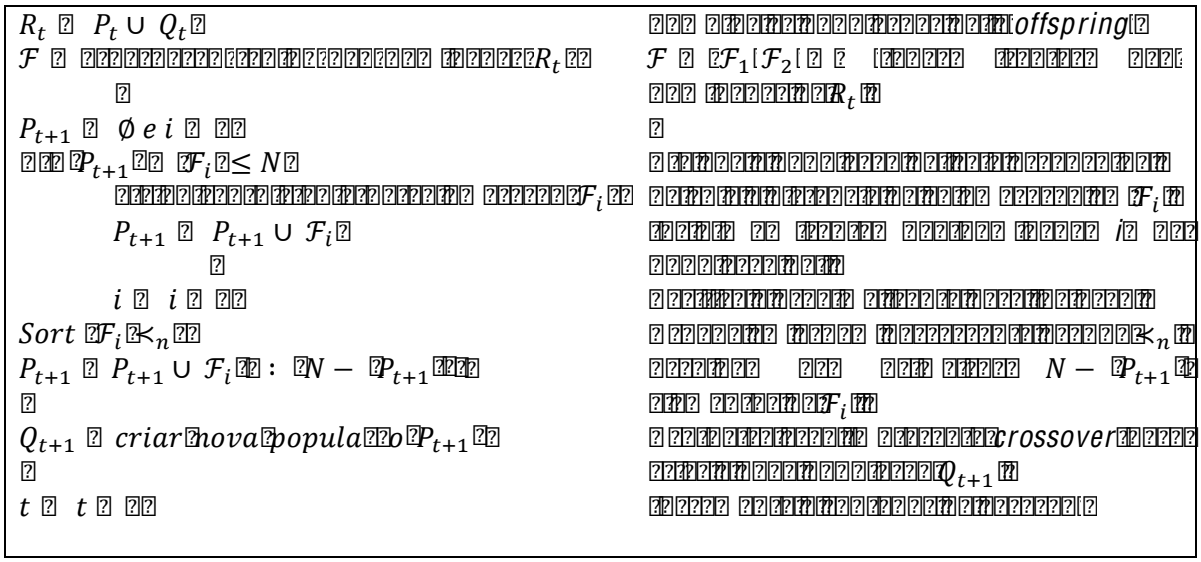


Figura 8. Pseudocódigo do algoritmo NSGA-II (DEB et al., 2002).

Sendo $\mathcal{F}_l$ o último conjunto não-dominado a ser acomodado, a quantidade de soluções de $\mathcal{F}_1$ a $\mathcal{F}_l$ geralmente é maior que o tamanho da população. Para escolher exatamente os N membros da população, todas as soluções da última frente são ordenadas usando o operador de comparação de aglomeração $\prec_n$ em ordem descendente e as melhores soluções necessárias para preencher todos os *slots* são escolhidas.

2.3.1.2. SPEA2

SPEA2 (*Strength Pareto Evolutionary Algorithm 2*) é outro algoritmo conhecido para problemas multiobjetivos. Ele foi proposto por Zitzler et al. (2002) como evolução do SPEA (ZITZLER e THIELE, 1999) através de *(i)* incorporação de uma melhor estratégia de atribuição de *fitness* que, para cada indivíduo da população, calcula quantos indivíduos ele domina e quantos é dominado; *(ii)* técnica de atribuição de densidade; e *(iii)* método de arquivamento truncado melhorado que garante a preservação das soluções limites.

A Figura 9 mostra o pseudocódigo do algoritmo SPEA2. Além das diferenças já mencionadas com relação ao SPEA, SPEA2 possui um tamanho de arquivo fixo, ou seja, sempre que o número de indivíduos não-dominados é menor que o tamanho de arquivo definido, o arquivo é preenchido pelos indivíduos dominados. Além disso, a técnica de agrupamento, que é invocada quando as frentes não-dominadas excedem o limite do arquivo, foi substituída pelo método de truncamento que possui características similares, mas não perde os pontos limítrofes.

Figura 9. Pseudocódigo do algoritmo SPEA2 (ZITZLER et al., 2002).

- **Associação de *Fitness***

No Passo 2 do pseudocódigo (Figura 9), cada indivíduo i no arquivo $\bar{P}_t$ e a população P_t é associado a um valor *strength* $S(i)$, representando o número de soluções que ele domina $S(i) = |\{j \mid j \in P_t + \bar{P}_t \wedge i \succ j\}|$ onde $|\cdot|$ denota a cardinalidade do conjunto, $+$ significa a união de multi-conjuntos e o símbolo $\succ$ corresponde a relação de dominação de Pareto. Com base nos valores de S, as aptidões $R(i)$ de um indivíduo i é calculada $R(i) = \sum_{j \in P_t + \bar{P}_t, j \succ i} S(j)$. Isto é, as aptidões são determinadas pelas frentes (*strengths*) de seus dominadores e ambos, arquivo e população, são considerados no contexto.

Além disso, a informação de densidade é incorporada para diferenciar indivíduos que possuem valores de linhas de *fitness* idênticos. A técnica de estimação de densidade é uma adaptação do método de *k*-vizinho mais próximo (SILVERMAN, 1986), na qual a densidade de um ponto é a função (decrescente) da distância para o *k*-ponto mais próximo. Aqui simplesmente é aplicado o inverso da distância do *k*-vizinho mais próximo como estimativa de densidade.

- **Seleção Ambiental**

A operação de atualização de arquivo, realizada no Passo 3 (Figura 9) do pseudocódigo, funciona da seguinte forma: primeiramente todos os indivíduos não-dominados, aqueles que possuem *fitness* menor que um, são copiados para o arquivo da próxima geração $\bar{P}_{t+1} = \{i \mid i \in P_t \cup \bar{P}_t \wedge F(i) < 1\}$. Se a frente de não-dominados couber exatamente dentro do arquivo $|\bar{P}_{t+1}| = \bar{N}$, o passo de seleção ambiental está completo. Caso contrário têm-se duas situações: ou o arquivo é menor $|\bar{P}_{t+1}| < \bar{N}$ ou maior $|\bar{P}_{t+1}| > \bar{N}$.

No primeiro caso, os melhores indivíduos dominados $\bar{N} - |\bar{P}_{t+1}|$ nos arquivos e população anteriores são copiados para o novo arquivo. Isto pode ser implementado ordenando-se os multi-conjuntos $P_t \cup \bar{P}_t$ de acordo os valores de *fitness* e copiando os primeiros $\bar{N} - |\bar{P}_{t+1}|$ indivíduos i com $F(i) \geq 1$ da lista ordenada resultante para $\bar{P}_{t+1}$. No segundo caso, quando o tamanho do corrente (multi)conjunto não-dominado excede $\bar{N}$, o procedimento de truncamento de população é invocado com iterativas remoções de $\bar{P}_{t+1}$ até $|\bar{P}_{t+1}| = \bar{N}$. O procedimento de truncamento escolhe o indivíduo com o menor valor de distância para outro indivíduo através no k-vizinho mais próximo de $\bar{P}_{t+1}$.

2.3.1.3. MoCell

MoCell (*A cellular genetic algorithm for multiobjective optimization*) é um algoritmo evolucionário proposto por NEBRO et al. (2006) que possui um desempenho tão bom quanto outros algoritmos considerados estado-de-arte, como NSGA-II (DEB et al., 2002) e SPEA2 (ZITZLER et al., 2002). Um diferencial dessa abordagem é que o espaço de soluções onde são aplicados os operadores não é tratado em uma única população (*panmixia*) e sim de forma estruturada usando o modelo celular (CANTÚ-PAZ, 2000), na qual apresenta uma melhora de comportamento em relação à abordagem anterior. Além disso, possui como características a utilização de um arquivo externo de soluções não-dominadas que podem voltar a população corrente através de um mecanismo de *feedback*.

O algoritmo utiliza o conceito de algoritmo genético celular, cGA (*cellular Genetic Algorithm*), no qual um indivíduo interage com os vizinhos mais próximos (MANDERICK e SPIESSENS, 1989), mas se diferencia pela existência de um arquivo externo que representa as soluções da frente de Pareto. Além disso, para gerenciar as inserções de soluções na frente de Pareto e obter diversidade, o estimador de densidade baseado na distância de aglomeração do algoritmo NSGAII é utilizado. O pseudocódigo é mostrado na Figura 10.

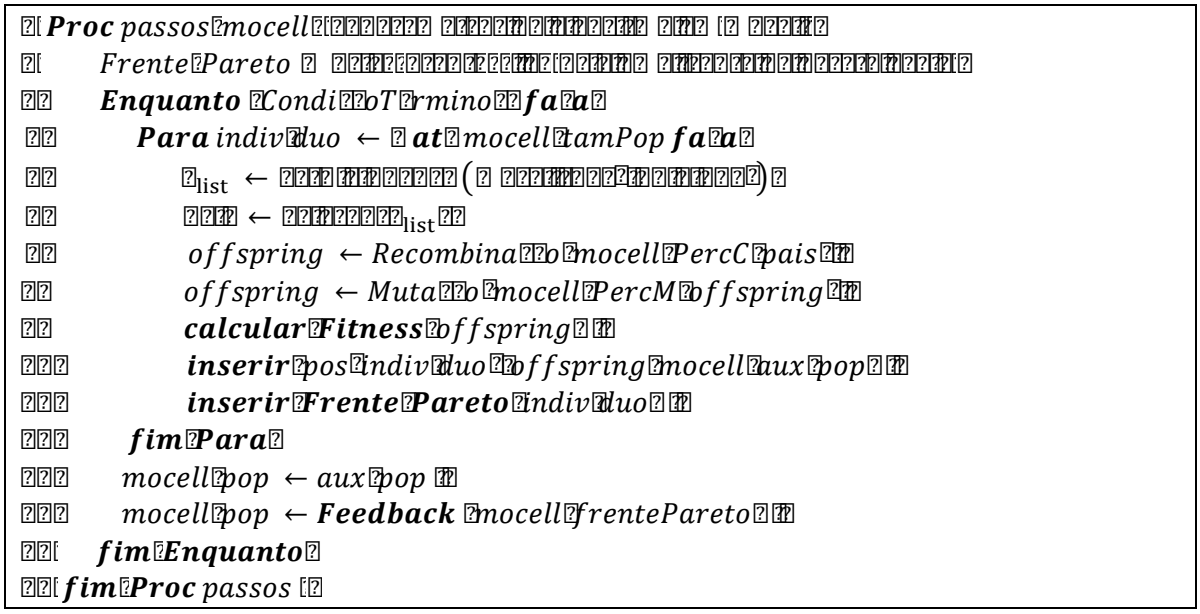


Figura 10. Pseudocódigo do algoritmo MoCell (NEBRO et al., 2006).

O algoritmo começa criando um frente de Pareto vazia (linha 2). Indivíduos são organizados em uma estrutura bidimensional e os operadores genéticos são sucessivamente aplicados a eles (linhas 7 e 8) até a condição de parada ser alcançada (linha 3). O algoritmo consiste em, para cada indivíduo, selecionar dois pais da vizinhança, recombiná-los para obter um *offspring*, realizar mutação deles, calcular o resultado individual e inseri-lo tanto na população (se ele não for dominado pela população corrente) e na frente de Pareto. Finalmente, depois de cada geração, a população antiga é substituída por uma auxiliar e o procedimento de *feedback* é invocado para substituir aleatoriamente um número fixo de soluções por elementos do arquivo.

2.4. Frameworks de Otimização

Existem vários frameworks de otimização disponíveis na literatura técnica, como JMetal (DURILLO e NEBRO, 2011), Opt4J (LUKASIEWYCZ *et al.,* 2011), EvA2 (KRONFELD *et al.,* 2010), ECJ (WILSON, 2004), *Watchmaker* (LEMNARU et al., 2012), JCLEC (VENTURA *et al.*, 2008), JGAP (CHEN *et al.*, 2001) e JCell (SPIETH *et al.*, 2006). A seguir vamos detalhar a arquitetura de quatro desses frameworks com o intuito de realizar uma comparação qualitativa entre eles com a finalidade de conhecer os recursos existentes que possam ser utilizados neste trabalho.

2.4.1. JMetal

JMetal é um framework de apoio ao desenvolvimento e experiência com meta-heurísticas para resolver problemas de otimização multiobjetivo. Tal framework tem uma

arquitetura extensível, repositório contendo problemas de otimização conhecidos, diversos algoritmos, bem como os operadores utilizados em algoritmos genéticos.

Tal framework também fornece suporte para implementação de meta-heurísticas, definição de problemas, utilização de indicadores de testes de desempenho e suporte de extensão para estudos experimentais. Possui também um pacote que permite a representação gráfica dos resultados com base no método pontos de referência interativos. Neste pacote você pode selecionar algoritmos, problemas e indicadores de qualidade.

2.4.1.1. Arquitetura

JMetal possui uma arquitetura orientada a objetos que facilita a inclusão de novos componentes e reutilização dos existentes. Dentre as principais classes existentes, têm-se a classe *Algorithm*, que resolve um *Problem* usando um ou mais *Solution-Set*, além de um conjunto de objetos *Operator*Figura 11

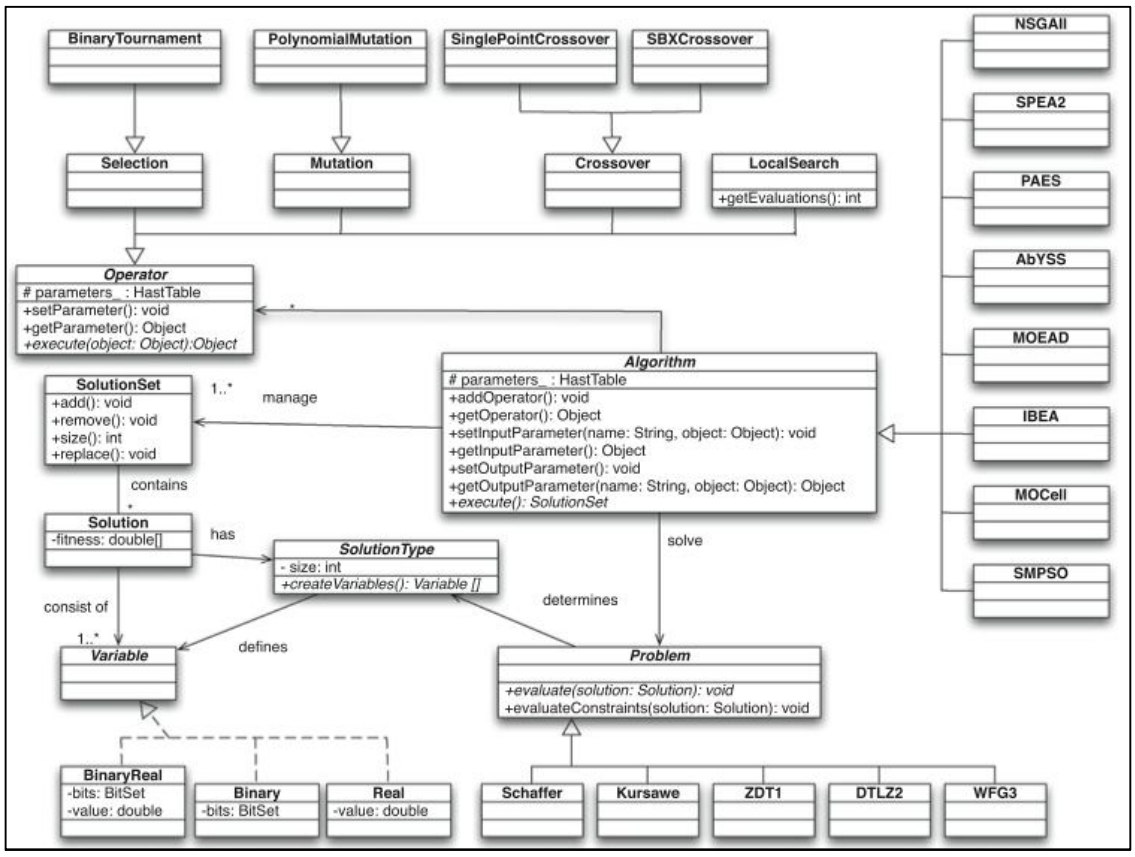


Figura 11. Arquitetura Geral do JMetal (DURILLO e NEBRO, 2011).

O framework possui uma terminologia genérica com os nomes das classes para que possam ser usadas com qualquer meta-heurística. No contexto de algoritmos evolucionários, populações e indivíduos correspondem às classes *SolutionSet* e *Solution*.

O mesmo pode ser aplicado para algoritmos *Particle Swarm Optimization*, no que diz respeito aos conceitos de *Swarm* e *Particles*.

A classe *Algorithm* representa a superclasse que precisa ser herdada para representar uma meta-heurística. Cada algoritmo utiliza um conjunto de parâmetros que podem ser manipulados pelos métodos *addParameter()* e *getParamenter()*. Além disso, um algoritmo utiliza alguns operadores, que no caso de algoritmos genéticos, podem ser para recombinação e seleção de indivíduos e podem ser adicionados e acessados pelos métodos *addOperator()* e *getOperator()*. Finalmente, o principal método dessa classe é *execute()*, responsável pelo início da execução do algoritmo.

Um *SolutionSet* é um conjunto de *Solution* que é associado a um *SolutionType*, que por sua vez é formada por um conjunto de variáveis. Essas variáveis podem ser do tipo inteiro, real, podem ser uma combinação de um mesmo tipo ou de tipos diferentes. Além disso, é possível criar seu próprio tipo estendendo a classe *Variable*.

A classe *Problem* possui dois métodos principais: *evaluate()* e *evaluateConstraints()*. Ambos recebem como parâmetro um objeto *Solution*, contendo uma solução candidata. O primeiro método realiza o cálculo dos valores para os objetivos de otimização, enquanto o segundo verifica se alguma restrição é violada. Esses métodos precisam ser implementados para um novo problema.

O JMetal possui algumas implementações para a classe *Operator*. Para algoritmos genéticos, temos *Simulated Binary Crossover*(SBX) como operador de *crossover*, *BinaryTournament* como operador de seleção e *PolynomialMutation* como operador de mutação, além de outros.

2.4.1.2. Indicadores de Qualidade

JMetal utiliza alguns indicadores de qualidade para problemas multiobjetivos de otimização para avaliar um conjunto de soluções obtidas. Alguns deles têm a intenção de medir apenas as propriedades de convergência ou de diversidade, e outros levam em conta dois critérios. Entre os indicadores disponíveis no JMetal estão:

- ***Generational Distance* (GD):** este indicador foi introduzido por Van Veldhuizen e Lamont (1998) para medir até que distância estão as soluções geradas da fronteira de Pareto. Um valor de GD = 0 indica que todos os elementos são gerados na fronteira de Pareto;
- ***Inverse Generational Distance* (IGD):** ela é uma variante da *Generational Distance*. Ele mede as distâncias entre cada solução pertencente à fronteira de Pareto e a soluções calculadas;

- ***Hypervolume* (HV):** este indicador calcula o volume, dado os valores de objetivos calculados das soluções geradas, coberto por membros de um conjunto de soluções não-dominadas Q (ZITZLER E THIELE, 2006);
- ***Epsilon*:** dada as soluções geradas para um problema A, este indicador é uma medida da menor distância que seria necessária para traduzir cada solução em A para dominar a fronteira de Pareto calculada (KNOWLES et al., 2006).
- ***Disseminação (Spread) ou* Δ:** este indicador mede a extensão da disseminação pelo conjunto de soluções computadorizadas (DEB, 2001). Ele assume um valor zero para uma distribuição ideal, apontando para uma expansão perfeita das soluções na fronteira de Pareto;
- ***Generalized Spread*:** o indicador baseia-se no cálculo anterior da distância entre as duas soluções consecutivas, que funciona apenas para problemas de dois objetivos. Este estende essa métrica calculando a distância de um ponto dado a seu vizinho mais próximo (ZHOU, 2006).

JMetal sempre aplica estes indicadores depois de normalizar os valores das funções objetivas, pois os indicadores não são livres na escala arbitrária de objetivos.

2.4.1.3. Interface Gráfica

JMetal possui uma interface gráfica para execução de um algoritmo para resolver um problema de otimização. Nessa tela é possível selecionar o algoritmo para o problema, configurar os parâmetros dele e visualizar graficamente a frente de Pareto para a aproximação obtida (Figura 12). Além disso, arquivos de texto contendo os valores para as funções objetivo e das variáveis de decisão são gerados e armazenados na pasta de trabalho.

É possível também realizar experimentos com a finalidade de ajustar os parâmetros, selecionar mais de um algoritmo para um conjunto de problemas e calcular os indicadores de qualidade medindo o desempenho para cada um dos algoritmos, além de configurar alguns campos, como:

- *Pareto Fronts Folder:* local onde ficarão armazenados os valores para as frentes de Pareto ótima dos problemas incluídos na comparação;
- *LaTeX tables*: caso se deseje gerar um resumo dos resultados dos experimentos utilizando tabelas no formato LaTeX;

- *Wilcoxon Test*: geração de scripts R para realizar testes Wilcoxon[3] (DEMŠAR, 2006) entre pares de algoritmos em cada problema;
- *Boxplot*: representação dos resultados utilizando gráficos Boxplot, sumarizando os resultados de todos os algoritmos para cada problema;
- *Runs*: indica a quantidade de execuções independentes para o problema;
- *Threads*: usado para simular a execução paralela do experimento.

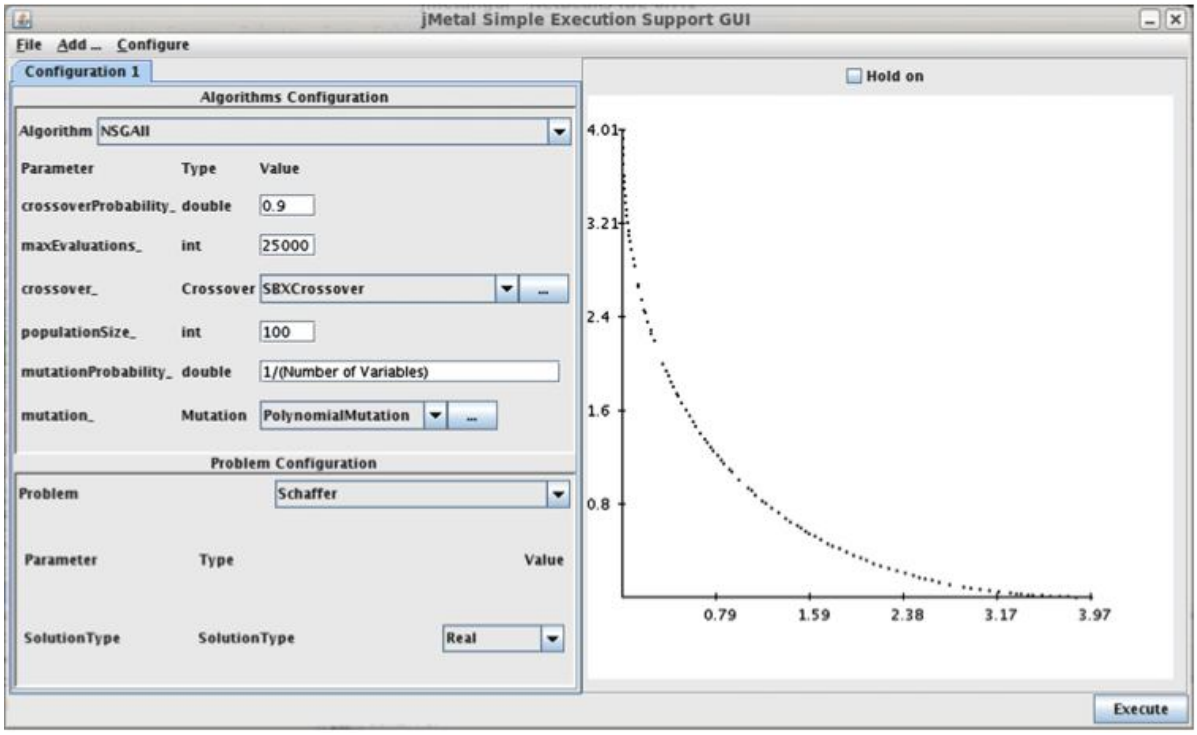


Figura 12. Tela do JMetal mostrando a execução de um algoritmo para resolver um problema (DURILLO e NEBRO, 2011).

2.4.2. Opt4J

Opt4J é um dos frameworks modulares de meta-heurísticas mais robustos para tarefas de otimização mais complexas e utiliza otimização simultânea através da subdivisão em sub-tarefas. Ele inclui o conceito de genótipo (Estrutura da Solução) e fenótipo (Informação obtida). Permite a implementação de alguns tipos de algoritmos como evolutivos multiobjetivos, evolução diferencial, *Particle Swarm Optimizer* (PSO) multiobjetivo e *Simulated Annealing* mono-objetivo. Ele tem uma interface gráfica na qual é possível configurar e visualizar as tarefas de otimização.

[3] Um teste de hipótese estatística não-paramétrica, que permite fazer comparações pareadas entre algoritmos de saber sobre o significado dos dados obtidos.

2.4.2.1. Arquitetura

Para compreender a arquitetura de Opt4J, é necessário introduzir os conceitos de tarefa e algoritmo de otimização (FIGURA 13).

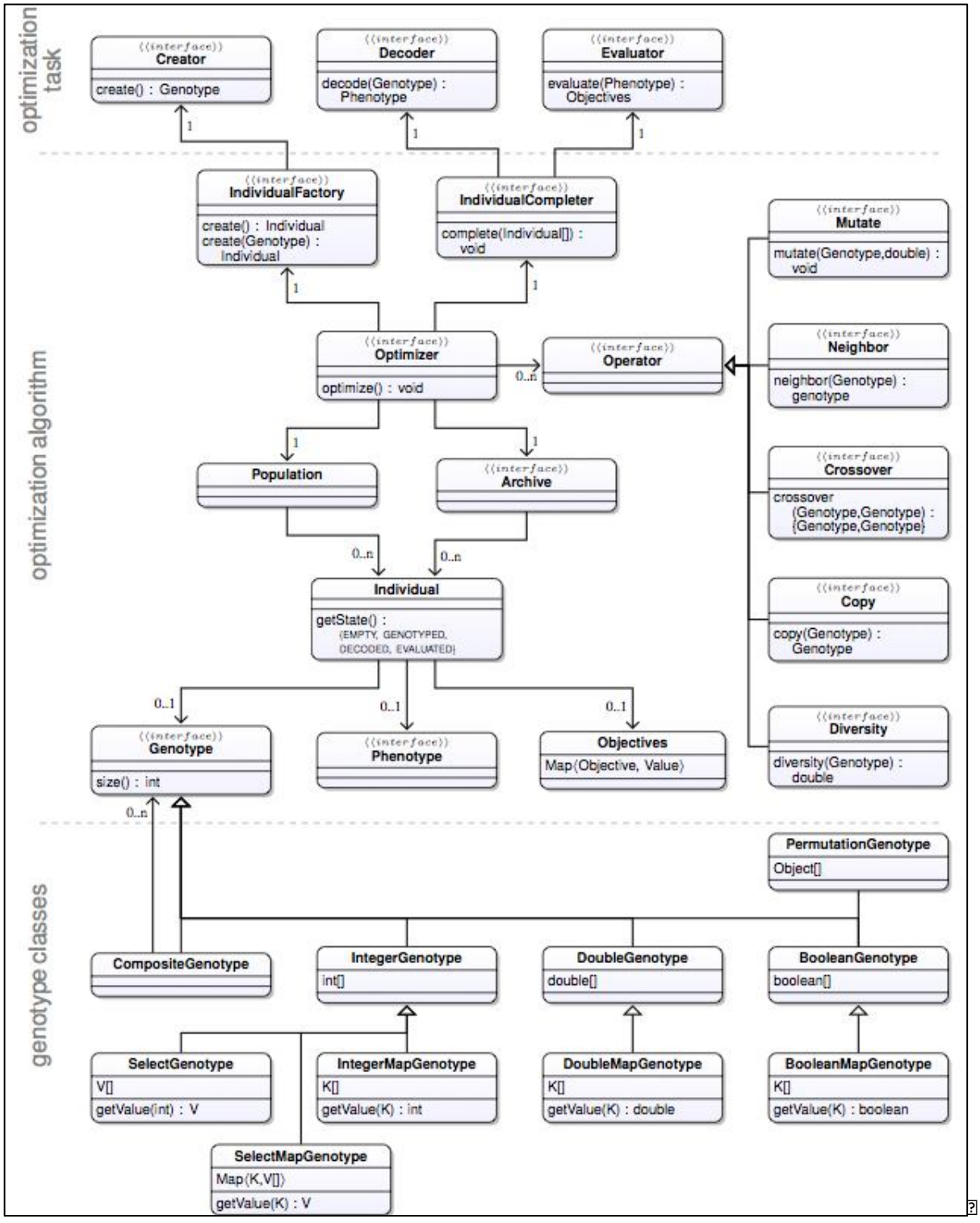


Figura 13. Arquitetura geral do framework Opt4J (LUKASIEWYCZ et al., 2011).

O problema que se deseja otimizar é representado no Opt4J como uma Tarefa de Otimização (*Optimization Task*) e pode é definido pela implementação das interfaces *Creator*, *Decoder* e *Evaluator*. A interface *Creator* é responsável por construir diversos *Genotype*, a interface *Decoder* converte um *Genotype* em um *Phenotype* e a interface *Evaluater* calcula os *Objectives* para o *Phenotype*.

Um *Genotype* refere-se à estrutura de uma solução e como ela é representada, como por exemplo, vetores de valores inteiros (*IntegerGenotype*), reais (*DoubleGenotype*), binários (*BooleanGenotype*) ou permutações (*PermutationGenotype*). Um *Phenotype* refere-se à informação que se obtém da estrutura de dados. Por exemplo, a estrutura pode ser uma cadeia de caracteres, enquanto a informação seria a frase formada por essa cadeia.

O conceito de Algoritmo de Otimização (*Optimization Algorithm*) refere-se a um algoritmo ou meta-heurística aplicado ao problema e é representado pela *interface Optimizer*. Um *Optimizer* manipula *Population* e *Archive*. Um *Population* contém os *Individuals* da atual interação, enquanto o *Archive* contém as soluções não dominadas de todo o processo de otimização.

Um *Individual* é composto por um *Genotype* (representação genética), um *Phenotype* (representação decodifica) e por *Objectives* (contendo os resultados dos objetivos avaliados para um *Phenotype*). Um *Optimizer* pode usar qualquer *Operator* para varrer um *Genotype* e usar a *interface IndividualFactory* para construir novos *Individuals.* Além disso, existe a *interface IndividualCompleter* que auxilia na construção de um *Individual*, evitando que *Decoder* e *Evaluator* sejam acessados diretamente.

Um *Operator* pode ser um *Mutate* ou *Crossover* para algoritmos genéticos, ou um *Neighbor* usados pelas meta-heurísticas *Simulated Annealing* ou *Tabu Search.* Pode ser também um *Algebra* usado por algoritmos como *Particle Swarm Optimization* e *Differential Evolution*, ou um *Copy* ou *Diversity* usados em algoritmos de busca local.

2.4.2.2. Interface Gráfica

Com o Opt4J é possível executar os algoritmos utilizando uma interface gráfica. Nela, problemas de otimização e algoritmos são tratados como módulos e automaticamente reconhecidos em suas categorias (Figura 14).

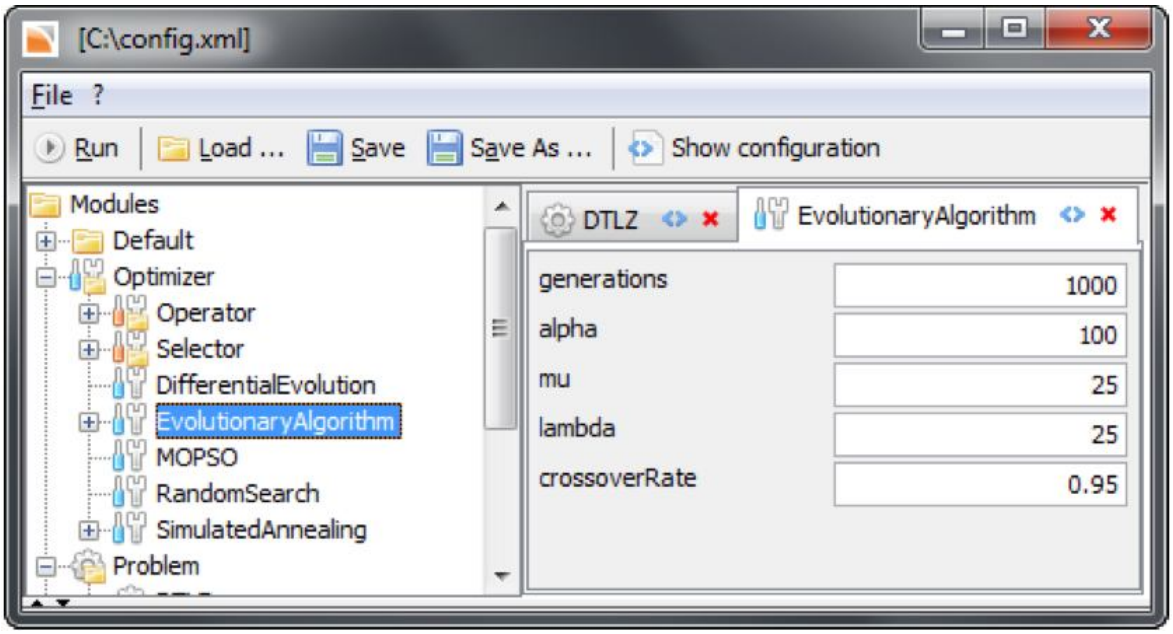


Figura 14. Tela do Opt4J mostrando a execução de uma tarefa (LUKASIEWYCZ et al., 2011).

Para configurar um experimento, é possível selecionar o problema e o algoritmo (*Otimizador*) e configurar seus parâmetros. Essas configurações podem ser salvas ou lidas de arquivos XML. Além disso, é possível monitorar e controlar (pausando ou parando) o processo de otimização, bem como visualizar diversos gráficos como convergência por objetivo e fronteira de Pareto, além da listagem de arquivo de soluções (soluções não-dominadas) (Figura 15) .

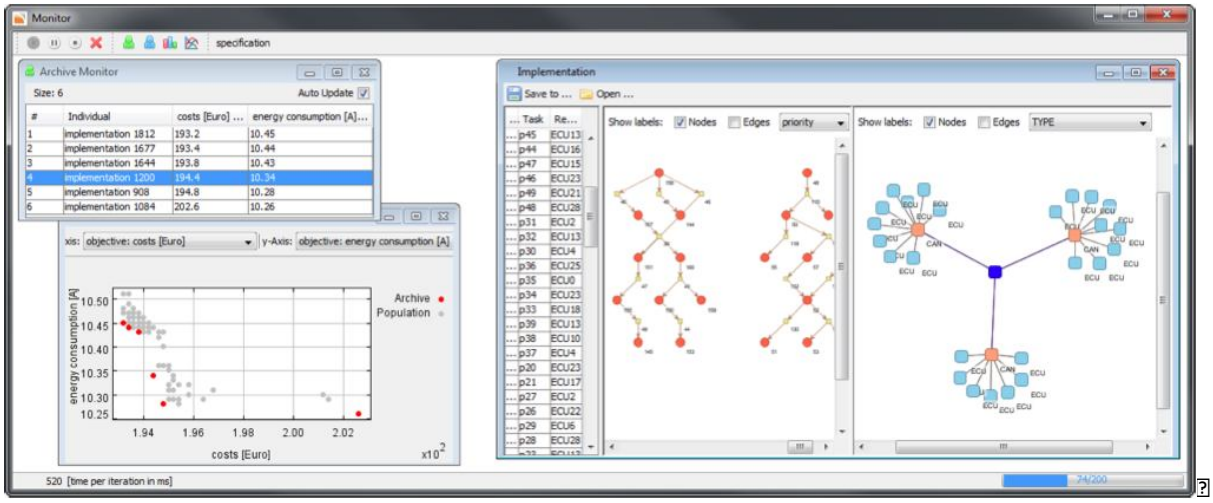


Figura 15. Monitoração Do Processo De Otimização mostrando o arquivo e a projeção bidimensional do arquivo e da população da iteração atual. Além disso, cada implementação do arquivo pode ser visualizada (LUKASIEWYCZ et al., 2011).

2.4.3. EvA2

EvA2 (*Evolutionary Algorithms framework, revised version 2*) é outro framework que implementa meta-heurísticas com ênfase em algoritmos evolucionários. É uma versão revisada do framework JavaEvA (STREICHERT e ULMER, 2005). Ele possui algumas meta-heurísticas disponíveis, como Evolução Diferencial, Otimização por Enxame de Partículas e *Nelder-Mead* ou *Simplex-Simulated Annealing*. Além disso, possui classes abstratas que podem ser herdadas para implementar problemas e algoritmos. EvA2 tem uma interface gráfica na qual se pode comparar o desempenho de diferentes algoritmos.

2.4.3.1. Arquitetura

EvA2 possui uma arquitetura orientada a objetos e baseada na estrutura cliente/servidor (Figura 16). Abstraindo alguns componentes do ciclo de otimização geral, novas implementações podem facilmente ser adicionadas usando as interfaces disponíveis. Na Figura 16 podemos ver as principais interfaces disponíveis no framework e como elas interagem. O problema é representando por *Problem* e os algoritmos ou meta-heurísticas por *Optimizer*. Um recurso interessante é a possibilidade de configurar a forma como um algoritmo termina através da classe *Terminator*. Além disso, existe um pacote utilizado para gerar estatísticas comparativas do experimento com várias execuções e mecanismos de interface com ferramentas de otimização externas como MATLAB.

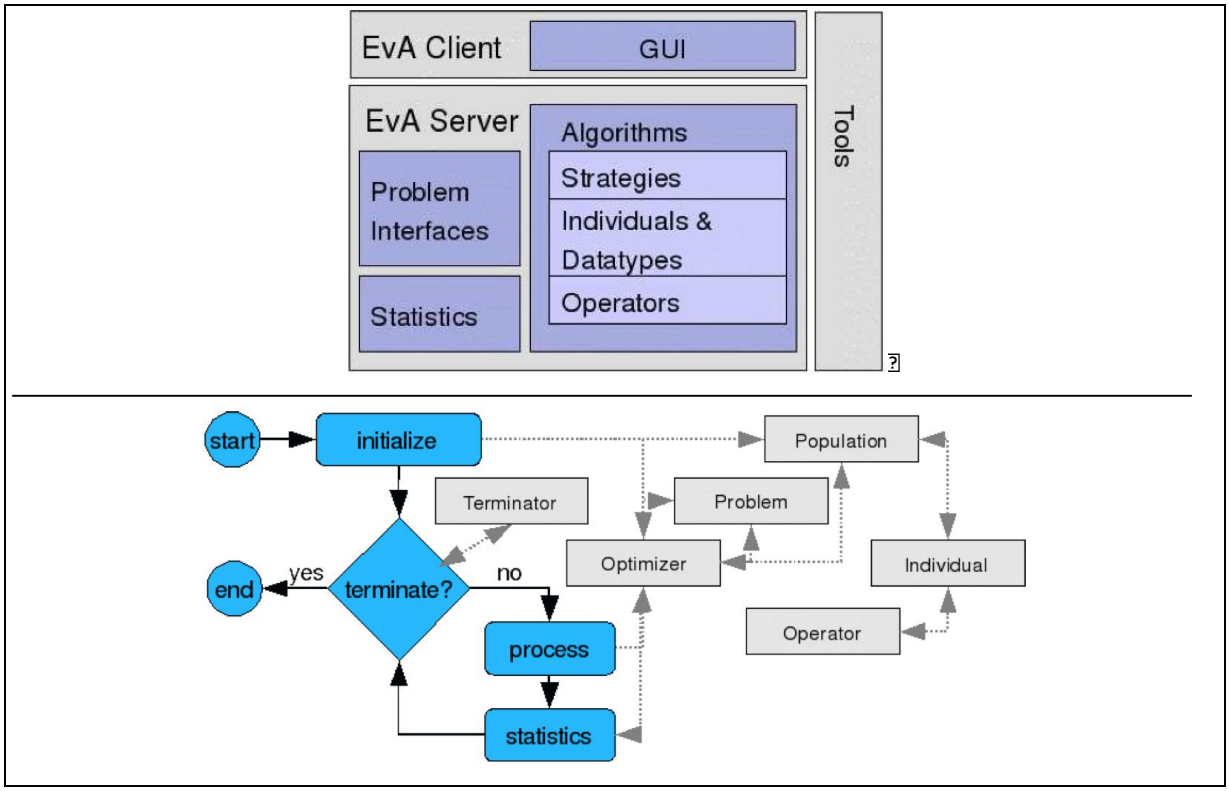


Figura 16. Arquitetura geral do EvA2 (acima); diagrama de processo indicando as principais interfaces (abaixo) (KRONFELD et al., 2010).

Problemas típicos de um único objetivo, problemas multiobjetivo e multimodais são tratados diretamente pelo framework EvA2. Através do mecanismo *Remote Method Invocation* (RMI) do Java, os algoritmos de EvA2 podem ser distribuídos ao longo da rede com base em uma arquitetura cliente-servidor.

EvA2 visa dois grupos de usuários. Em primeiro lugar, o usuário final que não sabe muito sobre a teoria de algoritmos evolutivos, mas quer usar algoritmos evolutivos para resolver um problema de aplicação. Em segundo lugar, o usuário científico que quer investigar o desempenho de diferentes algoritmos de otimização ou quer comparar o efeito de operadores evolutivos ou heurística alternativas ou especializada. O último geralmente sabe mais sobre algoritmos evolutivos ou otimização heurística e é capaz de estender EvA2 adicionando estratégias de otimização específicas ou representações de soluções.

2.4.3.2. Interface Gráfica

Em sua tela principal é possível selecionar os algoritmos (otimizadores) para um determinado problema, configurar os parâmetros de execução e visualizar os resultados para diferentes meta-heurísticas, comparando estes resultados (Figura 17).

É possível utilizar vários tipos de visualização gráficas de resultados, tanto em formato 2D quanto 3D.

2.4.4. ECJ

ECJ é um dos frameworks mais populares de programação genética. Foi lançado em 2004 e desde então novos recursos são adicionados. Ele suporta os algoritmos genéticos, estratégias evolutivas, otimização por Enxame de Partículas e evolução diferencial. Sua força é a programação genética enquanto sua interface gráfica exibe apenas gráficos de resultados.

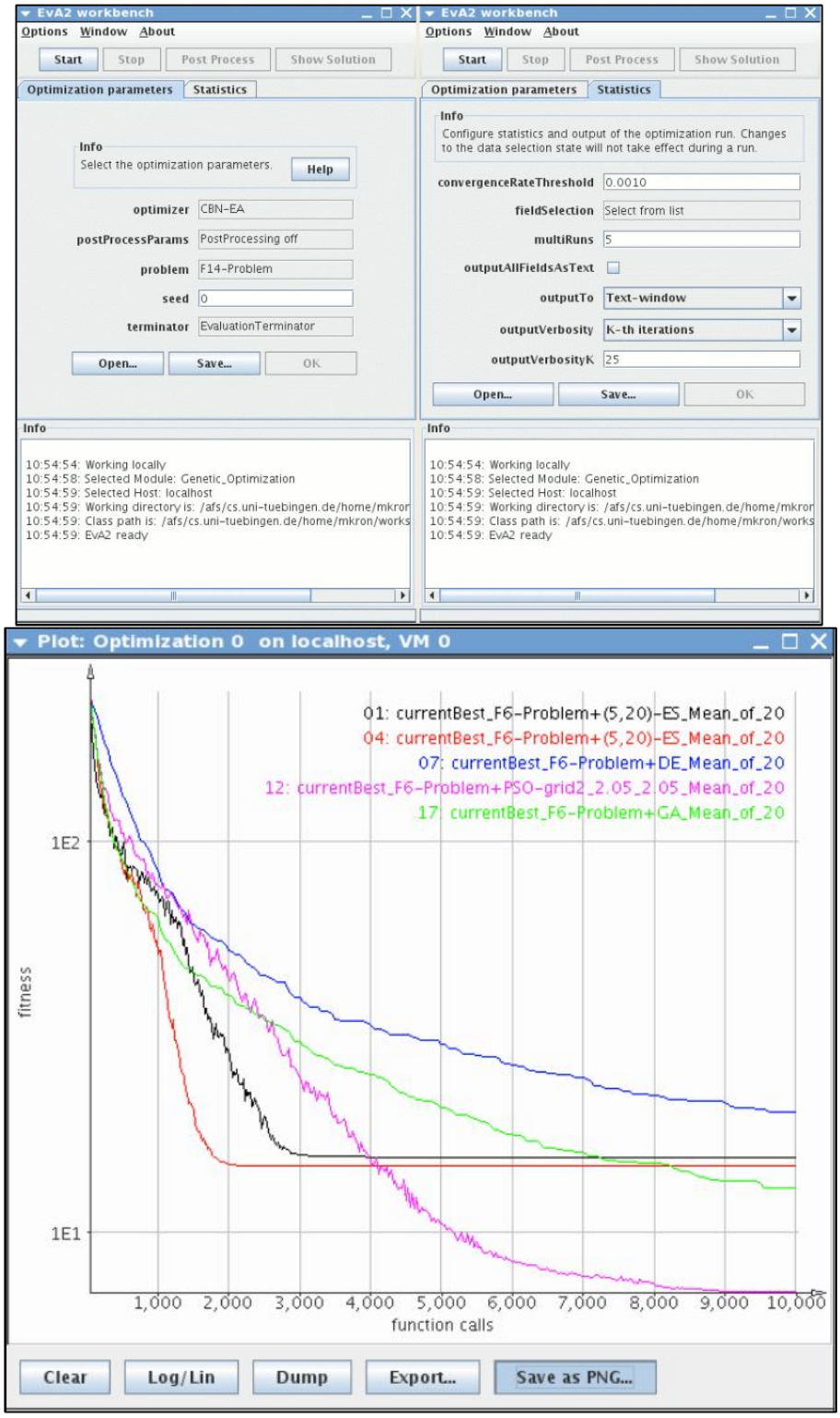


Figura 17. Visualização gráfica e configuração de parâmetros de EvA2 (KRONFELD et al., 2010)

2.4.4.1. Arquitetura

Entre as principais classes do ECJ estão (Figura 18):

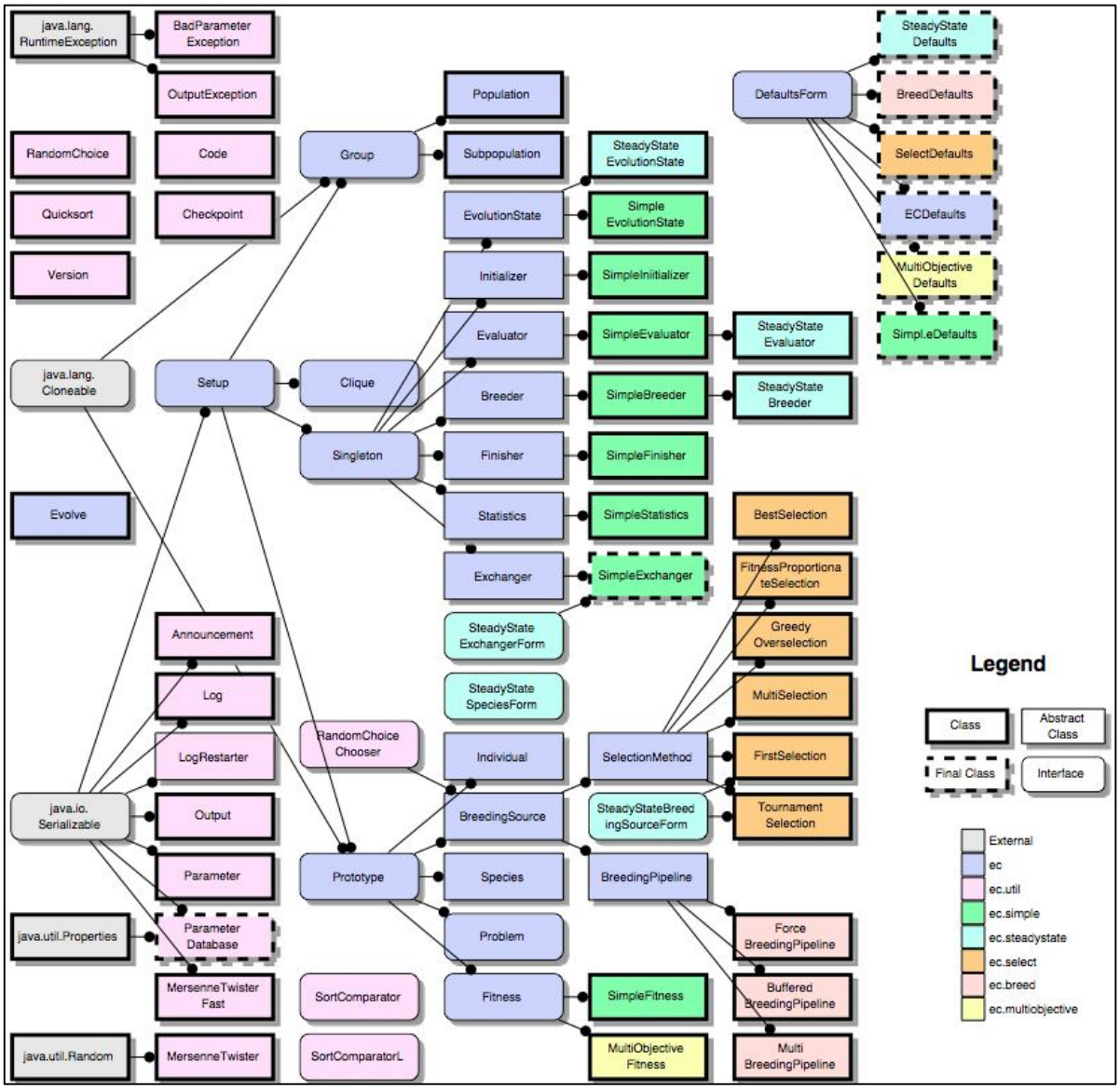


Figura 18. Arquitetura geral do ECJ (LUKE, 2010).

- *Population*: contém os indivíduos no processo evolutivo. Possui como parâmetros a geração atual (iteração) do processo evolutivo e o número total de gerações para ser executado (como ou quando essas duas últimas variáveis são usadas depende do processo evolutivo em questão);
- *Inicializer*: cujo trabalho é criar *Population* inicial no início da execução e um *Finalizer*, cujo trabalho é limpar ao final da execução;
- *Evaluator*: cujo trabalho é atribuir avaliações de qualidade (adequação) para cada membro da população e um *Breeder*, cujo trabalho é produzir uma nova população a partir da população da geração anterior, através de alguma coleção de operadores de seleção e modificação;

- *Exchanger*: opcionalmente exporta membros da população para outros processos ECJ ou importa alguns para adicionar à População. E, finalmente, um objeto de *Statistics*, cujos métodos são chamados em muitos pontos da execução para as estatísticas de saída sobre o desempenho de execução atual.

2.4.4.2. Interface Gráfica

Extensos níveis de representação de resultados são suportados através de três mecanismos básicos de relatórios:

- Saída padrão – avisos, erros e notificações da geração atual são normalmente impressos na tela;
- Arquivos de texto – informações estatísticas como: melhor aptidão, aptidão média, e tamanho do melhor indivíduo, tanto para cada geração quanto no geral;
- Arquivos LaTeX – descrição baseada em árvore pictórica do 'melhor' individual.

Tal como acontece com a maioria dos frameworks, há uma considerável flexibilidade das informações transmitidas e o número de arquivos gerados. Além disso, os usuários podem gerar arquivos de saída adicionais através de adaptações das classes fornecidas pelo framework.

2.5. Análise Comparativa

Na Tabela 1 iremos comparar algumas características de quatro frameworks gratuitos identificados nesta pesquisa e detalhados na seção anterior: JMetal, EvA2, ECJ e Opt4j. As características utilizadas para compará-los são:

- **(MD) Meta-heurísticas disponíveis:** meta-heurísticas implementadas/disponíveis no código-fonte do framework;
- **(EX) Extensíveis:** significa que o framework tem classes Java que implementam algoritmos, problemas de otimização e seus operadores básicos. Assim, é possível utilizar o mecanismo de herança para estender o framework e implementar novos problemas e algoritmos;
- **(MIC) Mecanismo para instanciar cenários:** indica o meio pelo qual novos cenários/problemas possam ser instanciados usando o framework. *As opções seriam:* por codificação ou por uma interface gráfica;
- **(RR) Representação dos resultados:** a forma como os resultados dos experimentos são representados, que pode ser por uma tela da ferramenta onde os

pontos são plotados em gráficos ou apenas armazenados em arquivo de texto para serem exportados;

- **Interface Gráfica de Usuário (GUI) para tradução dados reais para STSP**: processo de tradução de dados para o STSP: cenários, atributos de caracterização, tecnologias e projetos. Se o framework tem os recursos visuais para criar e manter dados utilizados na caracterização de cenários que servem como valores de entrada para o processo de otimização.

Tabela 1. Características dos Frameworks de Otimização.

Framework	MD	EX	MIC	RR	GUI
JMetal	Algoritmos Evolucionários (NSGAII, SPEA2, MOCell, entre outros), Evolução Diferencial, algoritmo Genético Celular, *Particle Swarm*	Sim	Por codificação	Por Algoritmo através de configurações dos parâmetros, por problema, ambos visualizando indicadores de qualidade, fronteira de Pareto, tabelas LaTeX.	Não
Opt4J	Algoritmos Evolucionários, Evolução Diferencial, PSO, *Simulated Annealing.*	Sim	Por codificação	Gráficos de Resultados, fronteira de Pareto, monitor do processo de otimização, Gráficos Y.	Não
EvA2	Algoritmos Evolucionários, *Scatter Search, Binary Scatter Search, Bayesian Optimization Algorithm, Simulated Annealing*, PSO	Sim	Por codificação	Comparativo de desempenho para diferentes algoritmos, interface com MATLAB.	Não
ECJ	Algoritmos Genéticos, co-evolução Mono- e Multi-população, *Particle Swarm Optimization, Differential Evolution*	Sim	Por codificação	Gráficos de Resultados por Algoritmo, arquivos de texto contendo estatísticas de melhor aptidão, aptidão média, tamanho de solução por geração. Tabelas LaTeX.	Não

Analisando as informações da Tabela 1, verificamos que os frameworks fornecem uma gama de recursos, muitos deles em comum, como suporte a implementação a diversos tipos de algoritmos com um diferencial para o JMetal que possui vários algoritmos genéticos já implementados. Além disso, esse framework foi analisado como o mais simples de aprender e utilizar. Opt4J é muito mais poderoso com relação aos outros, possui o foco maior para otimização paralela, o que foge um pouco ao objetivo deste trabalho. Eva2 e ECJ são frameworks mais antigos que os JMetal e Opt4 e mesmo assim possuem uma complexidade de aprendizagem maior que os demais e pouco material científico a respeito. Apesar disso, Eva2 possui uma integração com o MATLAB muito interessante que poderia ser útil para edição de fórmulas dos objetivos de otimização. ECJ foi avaliado com poucos recursos para visualização gráfica de resultados.

Estes frameworks fornecem recursos para usar meta-heurísticas para diferentes problemas disponíveis neles (pode-se implementar novos problemas). No entanto, a adoção desses frameworks para um cenário real de engenharia de software requer um elevado esforço que normalmente não pode ser gasto em um projeto de software, devido aos prazos curtos e os recursos limitados, principalmente quando essas estratégias são referentes a tarefas de tomada de decisão executadas por gerentes de projeto, tais como a seleção de tecnologias de software, pois as habilidades exigidas por estes profissionais não incluem o conhecimento sobre pesquisa com base em algoritmos e meta-heurísticas.

2.6. Considerações Finais

Ao longo deste capítulo foram apresentados os principais conceitos necessários ao entendimento e desenvolvimento da pesquisa, bem como trabalhos relacionados. Estes conceitos são importantes para mostrar a relevância deste trabalho e nos motivar com os resultados já apresentados na literatura.

Além disso, não foi encontrado na literatura nenhum trabalho com as características de instanciação para diversos ambientes de Engenharia de Software, mas apenas soluções específicas para um determinado domínio no qual é aplicada alguma meta-heurística.

Todos os frameworks citados têm um papel importante no framework proposto neste trabalho. Eles seriam integrados na nossa estrutura para executar diferentes experiências usando meta-heurísticas para o STSP, como será descrito no capítulo seguinte.

No capítulo seguinte, intitulado “Framework de Apoio à Instanciação de Técnicas de Seleção de Tecnologias de Software”, será apresentada efetivamente a solução elaborada ao longo desta pesquisa para alcançar o seu objetivo geral.

CAPÍTULO 3: FRAMEWORK DE APOIO À INSTANCIAÇÃO DE TÉCNICAS DE SELEÇÃO DE TECNOLOGIAS DE SOFTWARE

Este capítulo descreve a implementação do framework *de apoio ao problema seleção de tecnologias, mostrando a modelagem do problema como um problema de conjunto dominante mínimo, a estrutura do* framework *desenvolvido e suas funcionalidades.*

3.1. Introdução

Após a revisão da literatura apresentada no capítulo anterior, é possível verificar que SBSE possui diversos trabalhos publicados nas mais diversas áreas da Engenharia de Software. Nestes trabalhos, são utilizadas diversas meta-heurísticas para resolver um problemas específicos. O framework desenvolvido nesta pesquisa busca ampliar a abrangência do problema de seleção de tecnologias de software (STSP), instrumento deste trabalho. O objetivo é prover um framework de aplicação abrangente para o problema, englobando diversos sub-problemas de seleção de tecnologias de engenharia de software, no qual se possa configurar o cenário de interesse, alimentá-lo com um catálogo de soluções e que se possa escolher qual meta-heurística aplicar e investigar se com a combinação dessas soluções é possível gerar novas soluções de forma rápida e que garanta eficácia do resultado obtido.

A seguir será apresentada a modelagem para o STSP como um problema de otimização combinatória, caracterizado por três dimensões que identificam o problema e como será sua abordagem. Também é apresentada a definição formal deste problema.

3.2. Modelagem do STSP como problema de otimização combinatória

O STSP pode ser modelado como um problema de otimização combinatória com o objetivo de selecionar as melhores tecnologias, ou o menor subconjunto de tecnologias, que melhor atenda o(s) projeto(s) de software, de acordo com suas características. Um ou mais critérios podem ser considerados para maximizar a qualidade dos resultados.

Assim, além dos dois principais agentes (tecnologias e projetos de software), os critérios adotados para estimar a adequação de tecnologias de software para projetos também é um fator importante na qualidade da solução proposta. Portanto, nesta seção são definidos as dimensões principais consideradas na modelagem do STSP e o meta-modelo teórico que permite representar vários cenários de seleção de tecnologias no contexto da engenharia de software.

3.2.1. Identificação das dimensões principais

Durante a modelagem do STSP, três dimensões (cada uma com duas opções de instanciação) foram identificadas como relacionadas ao problema (Figura 19). As combinações entre elas resultam em especializações do STSP para diferentes cenários de engenharia de software.

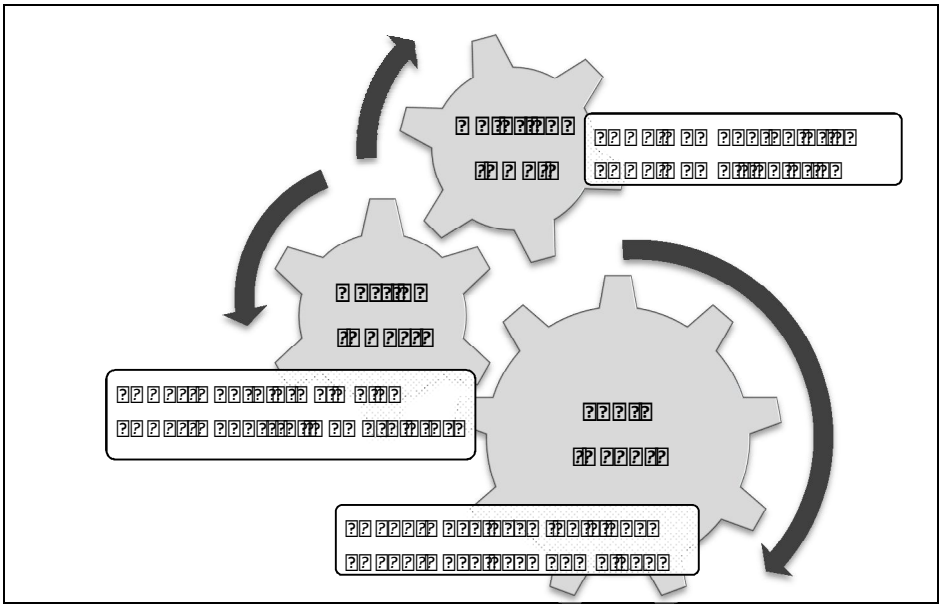


Figura 19. Dimensões relacionadas ao STSP.

- **Modelagem da Dimensão "Fonte" – DSour**

A dimensão **fonte** aborda a origem do processo de seleção, isto é, as tecnologias disponíveis para serem selecionadas no(s) projeto(s) de software. Há duas opções de instanciação para esta dimensão (Figura 19):

- ***DSour1* (seleção individual):** as tecnologias são consideradas individualmente, e apenas uma pode ser selecionada para um (portfólio de) projeto(s) de software;
- ***DSour2* (seleção combinada):** as tecnologias são consideradas de forma combinada e mais de uma pode ser, simultaneamente, selecionada para um (portfólio de) projeto de software.

- **Modelagem da Dimensão "Destino" – DDest**

A dimensão **destino** lida com o cenário em que as tecnologias serão aplicadas (os projetos de software). Há duas opções de instanciação para esta dimensão (Figura 19):

- ***DDest1*** **(único projeto):** as tecnologias serão selecionadas para serem aplicadas em um único projeto;
- ***DDest2*** **(portfólio de projetos):** as tecnologias serão selecionadas para ser aplicadas em uma carteira de projetos, isto é, o conjunto de elementos que representam a dimensão destino será composto da união entre as características de projetos de software que compõem a carteira.

- **Modelagem da Dimensão "Objetivo" – DObj**

A dimensão **objetivo** lida com os critérios a serem considerados para a otimização de funções objetivo. Há duas opções de instanciação para esta dimensão (Figura 19):

- ***DObj1*** **(mono-objetivo):** apenas um objetivo é considerado para a otimização, resultado de um único critério ou vários critérios combinados. Isso resulta na determinação de uma única solução utilizando a função objetivo estabelecida;
- ***DObj2*** **(multiobjetivo):** mais de uma função objetivo (que pode ter conflito) são consideradas para a otimização. Isso resulta na determinação não mais de uma única solução, mas em um conjunto de soluções (não dominadas) considerando todos os critérios (funções objetivo).

3.2.2. STSP como Problema do Conjunto Dominante Mínimo

O STSP pode ser modelado como um problema do conjunto dominante mínimo em um grafo bipartido e (multi) ponderado. Neste modelo geral, cada partição do grafo representa um conjunto independente (os vértices da mesma partição não conectam entre si). O processo de seleção também é geral, isto é, queremos selecionar o subconjunto de vértices (um ou no máximo $|V1|$) que dominam todos os vértices na segunda partição, como descrito abaixo:

> **Modelo Geral – Conjunto Dominante Mínimo (bipartido e (multi) grafo ponderado):** dado um grafo $G = (V, E)$ com a bipartição $V_1, V_2 \in V$ e com pesos w_{ei}, i m associados a cada aresta $e = \{v_1, v_2\}$, onde $v_1 \in V_1$ e $v_2 \in V_2$, determinar o conjunto dominante mínimo $D \in V_1$ em G, que para todo vértice $v \in V_2$ existe pelo menos um vértice $u \in D$, onde a aresta $\{u, v\} \in E$ e os pesos w_{ei} associados a cada aresta $\{u, v\} \in E$ seja o de melhor compromisso.

A Figura 20 apresenta o meta-modelo que podemos usar para instanciar diferentes combinações das dimensões identificadas para o STSP através do relacionamento dos

atributos que caracterizam tanto as tecnologias quanto os projetos. Para instanciar os modelos considerando a dimensão fonte *DSour1* (primeira partição), representando o cenário em que apenas uma tecnologia de software pode ser selecionada, a dimensão destino pode ser instanciada com a opção *DDest1* (cada vértice na segunda partição representa um único projeto) ou *DDest2* (cada vértice representa um componente conectado, nas quais sobreposições entre os componentes podem acontecer, indicando características semelhantes entre os projetos).

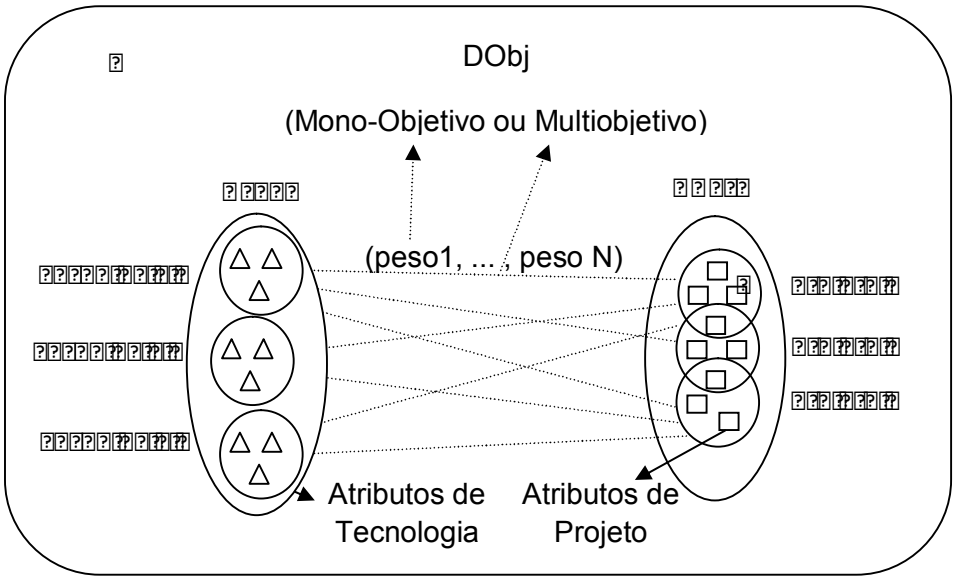


Figura 20. Grafo bipartido representando o meta-modelo para o STSP considerando o dimensões *DSour*, *DDest* e *DObj*.

Ainda na (Figura 20), podemos instanciar modelos considerando a dimensão fonte *DSour2* (primeira partição), que representa a situação em que um subconjunto de tecnologias pode ser selecionado. Neste caso, o modelo deve permitir que vértices na primeira partição gerada a partir de mais de uma tecnologia sejam conectados com vértices da segunda partição. Assim como no cenário anterior, a dimensão destino pode ser instanciada com as opções *DDest1* e *DDest2*.

A Figura 20 representa também o objetivo de dimensão (*DObj*), em que há duas opções: *DObj1* e *DObj2*. A instanciação *DObj1* para a dimensão *objetivo* representa o cenário em que apenas um peso está associado a cada aresta no grafo bipartido, ou seja, apenas uma função objetivo é considerada. A instanciação *DObj2* para a dimensão *objetivo* representa o cenário onde cada aresta é multi-ponderada, ou seja, cada um dos critérios a ser considerado durante a seleção de tecnologias é mapeado em uma função objetivo independente, e seu valor é representado como um peso nas bordas.

A partir da identificação das dimensões do STSP, podemos observar que oito diferentes cenários para a seleção de tecnologias poderiam ser instanciados, combinando

todas as opções em todas as dimensões (Tabela 2). Portanto, a instanciação de uma infraestrutura para apoiar a seleção de tecnologias para cada atividade em engenharia de software que pode ser modelado como um STSP seria uma tarefa complexa em um projeto de software quando executado manualmente, principalmente em atividades nas quais mais de uma instanciação pode ser feito.

Tabela 2. Possíveis Cenários Instanciados para o STSP em todas as dimensões.

Cenário	Origem – *DSour*	Destino – *DDest*	Objetivo – DObj
#1	DSour1 – seleção individual	DDest1 – único projeto	DObj1 – mono-objetivo
#2	DSour1 – seleção individual	DDest1 – único projeto	DObj2 – multiobjetivo
#3	DSour1 – seleção individual	DDest2 – portfólio de projetos	DObj1 – mono-objetivo
#4	DSour1 – seleção individual	DDest2 – portfólio de projetos	DObj2 – multiobjetivo
#5	DSour2 – seleção combinada	DDest1 – único projeto	DObj1 – mono-objetivo
#6	DSour2 – seleção combinada	DDest1 – único projeto	DObj2 – multiobjetivo
#7	DSour2 – seleção combinada	DDest2 – portfólio de projetos	DObj1 – mono-objetivo
#8	DSour2 – seleção combinada	DDest2 – portfólio de projetos	DObj2 – multiobjetivo

A escolha dessas dimensões reflete no tipo de otimização a ser realizada. Com o objetivo de reduzir o esforço para instanciar e utilizar infraestruturas para apoiar a seleção de tecnologias de software para diferentes atividades de engenharia de software, a próxima seção descreve a estrutura que implementa o meta-modelo proposto nesta seção e permite a instanciação de infraestruturas para apoiar a seleção de tecnologias de software.

3.3. Estrutura do Framework Proposto

O modelo proposto foi concebido para tornar viável a aplicação de estratégias baseadas em busca como suporte para o STSP. Assim, ele deve fornecer recursos que permita a instanciação de infraestruturas computacionais para lidar com os diferentes cenários de engenharia de software que podem ser classificados como STSP, considerando o metamodelo apresentado na seção anterior.

A próxima subseção descreve os requisitos a serem implementados no framework, apoiando a seleção de tecnologias em projetos de software.

3.3.1. Requisitos do Framework Proposto

O framework proposto deve atender aos seguintes requisitos:

- Deve proporcionar uma modelagem comum para o STSP de acordo com a solução definida na seção anterior;

- Deve permitir a caracterização de tecnologias de software, indicando os atributos utilizados nesta tomada de decisão e seus respectivos pesos neste processo;
- Deve fornecer um mecanismo para integrar a implementação do meta-modelo do STSP com os principais algoritmos baseados em busca utilizados na área de SBSE, tais como: algoritmo genético (e suas variações), arrefecimento simulado, busca tabu, subida de encosta, colônia de formigas, e outros. Assim, a estrutura poderia ser desenvolvida com a adição de novos algoritmos.

3.3.2. Visão Geral

O framework STSP foi projetado para funcionar como uma camada superior atuando em conjunto com outras frameworks de otimização, servindo como uma interface para os engenheiros de software utilizarem várias meta-heurísticas disponíveis nestes frameworks de otimização disponíveis na literatura técnica (Figura 21).

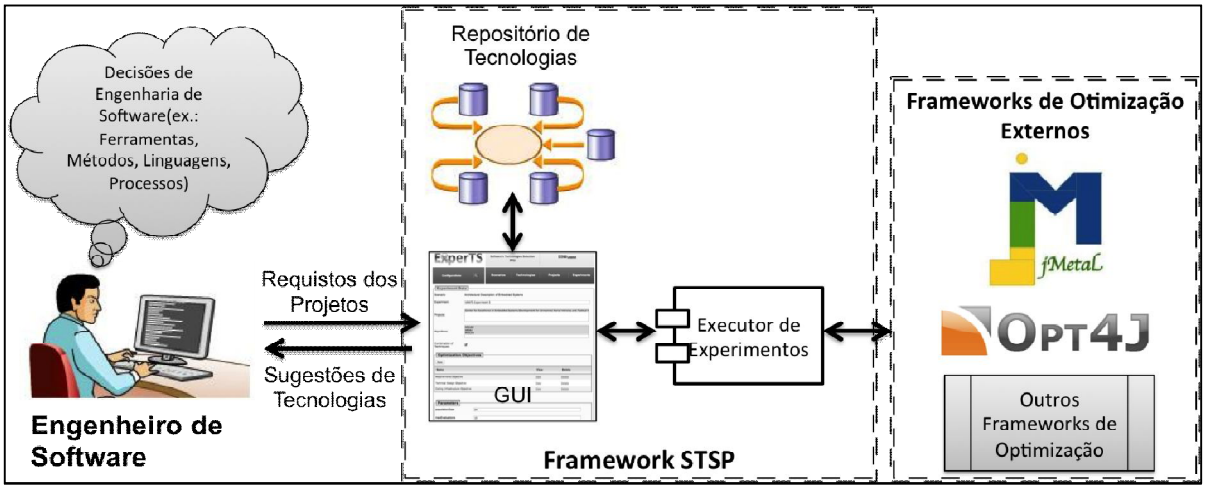


Figura 21. Visão geral do framework stsp.

Usando a interface gráfica do usuário do framework STSP (GUI), os usuários podem configurar cenários de engenharia de software para realizar a seleção de tecnologias de software e montar repositórios de tecnologias que correspondem a instanciação física do corpo de conhecimentos de técnicas contendo técnicas encontradas na literatura técnica. Existe um componente que suporta a configuração de experimentos (*Executor de Experimentos* na FIGURA 21). Este componente implementa o modelo descrito na seção 3.2.2 e integra os frameworks de otimização externos, fornecendo-lhes dados de entrada e recebendo os resultados gerados deles.

O componente *Executor de Experimentos* compreende os métodos responsáveis para calcular os valores para os objetivos de otimização, isto é, as funções de aptidão

(Figura 22). Avalia as funções de aptidão usando a fórmula *coeficiente de Jaccard*[4] (JACCARD, 1901). Assim, ele conta quantos atributos exigidos pelo projeto de software estão sendo atendidos pelas tecnologias selecionadas. Por exemplo, se o projeto que pretende aplicar alguma tecnologia específica exige dois atributos (ex.: linguagem de programação Java e plataforma Linux), e a tecnologia selecionada atende apenas uma delas, a função de aptidão retornará 0,5 (representando 50%).

$$\text{Função de Aptidão} = \frac{\sum Atributos\ Requeridos\ pelo\ Projeto \cap Atributoss\ Supridos\ pelas\ Tecnicas}{\sum Atributos\ Requeridos\ pelo\ Projeto}$$

Figura 22. Fórmula matemática para a função de aptidão do framework STSP.

A próxima subseção descreve o processo definido para a aplicação do framework em um cenário real de software de seleção de tecnologias.

3.3.3. Processo para Instanciar Infraestruturas para apoiar a seleção de tecnologias de software

O processo para criar uma instância de infraestrutura computadorizada para apoiar a seleção de tecnologias de software segue uma abordagem dividida em fases. Assim, o framework proposto funciona como um "meta-modelador" do processo de seleção de tecnologias, possibilitando sua aplicação em diferentes cenários de engenharia de software. Três passos são definidos para apoiar esta instanciação, tal como descrito na (Figura 23).

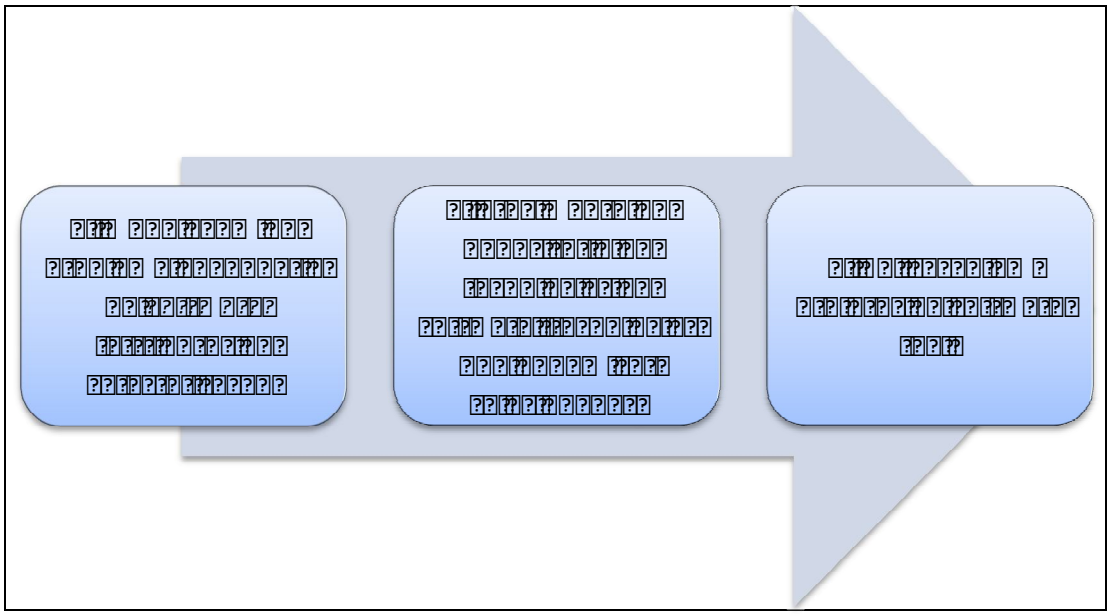


Figura 23. Representação gráfica do processo de instanciação do Framework proposto.

[4] *Jaccard Coefficient* (1901) compara o número de elementos similares e o total de elementos no conjunto (também chamado coeficiente de similaridade).

(1) **Modelagem do Problema de Engenharia de Software:** atributos que caracterizam tecnologias de software para serem selecionadas e o projeto de software (ou portfólio) devem ser identificados. Assim, o tipo de seleção deve ser definido de acordo com as dimensões apresentadas na seção anterior: Origem, Destino e Objetivo;

(2) **Provimento do repositório de tecnologias de software:** as tecnologias de software que serão apresentadas para a seleção de projetos de software devem ser fornecidas a partir da literatura técnica, e o framework proposto deve fornecer recursos para criar e manter este repositório;

(3) **A aplicação em projetos de software reais:** após concluir as etapas anteriores, o framework será instanciado em projetos de software reais permitindo a escolha de algoritmos baseados em busca disponíveis e as características do(s) projeto(s) de software nos quais as tecnologias de software selecionadas poderiam ser aplicadas.

3.4. Instanciação da Modelagem do STSP para a Seleção de Técnicas de Teste de Software

Foi realizada a instanciação do metamodelo proposto na seção anterior para seleção de técnicas de teste baseado em modelos (em inglês *Model-Based Testing Techniques Selection Problem* – MBTTSP), um subproblema de STSP. Esta implementação foi desenvolvida como uma extensão de duas técnicas de seleção de técnicas de teste baseado em modelos disponíveis na literatura técnica e implementadas na ferramenta Maraká: *Porantim* (DIAS-NETO, 2009) e *Porantim-Opt* (DIAS-NETO et al., 2011). Assim, a instanciação do metamodelo realizada nesta pesquisa também foi implementada na ferramenta Maraká.

Nesta instanciação, a dimensão fonte foi instanciada como *DSour2* (seleção combinada), a dimensão destino foi instanciada como *DDest1* (para um único projeto) e a dimensão objetivo foi instanciada como *DObj2* (dois objetivos foram considerados). Os objetivos definidos são os mesmos utilizados pelas técnicas *Porantim* e *Porantim-Opt*, o que permite uma comparação dos resultados das três técnicas, são:

- ***SWP_Cov* (Cobertura de Projeto de Software):** indicador da proporção de características de projeto do software que estão sendo cobertas pelas técnicas de teste selecionadas. Este objetivo gera um número entre 0% e 100%. O propósito é maximizar este objetivo;
- ***SavModEffort* (Esforço de modelagem Salvo):** indicador da proporção de modelos requeridos para utilizar as técnicas selecionadas já previstos no

processo de desenvolvimento de software e quantos modelos precisam ainda ser desenvolvidos durante o processo de geração de teste. Este objetivo também gera um número entre 0% e 100%. O propósito também é maximizar este objetivo.

O MBTTSP foi modelado como um Algoritmo Multiobjetivo Evolucionário (Genético), e foi chamado *Porantim-GA* (GRANDE et al., 2012), uma extensão das técnicas *Porantim* e *Porantim-Opt* propostas em Dias-Neto e Travassos (2009) e Dias-Neto et al. (2011). Nesta modelagem, um cromossomo corresponde à solução de um problema e é formado pela combinação de um, dois ou três elementos que são técnicas de testes. Portanto, o tamanho do cromossomo é variável contendo uma, duas ou três técnicas (Figura 24). Um cromossomo seria inválido quando uma técnica aparece repetida no mesmo cromossomo.

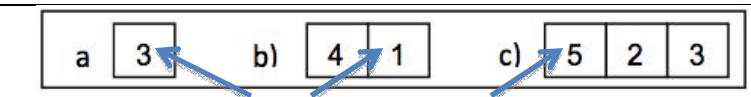


Figura 24. Cromossomos com 1 (um), 2 (b) e 3 elementos (c).

Como estratégia de evolução, uma população inicial foi gerada com soluções contendo três técnicas selecionadas aleatoriamente, sem repetições, a fim de garantir uma maior variabilidade. Depois disso, cada geração do algoritmo genético de 50 diferentes soluções (indivíduos) são selecionadas entre aquelas que têm o melhor valor para a função de adequação na geração anterior – o elitismo (pelo menos 6% ou 3 soluções). O elitismo é realizado obtendo o conceito da fronteira de Pareto considerando os dois objetivos de otimização. Caso a quantidade de soluções na fronteira de Pareto não atinja a quantidade mínima definida pelo parâmetro configurado, a quantidade é preenchida com as melhores soluções dominadas.

A fórmula para calcular as funções de adequação para os dois objetivos apresentadas a seguir são as mesmas utilizadas na abordagem *Porantim-Opt* e podem ser encontradas em Dias-Neto et al. (2011) e Dias-Neto e Travassos (2009a) respectivamente.

$$SWP_{Cov} = 1 - \frac{\sum_{i=1}^{11} \text{[illegible]}_i}{\text{[illegible]}}$$

Utilizando a fórmula de indicador: $Indicador = \frac{Atributos\ proj - Atributos\ proj \cap t\acute{e}cnicas}{Atributo\ proj}$

Onde:

- $Atributos\ proj$: é o número de atributos requeridos pelo projeto de software;
- $Atributos\ proj \cap t\acute{e}cnicas$: é o número de diferentes atributos requeridos pelo projeto de software e providos pelas técnicas MBT selecionadas.

Calculados para os seguintes atributos:

1. Modelos comportamentais / estruturais previstos pelo projeto de software;
2. Paradigma de Desenvolvimento adotado no projeto de software;
3. Linguagem de programação usada para o desenvolvimento de software;
4. Habilidades fornecidas pela equipe de testes;
5. Plataforma de execução Software;
6. Características de qualidade de software necessárias para o desenvolvimento;
7. Ferramentas de Suporte;
8. Tecnologia utilizada para a modelagem de software;
9. Níveis de testes a serem aplicados no projeto de software;
10. Tipo de falhas a ser revelado;
11. Tipo de técnica de teste a ser aplicado.

$$SavModEffort = 1 - \frac{Mod[b] * 1 + Mod[c] * 2 + Mod[d] * 3 + Mod[e] * 4}{Mod[a] + b + c + d + e] * 4}$$

Onde: Mod[cat]: é o número de modelos usado pela técnica MBT ou provida pelo projeto de software em uma das categorias abaixo:

a) Se o modelo é construído ao longo do processo de desenvolvimento de software, mas não é exigido pelas técnicas MBT selecionados: não há esforço de modelagem para adaptá-lo, então esses modelos não são considerados durante a análise;
b) Se o modelo é construído ao longo do processo de desenvolvimento de software e é necessário apenas por uma técnica MBT selecionada: há um esforço de modelagem mínimo para adaptar este modelo para a técnica MBT, então vamos considerar um peso de 1 para executar esta tarefa;
c) Se o modelo é construído ao longo do processo de desenvolvimento de software e é necessário mais do que uma técnica MBT selecionado: há um esforço de modelagem pequena (maior do que o caso anterior) para adaptar este modelo para as técnicas de MBT, então vamos considerar um peso 2 para executar esta tarefa;
d) Se o modelo não é construído ao longo do processo de desenvolvimento de software, mas é exigido por apenas uma técnica MBT selecionada: há um esforço de modelagem significativa (superior aos dois casos anteriores) para construir e adaptar este modelo para a técnica MBT selecionada, então nós considerar um peso 3 para executar esta tarefa;
e) Finalmente, se o modelo não é construído todo o processo de desenvolvimento de software, mas é necessária mais do que uma técnica MBT selecionada: existe um elevado esforço de modelagem (maior do que os três processos anteriores) para criar e adaptar este modelo para as

técnicas MBT selecionadas (cada uma pode exigir uma adaptação diferente), então vamos considerar um peso 4 para executar esta tarefa.

Para as outras soluções, a estratégia de geração tem duas operações: *crossover* e mutação. O primeiro é baseado na combinação de duas soluções escolhidas aleatoriamente a partir da população anterior. A combinação depende do tamanho das soluções selecionadas, que pode gerar uma nova solução com duas ou três técnicas dependendo do tamanho da solução pai. A segunda operação (mutação) modifica uma solução existente. Se ela é composta de apenas 1 técnica, 1 ou 2 técnicas são selecionadas aleatoriamente e adicionadas para formar uma nova solução. Caso contrário, 1 técnica é excluída da solução selecionada.

Para parar o processo de evolução, dois critérios são utilizados: (1) após a produção de 100 gerações ou (2) quando as cinco melhores soluções são as mesmas (não há nenhuma evolução) passadas 10 iterações consecutivas. O pseudocódigo do algoritmo implementado em *Porantim-GA* está descrito na Figura 25.

Pseudocódigo de Porantim-GA

1. Escolha da população inicial de indivíduos
2. Avaliar a função *fitness* de cada indivíduo da população
3. Repita sobre esta geração até que os critérios de término seja atingido:
   - 3.1. Selecionar os indivíduos de melhor fitness para a operação de seleção
   - 3.2. Produzir novos indivíduos por meio de operações de crossover e mutação
   - 3.3. Avaliar a função fitness de novos indivíduos
   - 3.4. Substituir os indivíduos com piores fitness da população pelos novos indivíduos
4. Selecione as soluções trade-off (considerando os dois objetivos) obtidos na última geração.

Estratégia de seleção

→ Dar a população real:

1. Ordem crescente da população que usa a função de aptidão;
2. Selecione as 3 (6%) melhores soluções (Frente de Pareto);
3. Copie as soluções selecionadas para a nova população;

Cálculo da População Inicial

→ Repetir até que as técnicas MBT do repositório esteja vazio:

1. T1 = Selecione uma técnica aleatoriamente;
2. Excluir T1 das técnicas MBT do repositório;
3. T2 = Selecione uma técnica aleatoriamente;
4. Excluir T2 das técnicas MBT do repositório;
5. T3 = Selecione uma técnica aleatoriamente;
6. Excluir T3 das técnicas MBT do repositório;
7. Solução = array (T1, T2, T3);
8. População_Inicial [] = solução;

Estratégia mutação

→ Dada a população atual:

→ Enquanto solu. geradas <22(44%)

1. T1 = Selecione uma técnica aleatoriamente;
2. L1 = tamanho de T1;
3. Se L1> 1
4. Então NovaSolução = {T1} - {aleatoriamente selecionado técnica de T1};
5. Senão
   a. T2 = Selecionar zero, um ou duas técnicas aleatoriamente;
   b. NovaSolução = T1 ∪ T2;
6. Verifique se NovaSolução é válida e que não existe na nova população;

Estratégia CrossOver

→ Dada a população atual:

→ Enquanto soluções geradas <25(50%)

1. T1 = Selecione um indivíduo (solução) de forma aleatória;
2. T2 = Selecione outro indivíduo aleatoriamente;
3. L1 = tamanho de T1;
4. L2 = tamanho de T2;
5. NovaSolução = Lado esquerdo do T1 (1 ou 2 técnicas) ∪ lado direito do T2 (1 ou 2 técnicas);
6. Verifique se NovaSolução é válida e que não existe na nova população;

Estratégia de Término

→ Enquanto Iterações <100 e Convergência == 10:

1. NovaPopulação = {estratégia de seleção};
2. Se NovaPopulação = melhor (AtualPopulação)
   a. Então Convergência + +;
   b. Senão Convegência = 0;
3. NovaPopulação= NovaPopulação ∪ {estratégia de Crossover};
4. NovaPopulação= NovaPopulação ∪ {estratégia de mutação};

Figura 25. Pseudocódigo de *Porantim-GA* (GRANDE et al., 2012).

O experimento realizado teve como objetivo comparar a eficácia dos três métodos de seleção (*Porantim*, *Porantim-Opt* e *Porantim-GA*) para dois projetos previamente

selecionados: Video Manager e Parking System. Estes projetos foram definidos num pacote publicado em (DIAS-NETO e TRAVASSOS, 2009), e eles foram utilizados na avaliação da versão original da *Porantim*.

Como *benchmark* para este experimento, foi utilizado o repositório de técnicas de testes baseado em modelos fornecido em Maraká [5] que representa o corpo de conhecimento de *Porantim*. Ele contém 219 diferentes técnicas de testes e reais identificadas na literatura técnica. Se analisados manualmente combinando três técnicas de teste, este repositório possibilita a instanciação de mais de 10.000.000 de soluções (219 x 218 x 217). Assim, as abordagens de seleção foram avaliadas simulando a sua aplicação em situações reais e complexas.

Cada projeto de software foi analisado individualmente. Assim, cada abordagem de seleção foi executada coletando os dois objetivos (*SWP_Cov* e *SavModEffort*). Usando *Porantim* e *Porantim-Opt*, apenas uma execução foi realizada, pois os resultados gerados são sempre as mesmas para um projeto de software. No caso de *Porantim-GA*, decidimos executar o algoritmo 10 vezes para cada projeto e, em seguida, nós selecionamos os melhores e mais recorrentes resultados. Ao final, 50 soluções foram geradas utilizando cada abordagem (resultados de alguns deles repetidos).

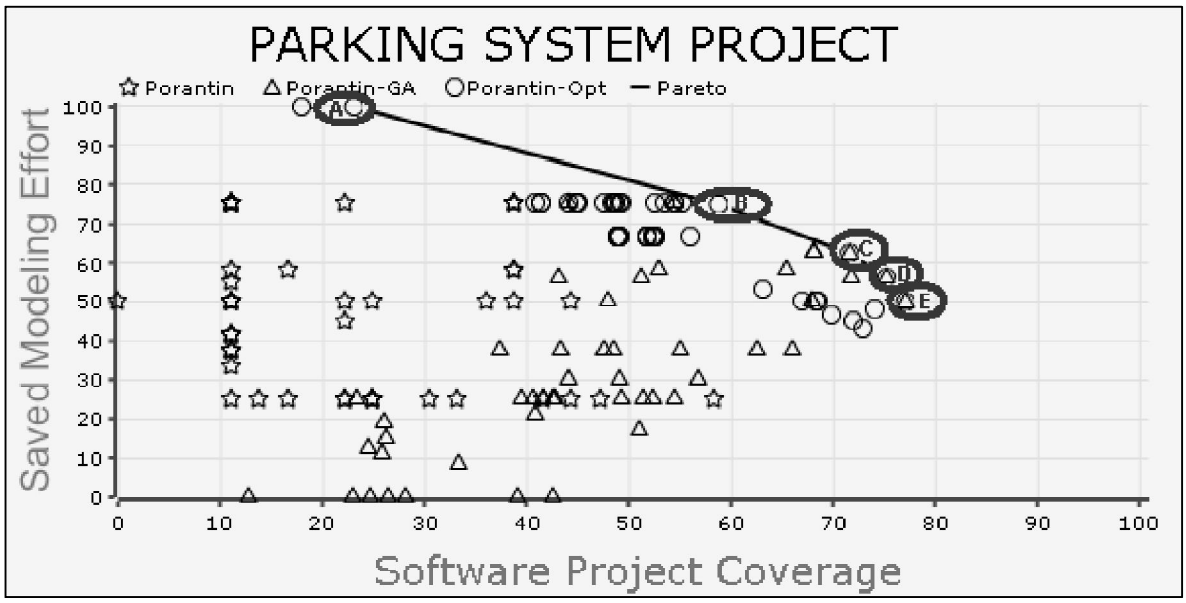


Figura 26. Resultados para o Projeto *Parking System*.

Analisando os resultados dos dois critérios de seleção para o projeto de software chamado *Parking System*, descrito inicialmente em (DIAS-NETO e TRAVASSOS, 2009), observou-se que: no total, cinco pontos (soluções) compõem a fronteira de Pareto, 2

5 www.icomp.ufam.edu.br/maraka

gerados pela abordagem *Porantim-Opt* ("A" e "B"), 2 gerados por *Porantim-GA* ("C" e "D"), e 1 gerado tanto por *Porantim-OPT* como por *Porantim-GA* ("E"). Estas informações podem ser observadas graficamente na Figura 26. Os resultados para o segundo projeto podem ser encontradas em GRANDE et al. (2012).

Isto sugere, como ponto inicial, que *Porantim-GA* poderia contribuir para a geração de novas soluções de compromisso que não foram geradas pelas versões anteriores. Além disso, outras soluções próximas à curva de Pareto foram geradas usando *Porantim-GA* (analisando os triângulos na Figura 26), representando soluções alternativas que podem ser utilizadas em um projeto de software, uma vez que, por vezes, a melhor opção para um projeto de software não pode ser selecionada para aspectos não técnicos (por exemplo: alto custo para adquirir a licença de uma ferramenta). *Porantim-GA* não se encontra integrado no framework STSP, serviu apenas como prova de conceito do trabalho.

3.5. Framework de Apoio a STSP

Esta seção irá detalhar o framework desenvolvido para apoiar a instanciação de técnicas para seleção de tecnologias de software (framework STSP). Será apresentada sua arquitetura extensível, como ele se integra com outros frameworks, principais configurações necessárias para utilizar o framework STSP, os passos necessários para criar um novo cenário de otimização, planejamento e execução de experimentos e suas formas de representação dos resultados.

3.5.1. Arquitetura do Framework

O framework STSP foi implementado em Java JEE 1.6 utilizando o padrão de projeto MVC e usa o framework Spring 2.5 para a camada de negócios, o framework JSF 1.2 para camada de apresentação e *Hibernate* 3 para a camada de dados. Como sistema de gerenciamento de banco de dados, foi utilizado o MySQL 5.1 e como servidor *web* o *Tomcat* 6.

Foi utilizado como ambiente de desenvolvimento o *Eclipse* IDE versão *Juno*. O código-fonte foi organizado como sugerido por GONÇALVES (2008), dividindo a aplicação em dois projetos: um para a camada de modelo de dados, no qual é realizado o mapeamento objeto-relacional, e outro para as camadas de apresentação e negócio, no qual contém as páginas web e as classes controladoras.

A modelagem do banco de dados para o framework STSP se encontra no Apêndice A e foi dividida em duas partes para facilitar a visualização. No apêndice A.1 encontra-se a modelagem das tabelas para o problema de seleção de tecnologia em si, enquanto no

apêndice A.2 possui a modelagem para a configuração de experimentos e como eles são organizados.

As principais tabelas para a modelagem do STSP são: *Cenários*, *Atributos*, *Tecnologias* e *Projetos*. Todas as demais configurações envolvem essas tabelas.

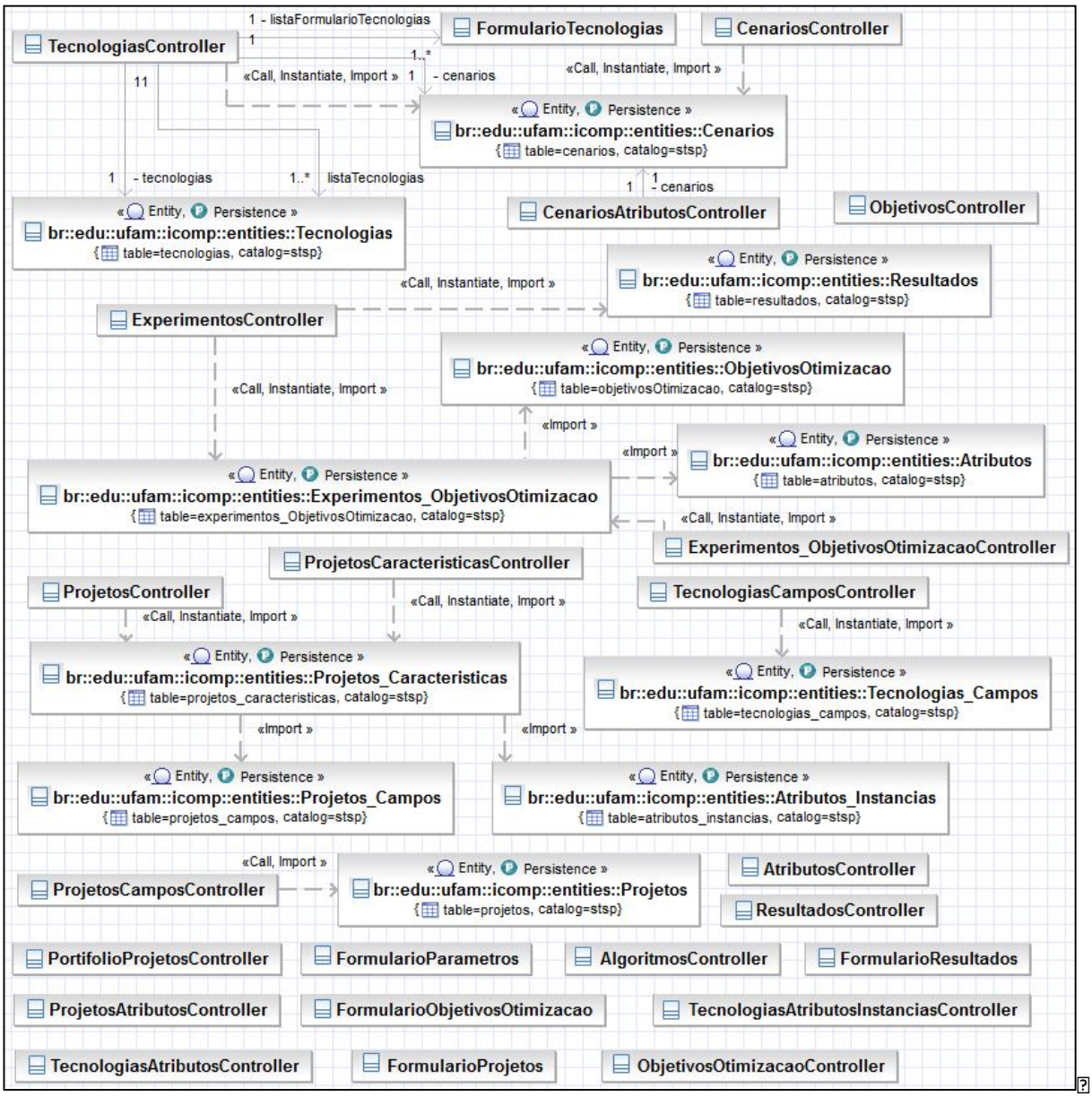


Figura 27. Diagrama de classes para o framework STSP.

Um *Cenário* representa uma subárea que se pretende otimizar e é definido por seus *Atributos*. Os *Atributos* representam as características utilizadas para compor um *Cenário* (tabela *Cenarios_Atributos*). Identificá-los é umas das principais etapas do trabalho, pois eles serão usados em várias etapas do processo de seleção. Essa configuração é realizada uma única vez e servirá de base para a configuração das *Tecnologias* e *Projetos*.

Tecnologias representam as técnicas disponíveis na literatura para um determinado cenário. Elas possuem campos (tabela *Tecnologias_Campos*) que incluem obrigatoriamente os atributos do Cenário (tabela *Tecnologias_Atributos*) e outros que se desejem incluir com conteúdo meramente informativo.

A mesma ideia é aplicada para os Projetos vinculados a um cenário. Estes possuem campos que os caracterizam (tabela *Projetos_Campos*) e que contém obrigatoriamente os atributos do cenário em questão (tabela *Projetos_Atributos*) e/ou outros com conteúdo informativo. A razão de se possuir uma tabela de atributos para *Projetos* e outra para *Tecnologias* é o fato de que o mesmo atributo pode ter uma cardinalidade ou valoração para tecnologias e outra diferente para projetos, ou seja, um atributo pode receber um único valor para uma técnica e múltiplos para um projeto ou vice-versa (tabela *Objetivos*).

Quanto à modelagem de experimentos, as principais tabelas são: Experimentos e Resultados. Um Experimento é composto por um ou mais projetos (tabela *Portifolio_Projetos*), meta-heurísticas (tabela Algoritmos) e por objetivos de Otimização (tabela *ObjetivosOtimizacao*). Após executar um experimento, os resultados são salvos (tabela Resultados) e vinculados um conjunto de soluções (tabela *Solucoes*). Cada solução é formada por uma combinação de técnicas (tabela *Combinacoes*).

A Figura 27 exibe o diagrama contendo as principais classes do framework STSP contendo suas dependências, associações e heranças.

3.5.2. Integração com Frameworks externos

Com o framework STSP é possível utilizar alguns frameworks Java de Meta-heurísticas disponíveis na literatura técnica. A Figura 28 mostra o framework STSP representado pela classe *Client* interagindo com outros frameworks através da classe *FrameworkFactory*. Esta classe segue o padrão de projeto *Factory Pattern*. Assim, para a integração de cada framework com o framework STSP precisamos criar uma classe que implementa a interface de *IFramework*. Essa *interface* contém métodos para a criação de uma nova solução e para calcular os valores dos objetivos que você deseja otimizar.

Na Figura 28 é apresentado um exemplo de integração de dois frameworks: JMetal (DURILLO e NEBRO, 2011) e Opt4J (LUKASIEWYCZ *et al.,* 2011), Para integrá-los, é necessário incluir as bibliotecas desses frameworks no framework STSP e criar uma classe que implementa a interface (*JMetalSTSP* e *Opt4JSTSP*, para os frameworks JMetal e Opt4J, respectivamente). Essas classes estendem as classes que representam os problemas em cada framework (classes *Problem* e *Optimizer Task* para os frameworks JMetal e Opt4J, respectivamente). Essas classes têm métodos para criar uma nova solução e calcular os valores de seus objetivos de otimização. A assinatura destes

métodos deve ser incluída na *interface IFramework*. No exemplo da Figura 28, foram incluídas as assinaturas dos métodos *create* e *evaluate* para o framework JMetal.

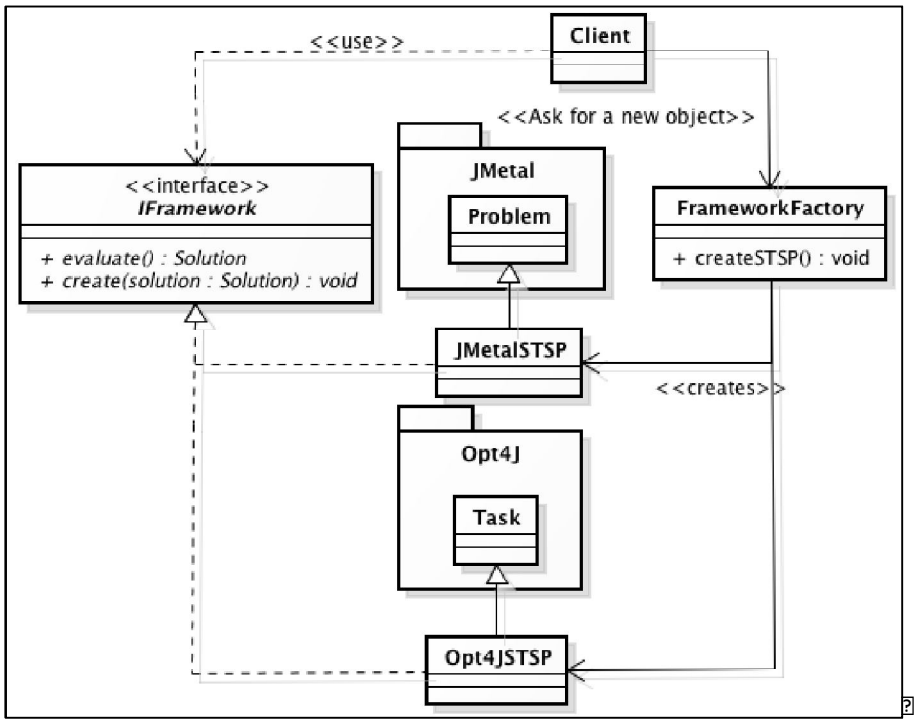


Figura 28. Arquitetura de Integração do Framework STSP com frameworks de meta-heurísticas JMetal e Opt4J.

Para configurar uma nova meta-heurística para ser utilizada no framework STSP, é necessário efetuar alguns passos. Primeiro, é preciso incluir o nome do algoritmo que se deseja incluir no cadastro de algoritmos (Figura 29). O nome do algoritmo deve ser equivalente ao nome da classe em que o algoritmo será implementado.

Figura 29. Tela de cadastro de algoritmos do framework STSP.

Depois disso, é necessária a criação de um *Driver* para o algoritmo, conforme ilustrado na Figura 30.

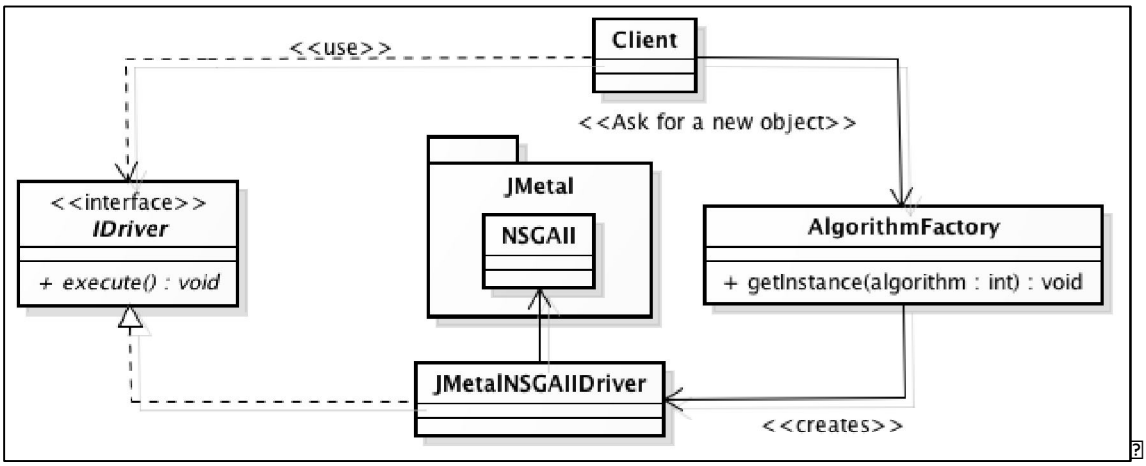


Figura 30. Integração do algoritmo NSGAII do JMetal com o Framework STSP.

3.5.3. Configurando Novos Cenários

O Framework STSP fornece recursos para configurar diferentes cenários para a seleção tecnologias de software (Figura 31). Essa fase é baseada na configuração de um novo cenário de engenharia de software no qual a seleção de tecnologias de software será realizada (por exemplo: técnicas de teste de software, ferramentas de modelagem, métodos de elicitação de requisitos).

Figura 31. Tela de Cadastro de Cenários no Framework STSP.

Para cada cenário, os atributos são definidos e associados como atributos de tecnologias de software (dimensão *DSour*) e atributos projetos de software (dimensão *DDest*), conforme a Figura 32. Esses atributos podem ser concebidos como monovalorados (apenas um valor é aceito por projeto de software / tecnologia - por

exemplo: um projeto de software) ou multivalorados (mais do que um valor pode ser aceito por projeto de software / tecnologia).

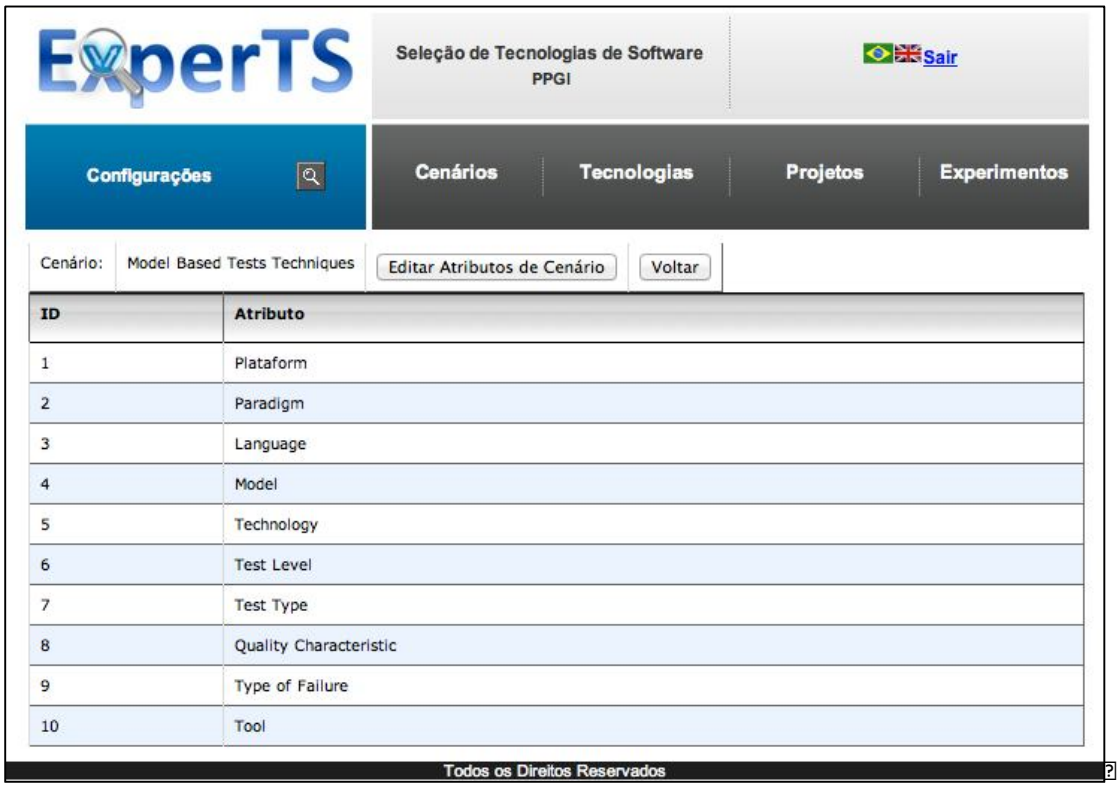


Figura 32. Tela de Atributos de Cenários.

3.5.3.1. Atributos de Cenários de Engenharia de Software

Os atributos de Cenários são os elementos responsáveis por identificar um cenário. Essa é uma das principais configurações, pois esses atributos serão usados para identificar as Tecnologias e Projetos, além de comporem os objetivos de otimização de um experimento. A configuração dos atributos é feita acessando a tela de atributos de Cenário (Figura 32) clicando no botão Editar Atributos de Cenários. Feito isso, abrirá uma tela na qual é possível selecionar os atributos disponíveis a partir do cadastro geral de atributos que sejam específicos para o cenário que está sendo cadastrado.

3.5.3.2. Atributos de Tecnologias de Software

Os Atributos de Tecnologia são os itens que identificam tecnologias de software. São os campos que apareceram no formulário de cadastro de tecnologia quando se desejar cadastrar uma nova técnica. Primeiramente, é necessário incluir um atributo de tecnologia (Figura 33), informando um nome e se este está associado a algum atributo do cenário. Caso o valor do campo Atributo seja nenhum, este atributo possuirá conteúdo meramente informativo e não será utilizado no processo de otimização.

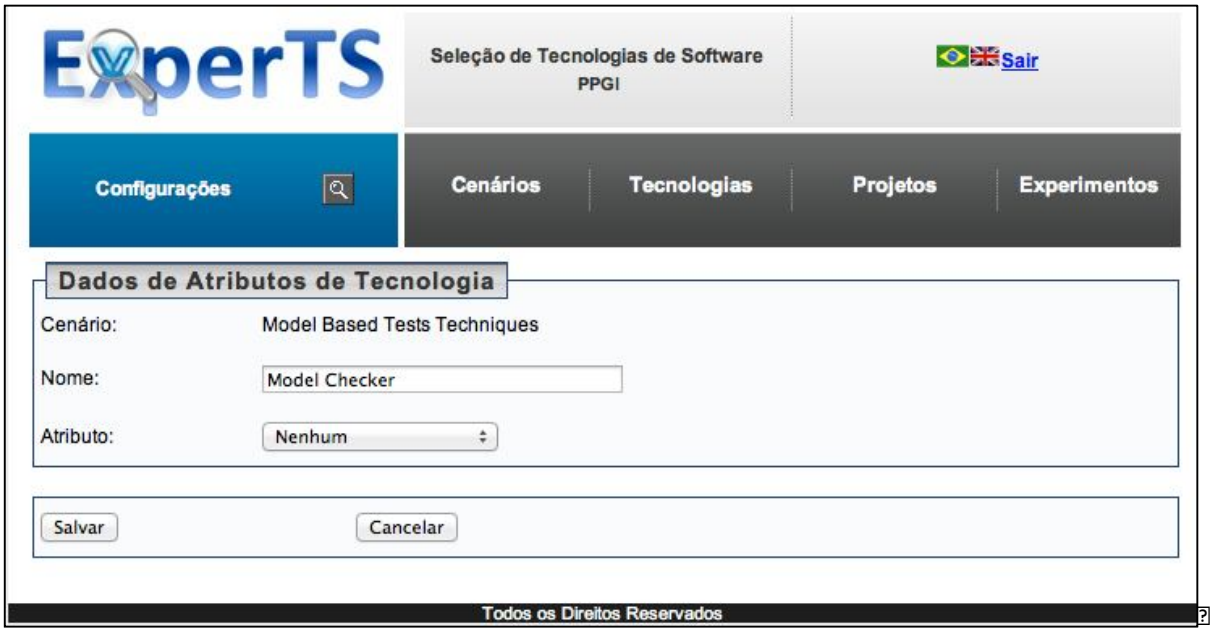


Figura 33. Tela de Novo Atributo de Tecnologia de Software.

Os atributos de tecnologias podem ser gerais ou específicos. Todos os atributos que são cadastrados pela ferramenta são atributos específicos para a Tecnologia referente a um cenário. Além disso, existem os Atributos Gerais, que sempre estarão presentes ao se cadastrar uma técnica. Como as técnicas são obtidas da literatura científica, foram definidos cinco atributos gerais de tecnologias: Nome (toda técnica precisa de um nome), Artigo (informando em qual artigo a técnica foi citada), Ano (ano de publicação do artigo), Autores (lista de autores do artigo) e Referência Completa (referência bibliográfica do artigo).

Após cadastrar os atributos de tecnologia, é necessário realizar a configuração da cardinalidade ou valoração dos atributos de tecnologia que estão associados aos atributos de cenário. Essa configuração é realizada através da tela de atributos de tecnologia (Figura 34). Nela são listados todos os atributos de tecnologia cadastrados, mostrando primeiramente os que estão associados a algum atributo de cenário e depois os com conteúdo meramente informativo.

Nos campos associados a algum atributo é possível realizar a configuração através da coluna Valoração. Nessa coluna é apresentado um campo no qual é possível selecionar se o atributo de tecnologia poderá receber um único valor ou vários.

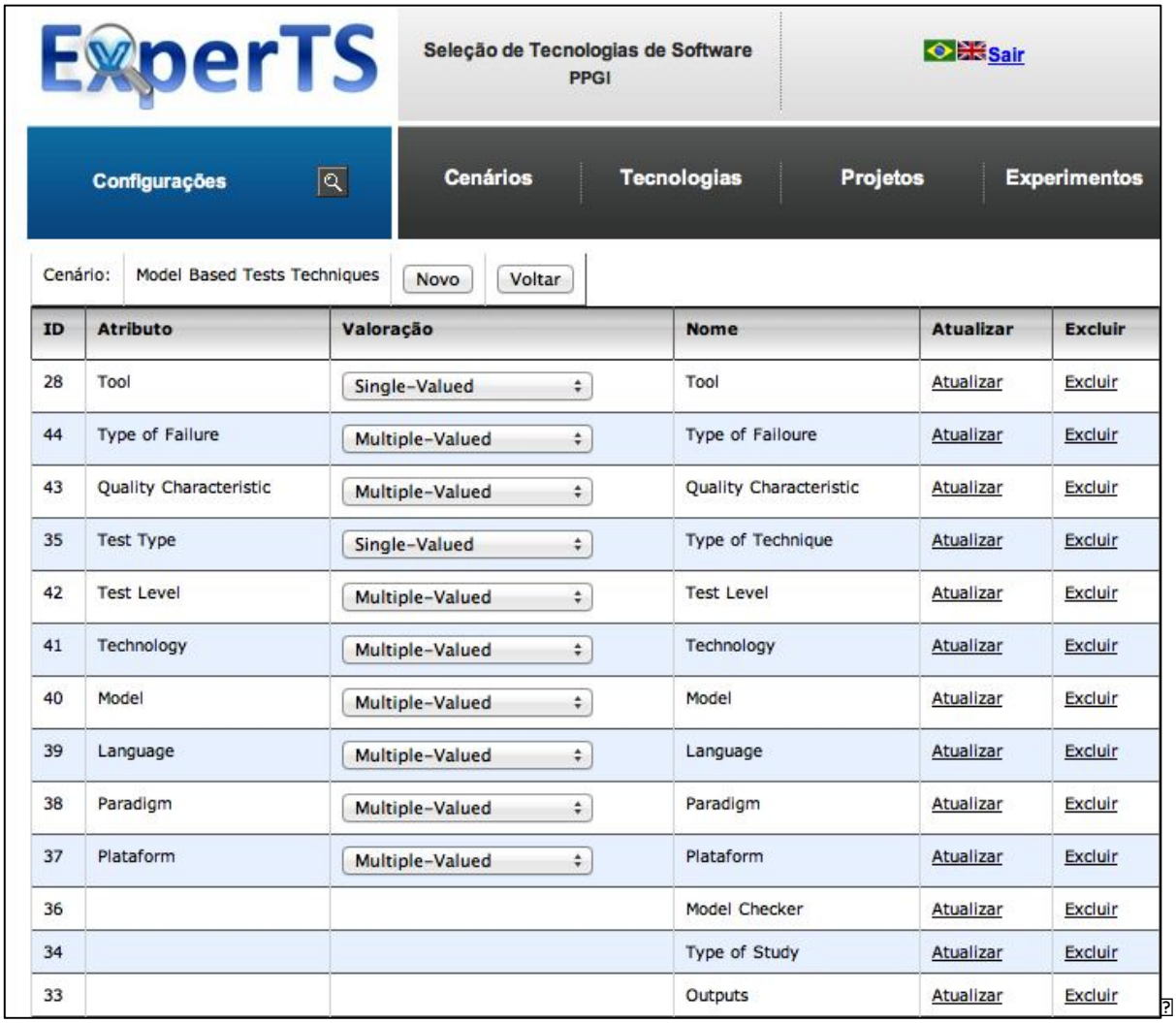


Figura 34. Tela de Atributos de Tecnologias de Software.

Essa configuração influenciará o cadastro das técnicas, pois o formulário de cadastro é gerado dinamicamente. Nele, se um atributo for configurado como monovalorado será exibido um campo de listagem contendo os valores (atributo *Type of Technique* na Figura 35) onde apenas uma opção pode ser selecionada. Caso o atributo seja configurado como multivalorado, serão exibidos todos os valores possíveis para o atributo, podendo ser selecionada mais de uma opção (atributo *Plataform* na Figura 35).

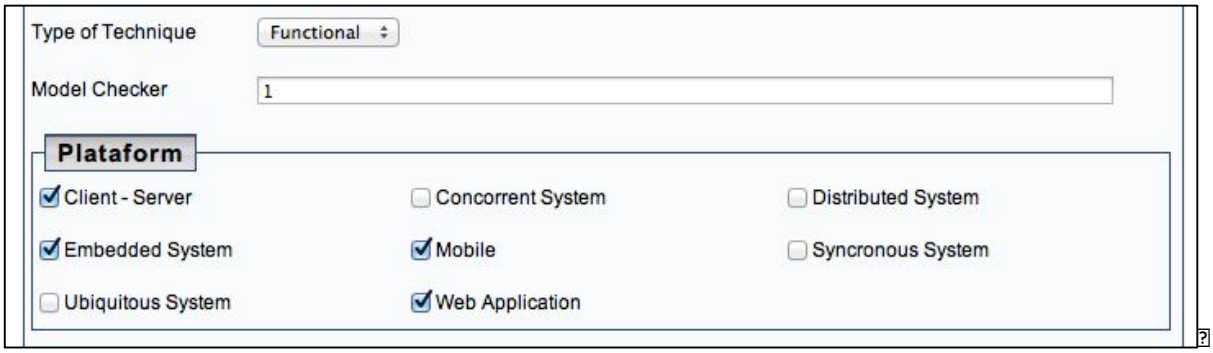


Figura 35. Formulário de Cadastro de Tecnologia para um Cenário.

3.5.3.3. Atributos de Projetos de Software

Similarmente aos Atributos de Tecnologias, os Atributos de Projetos são os itens que identificam projetos. São os campos que apareceram no formulário de cadastro de um novo projeto. Primeiramente, é necessário incluir um atributo de projeto informando um nome e se este está associado a algum atributo do cenário (Figura 36). Caso o valor do campo Atributo seja Nenhum, este atributo possuirá conteúdo meramente informativo e não será utilizado no processo de otimização.

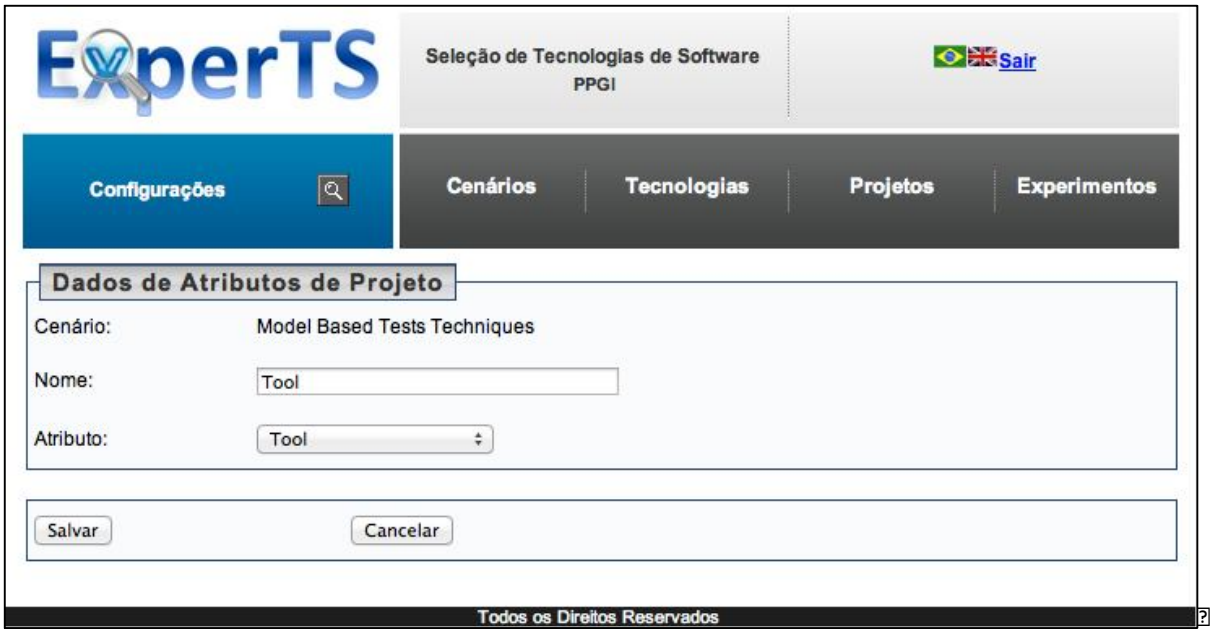


Figura 36. Tela de Novo Atributo de Projeto de Software.

Assim como ocorre em tecnologias de software, os atributos de projetos também podem ser gerais ou específicos. Todos os atributos que são cadastrados pela ferramenta são atributos específicos para projetos referentes a um cenário definido. Além disso, existem os Atributos Gerais que sempre estarão presentes ao se cadastrar um projeto. Foram definidos três atributos gerais de tecnologias: Nome/Título (todo projeto precisa de

um nome), Data de Início (informando em que momento o projeto é iniciado) e Duração (prazo definido para se concluí-lo). Estes atributos foram escolhidos devido à definição de projetos contida no guia de gerenciamento de projetos PMBOK (PMBOK, 2004), que diz um projeto é um esforço temporário, que possui um início e fim definidos.

Após cadastrar os atributos de projetos, é necessário realizar a configuração da cardinalidade ou valoração dos atributos de projetos que estão associados aos atributos de cenário. Essa configuração é realizada através da tela de atributos de projetos (Figura 37). Nela são listados todos os atributos de projetos cadastrados, mostrando primeiramente os que estão associados a algum atributo e depois os com conteúdo meramente informativo. Nos campos associados a algum atributo, é possível realizar a configuração através da coluna Valoração. Nessa coluna é apresentado um campo no qual é possível selecionar se o atributo de projetos poderá receber um único valor ou vários.

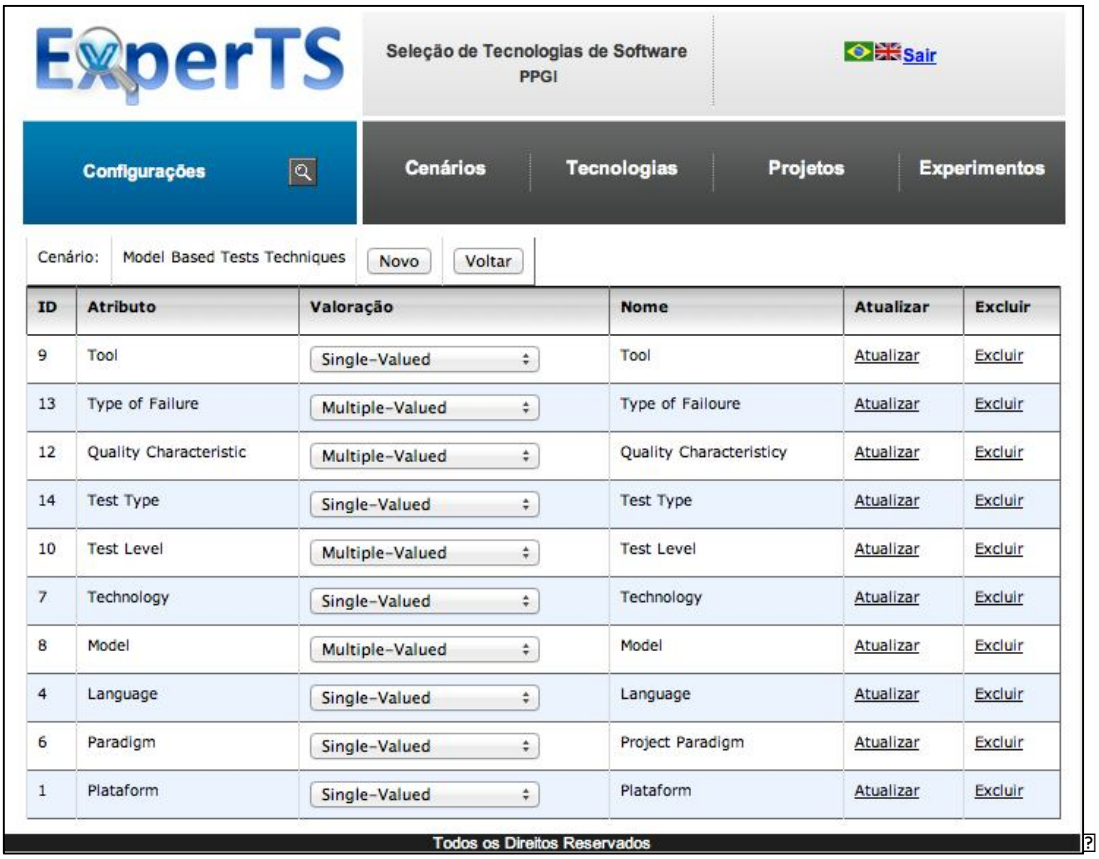


Figura 37. Tela de Atributos de Projetos de Software.

Essa configuração influenciará o cadastro dos projetos, pois o formulário de cadastro é gerado dinamicamente. Nele, se um atributo for configurado como monovalorado, será exibido um campo de listagem contendo os valores (atributo *Tool* na Figura 38) onde apenas uma opção pode ser selecionada. Caso o atributo seja configurado

como multivalorado, serão exibidos todos os valores para o atributo, podendo ser selecionada mais de uma opção (atributo *Test Level* na Figura 38).

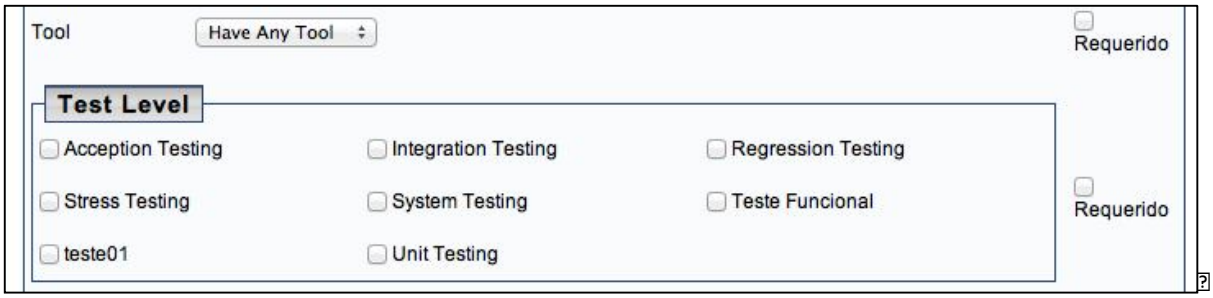


Figura 38. Tela de Cadastro de um Novo Projeto de Software.

Além disso, outro recurso disponível nessa tela é o campo *Requerido*. Através deste campo, é possível realizar um filtro das técnicas que serão utilizadas no Experimento. Se este campo for marcado, somente as tecnologias que possuírem ao menos um dos valores definidos para o mesmo atributo no projeto serão selecionadas no experimento.

3.5.3.4. Configuração de Experimento e Execução

Um experimento é definido pela configuração das 3 dimensões apresentadas na Secção 3.2.1: *Origem*, *Destino* e *Objetivo* (Figura 39).

A dimensão *Origem* indica se as soluções geradas compreendem uma única técnica ou combinação de técnicas (representado pelo campo combinação de técnicas). A combinação das técnicas indicará que uma solução poderá ser composta por mais de uma tecnologia de software (parâmetro *numberOfVariables*). Isso pode resultar em um aumento significativo (exponencial) do número de soluções possíveis para o cenário avaliado.

A dimensão *Destino* indica em quais projetos de software as tecnologias selecionadas serão aplicadas. Neste caso, é possível selecionar um único projeto ou um portfólio de projetos (campo Projetos). O número de projetos selecionados terá impacto sobre a forma de calcular os objetivos de otimização.

Finalmente, a dimensão *Objetivo* define as metas de otimização. Usando o framework STSP, os objetivos de otimização são definidos pela combinação de alguns atributos configurados para o cenário. Assim, é possível incluir vários objetivos quanto se deseje otimizar. Este cálculo é realizado por meio da análise da cobertura de características definidas no projeto e atendidas pelas tecnologias selecionadas para cada atributo.

Nesta fase, o usuário também pode escolher qual(is) framework(s) externo(s) e meta-heurísticas integradas ao framework STSP deseja executar para o cenário configurado. Além disso, os parâmetros necessários para usar estes algoritmos (estes parâmetros são definidos pelo framework original que implementa os algoritmos) devem

ser preenchidos, tais como: tamanho da população, número de gerações, a porcentagem de crossover e mutação, entre outros.

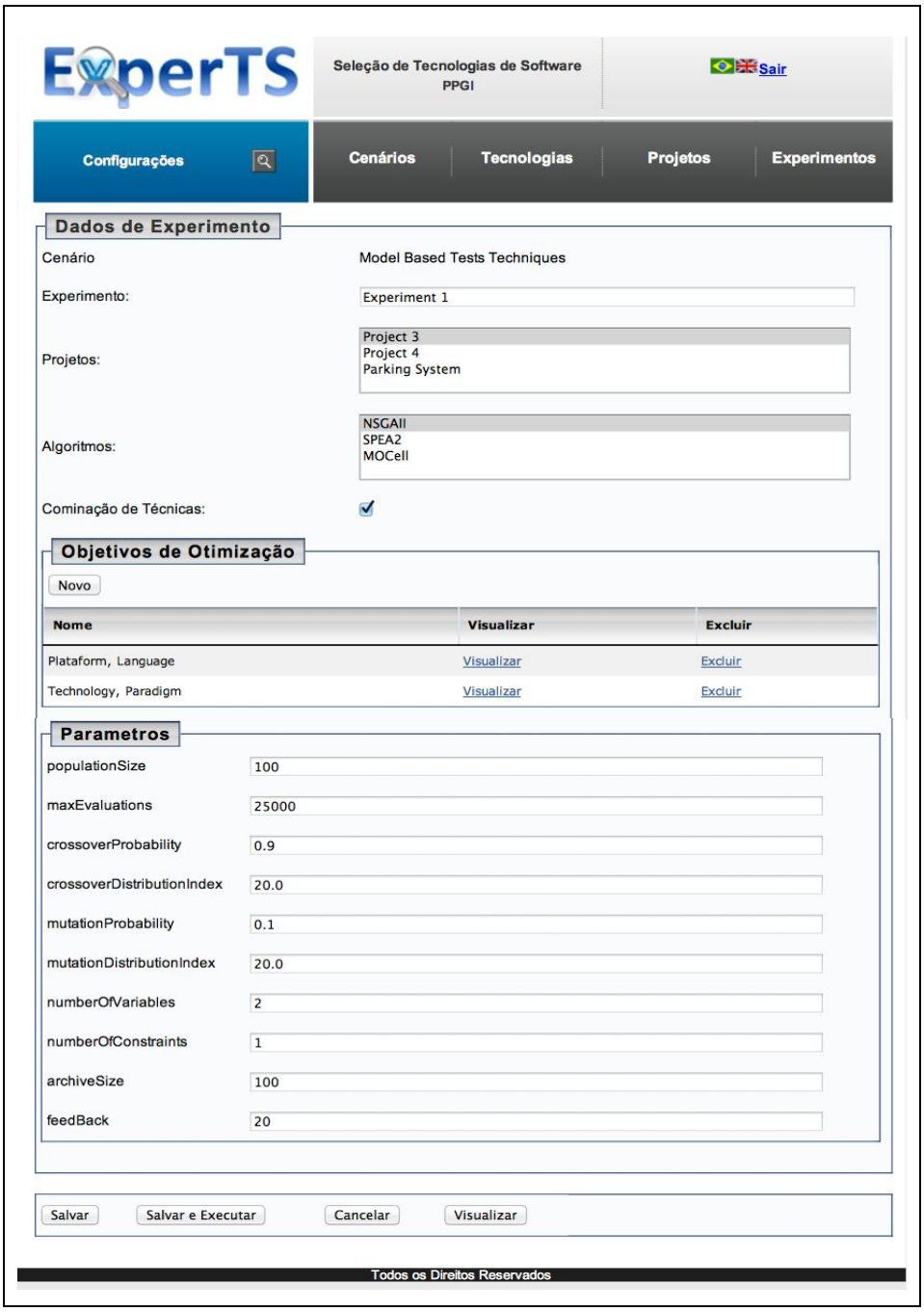


Figura 39. Tela de Configuração e Execução de Experimento.

3.5.3.5. Representação dos Resultados

Após a execução dos experimentos, os resultados são importados do framework externo que implementa a meta-heurística selecionada para o experimento e são armazenados na base de dados do framework STSP. Em seguida, são apresentados na tela do framework STSP os resultados obtidos. Nesta tela, o usuário pode ver um gráfico com as soluções geradas para cada algoritmo. O gráfico usado para representar os

resultados varia de acordo com o número de objetivos de otimização definidos para o cenário:

- Se um experimento possui apenas um objetivo (mono-objetivo), apenas uma lista ordenada das melhores soluções é exibida;
- Se tem dois objetivos, a lista de melhores soluções de compromisso (*trade-off*) e um gráfico de linha no plano XY é exibido, onde cada eixo representa um dos objetivos do problema (Figura 40);

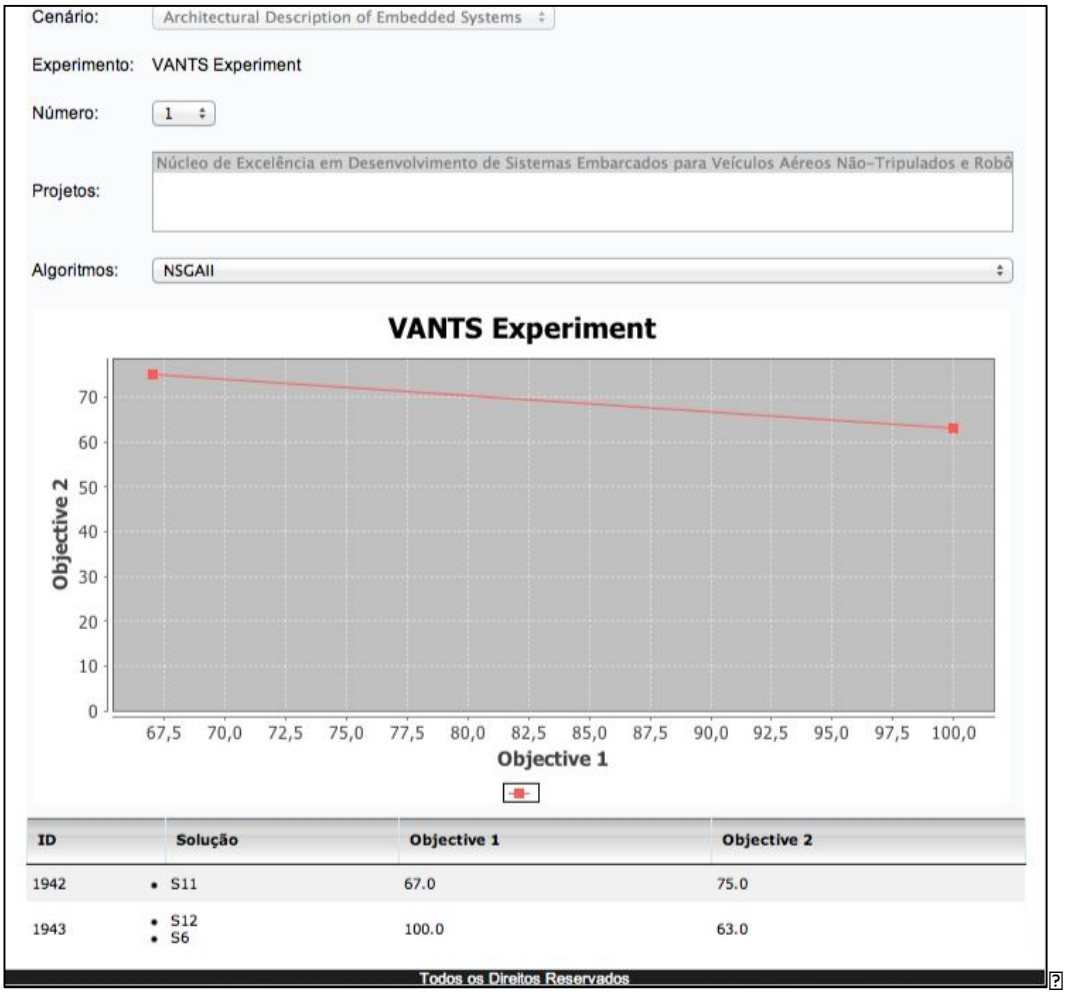
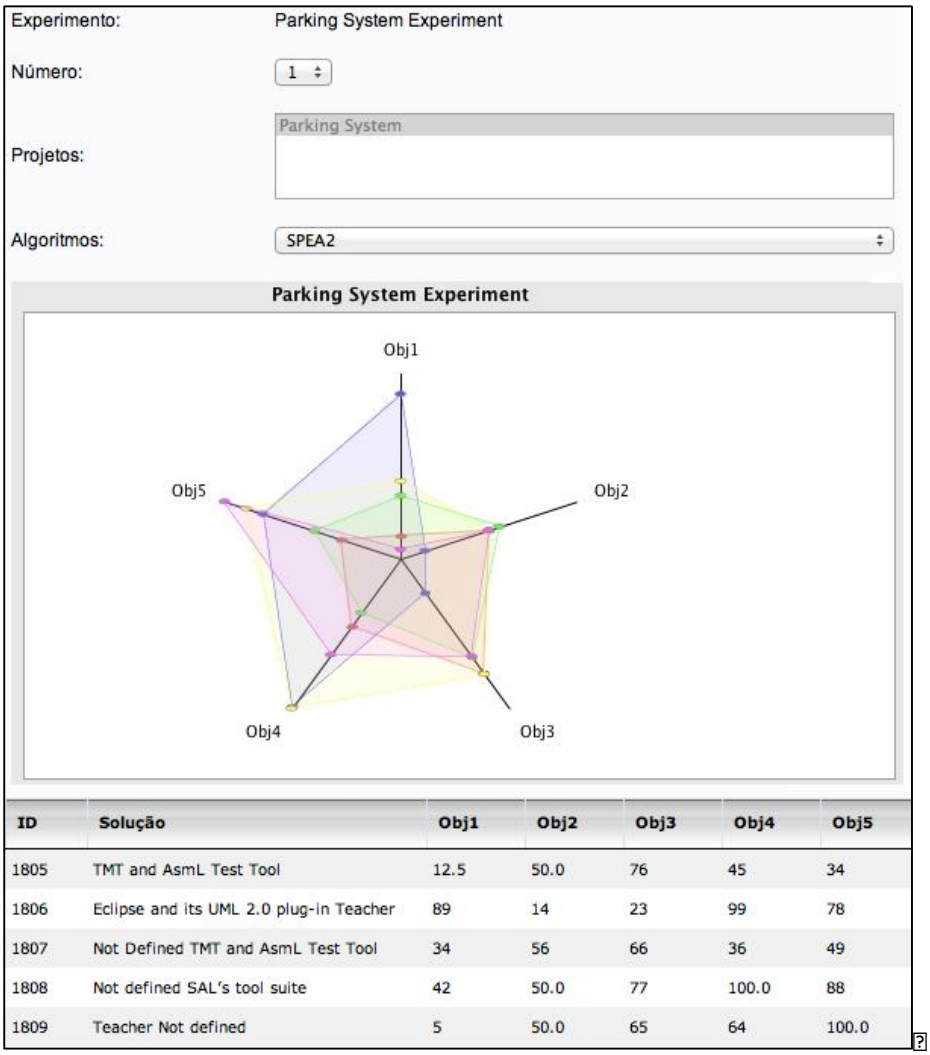


Figura 40. Telas de Gráficos de resultados com experimento com 2 objetivos.

- Se o experimento tem mais de dois objetivos, a lista das melhores soluções de compromisso (*trade-off*) são apresentadas e os resultados serão apresentados graficamente em um gráfico de radar onde cada atributo consiste em um vértice do gráfico (Figura 41).



ID	Solução	Obj1	Obj2	Obj3	Obj4	Obj5
1805	TMT and AsmL Test Tool	12.5	50.0	76	45	34
1806	Eclipse and its UML 2.0 plug-in Teacher	89	14	23	99	78
1807	Not Defined TMT and AsmL Test Tool	34	56	66	36	49
1808	Not defined SAL's tool suite	42	50.0	77	100.0	88
1809	Teacher Not defined	5	50.0	65	64	100.0

Figura 41. Telas de Gráficos de resultados com experimento com 3 objetivos.

Os gráficos da Figura 40 e, em especial o da Figura 41, introduz uma forma não encontrada na revisão bibliográfica feita para este trabalho de representar graficamente otimizações com 3 ou mais objetivos, o gráfico em forma de radar. Cada eixo representa um objetivo de otimização e as figuras coloridas representam as soluções geradas. Nele, não é necessário recorrer a projeções multidimensionais, bastando analisar as regiões sobrepostas de cada solução representada de uma cor, significando que a mesma domina ou é dominada para algum objetivo.

Além disso, na parte inferior da tela existe uma listagem contendo os valores dos objetivos de otimização para cada solução gerada e representada no gráfico. Os resultados são exibidos por algoritmo escolhido e pelo número de ordem em que foi executado. Também é possível comparar os resultados para vários algoritmos e várias execuções.

3.6. Considerações Finais

Neste capítulo foi apresentada a implementação do framework proposto, partindo da definição formal do problema de otimização e de sua modelagem. A partir disso, foram apresentados alguns resultados de experimentos para a instanciação do modelo em um cenário específico por meio da implementação de uma meta-heurística evolutiva *Porantim-GA* (GRANDE et al., 2012) que serviu como prova de conceito.

A próxima seção apresentará dois estudos de caso realizados para analisar a viabilidade de utilizar o framework desenvolvido para apoiar a seleção de tecnologias de software em dois cenários distintos de engenharia de software: seleção de técnicas de teste baseado em testes e seleção de técnicas de modelagem arquitetural para sistemas embarcados.

CAPÍTULO 4: ESTUDOS DE CASO DA APLICAÇÃO DO FRAMEWORK PROPOSTO

Esta seção apresenta dois estudos de caso que representam cenários reais da Engenharia de Software *relacionados à seleção de tecnologias de* software *onde o* framework *STSP foi aplicado. O principal objetivo deste estudo é avaliar a viabilidade do* framework *STSP para instanciar novos cenários de seleção de tecnologias de* software*.*

4.1. Introdução

Para mostrar a viabilidade do framework STSP e sua operação, foram realizados dois estudos de caso em cenários reais da engenharia de software relacionados ao problema da seleção de tecnologias de software, visto que o objetivo do framework é viabilizar experimentos utilizando dados reais. O primeiro cenário onde o framework STSP foi aplicado é o de seleção de técnicas de teste baseado em modelos e o segundo foi o da seleção de técnicas de modelagem arquitetural para sistemas embarcados. Assim, o STSP foi instanciado para ambos os problemas utilizando o framework STSP e então foram executados alguns experimentos usando diferentes meta-heurísticas fornecidas pelo framework proposto.

A justificativa para a seleção de ambos os cenários é: para o primeiro cenário já existia uma modelagem deste problema como um problema de otimização combinatória publicada em (DIAS-NETO et al., 2011) e (GRANDE et al., 2012). Assim, esta pesquisa resultou em uma evolução destes trabalhos anteriores. Além disso, já existe um repositório de técnicas de teste baseado em modelos disponível na literatura em (DIAS-NETO e TRAVASSOS, 2010) para ser utilizada no experimento. O segundo cenário foi escolhido devido à participação dos pesquisadores responsáveis pelo trabalho no INCT-SEC (Instituto Nacional de Ciência e Tecnologia – Sistemas Embarcados Críticos[6]). Neste grupo de pesquisa, uma das atribuições dos pesquisadores seria a identificação de técnicas de modelagem arquitetural para esta categoria de software e a criação de uma abordagem que apoiasse a seleção das diferentes técnicas que seriam mais adequadas para projetos de desenvolvimento de sistemas embarcados de acordo com as características específicas de cada projeto.

6 http://www.inct-sec.org/br/

A avaliação do framework será fundamentada na metodologia DESMET (KITCHENHAM et al., 1997) que possui nove diferentes formas de avaliação. O método escolhido para esse trabalho foi uma análise qualitativa através de estudos de caso, nos quais a metodologia é aplicada em cenários reais e os resultados são avaliados por meio de questionários/observações providos pelos pesquisadores. A utilização de uma análise quantitativa utilizando grupos de controles iria requerer um formalismo maior nos experimentos para futuras replicações e comparações, e este não foi o cenário adotado para avaliação deste trabalho. Portanto, a abordagem quantitativa foi descartada. Da mesma forma, a utilização de pesquisas de opiniões (*surveys*) também foi descartada, visto que seria necessário um grupo adequado de usuários de sistemas que pudessem avaliar as técnicas instanciadas por meio do framework provido, e este recurso não esteve disponível nesta pesquisa.

Assim, o objetivo principal com estas avaliações é observar a viabilidade de instanciação de técnicas reais provenientes da literatura técnica de apoio à seleção de tecnologias para diferentes cenários de engenharia de software. Além de técnicas reais, as avaliações se referem a outros elementos contendo dados reais extraídos da literatura técnica:

- Atributos definidos para caracterização das tecnologias/projetos de software;
- Dados das tecnologias identificadas em corpos de conhecimentos publicados em artigos científicos;
- Dados dos projetos de software utilizados nos experimentos, provenientes de projetos reais desenvolvidos na indústria e que os pesquisadores tiveram acesso.

4.2. Estudo de Caso 1: Técnicas de Teste Baseado em Modelos

4.2.1. Configuração do Cenário

Para o cenário do problema de seleção de técnicas de teste baseado em modelos (MBTTSP), foram definidos 10 atributos para caracterizar as técnicas de teste baseado em modelos (MBT) e projetos de software, obtidos a partir de (DIAS-NETO e TRAVASSOS, 2009). O objetivo neste problema é analisar se uma determinada técnica de MBT está atendendo aos requisitos de teste em um projeto de software. Assim, para cada atributo foi definida sua valorização (mono ou multivalorado) na caracterização das técnicas de MBT e na caracterização de um projeto de software, como descrito na Tabela 3. Um atributo monovalorado significa que um único valor pode ser instanciado para ele. Atributo multivalorado significa que mais do que um valor pode ser atribuído a ele.

Tabela 3. Caracterização dos Atributos de Tecnologia e Projeto para o MBTTSP.

[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

Após a configuração dos atributos das tecnologias e do projeto de software, é necessário criar o repositório de tecnologias (neste estudo, técnicas de MBT) que podem ser utilizadas em um projeto de software. Este passo é importante devido ao fato de que a proposta do framework STSP é o uso de dados reais nos experimentos usando meta-heurísticas para identificar as soluções mais adequadas para um projeto de software. Portanto, para este cenário foi usado como *benchmark* o repositório com 230 técnicas de MBT reais identificadas na literatura (DIAS-NETO e TRAVASSOS, 2010). Estas técnicas foram importadas para o framework STSP e foram utilizadas nos experimentos realizados neste estudo, como relatado na próxima subseção.

4.2.2. Arquitetura do Experimento

O experimento foi configurado com as seguintes dimensões: como dimensão *Origem*, foi definida a possibilidade de combinação de técnicas (*DSour2*), como dimensão *Destino*, seria utilizado um único projeto de software (*DDest1*) chamado *Parking System*, que foi caracterizado e utilizado nos estudos publicados em (DIAS-NETO et al., 2011) e (GRANDE et al., 2012), e como dimensão *Objetivo*, cinco objetivos de otimização foram definidos (*DObj2* - multiobjetivo). Os objetivos foram definidos a seguir e representam as funções de aptidão calculadas conforme a Figura 22 já mencionada:

- **Objetivo 1:** compreende o atributo de plataforma de software;

- **Objetivo 2:** compreende os atributos de paradigma de desenvolvimento e linguagem de programação;
- **Objetivo 3:** compreende os atributos modelo usado para a geração de teste e Tecnologia de Modelagem;
- **Objetivo 4:** compreende os atributos de nível de Teste e tipo de teste;
- **Objetivo 5:** compreende os atributos de característica qualidade, tipo de falha e ferramentas.

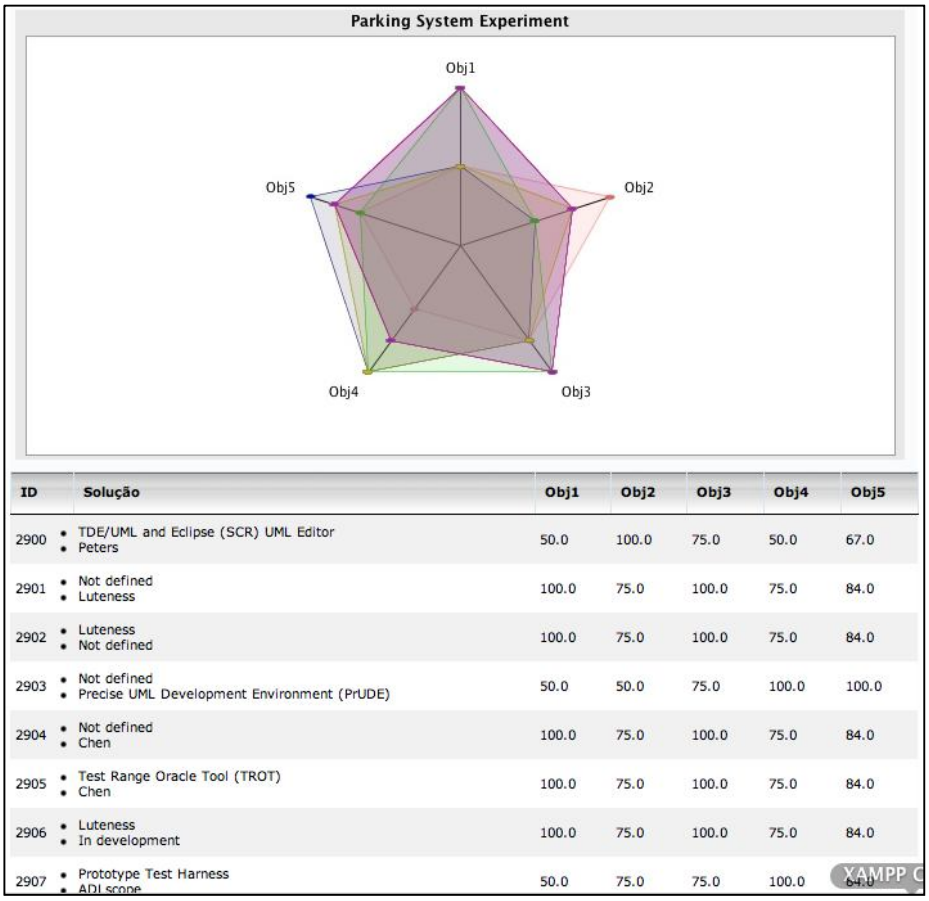
4.2.3. Execução do Experimento e Resultados.

Para a execução do experimento, foram utilizadas duas meta-heurísticas disponíveis no framework JMetal e que estão integradas ao framework STSP: NSGAII, SPEA2 e SPEA2. Os parâmetros utilizados para os algoritmos de otimização são aqueles utilizados como padrão pelo framework JMetal, pois estamos considerando as soluções geradas com combinação de 2 técnicas, o que equivale a uma quantidade de 40.000 possibilidades. Neste cenário, os parâmetros abaixo atendem a este requisito:

- Tamanho da População = 100;
- Máximo de Avaliações = 100;
- Probabilidade de Cruzamento *(Crossover)* = 0.9;
- *Probabilidade de Mutação (Mutation)* = 0.1;
- Índice de Distribuição = 20;
- Número de Variáveis (Técnicas em uma solução) = 2;
- Número de Restrições (Repetições de técnicas dentro de uma solução) = 1;
- Quantidade de Soluções não-dominadas arquivadas (*Archive Size*) = 20;
- Soluções Retornadas (*Feedback*) = 20.

Portanto, uma vez executado cada algoritmo, os resultados obtidos foram apresentados em um gráfico de radar, pois como dito anteriormente esta é a forma de representação quando se tem mais de dois objetivos (neste experimento foram definidos 5 objetivos para este experimento). O resultado está apresentado graficamente na Figura 42.

Utilizando as porcentagens obtidas pelo framework desenvolvido, o usuário pode visualizar todas as soluções não-dominadas. Além da representação gráfica, é possível visualizar a lista de soluções (combinação de técnicas de MBT) com os respectivos valores calculados para cada objetivo. A partir desta lista, os engenheiros de software terão mais informações para apoiar sua decisão sobre quais técnicas de MBT adotar no projeto de software.



ID	Solução	Obj1	Obj2	Obj3	Obj4	Obj5
2900	• TDE/UML and Eclipse (SCR) UML Editor • Peters	50.0	100.0	75.0	50.0	67.0
2901	• Not defined • Luteness	100.0	75.0	100.0	75.0	84.0
2902	• Luteness • Not defined	100.0	75.0	100.0	75.0	84.0
2903	• Not defined • Precise UML Development Environment (PrUDE)	50.0	50.0	75.0	100.0	100.0
2904	• Not defined • Chen	100.0	75.0	100.0	75.0	84.0
2905	• Test Range Oracle Tool (TROT) • Chen	100.0	75.0	100.0	75.0	84.0
2906	• Luteness • In development	100.0	75.0	100.0	75.0	84.0
2907	• Prototype Test Harness • ADL scope	50.0	75.0	75.0	100.0	64.0

Figura 42. Resultados do Experimento para o Projeto Parking System.

Na parte abaixo do gráfico da Figura 42 existe a listagem com os valores em porcentagem calculados para cada objetivo definido. Por esses valores é possível verificar que nenhuma das soluções domina as outras considerando todos os objetivos. Portanto, a decisão de qual solução escolher cabe ao engenheiro de software e sua análise será baseada em qual objetivo deseja priorizar mais em relação ao outros. Na próxima seção iremos analisar o conjunto de todas as soluções geradas para todas as execuções dos algoritmos selecionados.

4.2.4. Análise do Estudo de Caso 1

Para a execução do experimento, foram utilizadas três meta-heurísticas disponíveis no framework JMetal e integrado ao framework STSP: NSGAII, SPEA2 e MOCell.

Cada algoritmo foi executado 10 vezes e os resultados obtidos com as soluções não-dominadas foram apresentados em um gráfico de radar, porque cinco objetivos foram definidos para este experimento. Cada execução foi armazenada no banco de dados a ser consultado em um momento futuro. Os engenheiros de software pode reexecutar este experimento quantas vezes quiserem.

Usando o framework STSP, engenheiros de software podem escolher qual algoritmo (que também pode selecionar todos eles) e qual número de execução querem exibir as soluções geradas para comparar os resultados entre os algoritmos. Além da representação gráfica, os engenheiros de software podem ver a lista de soluções (combinação de técnicas MBT) com os respectivos valores calculados para cada objetivo. A partir dessa lista, os engenheiros de software terão mais informações para apoio de que decisão sobre quais técnicas MBT usar no projeto de software.

A fim de realizar a comparação dos algoritmos, os resultados de todas as 10 execuções foram unidos gerando um gráfico de radar para cada algoritmo (apenas com as soluções não-dominadas). Além disso, podemos também montar os resultados de todos os algoritmos no mesmo gráfico (Figura 43).

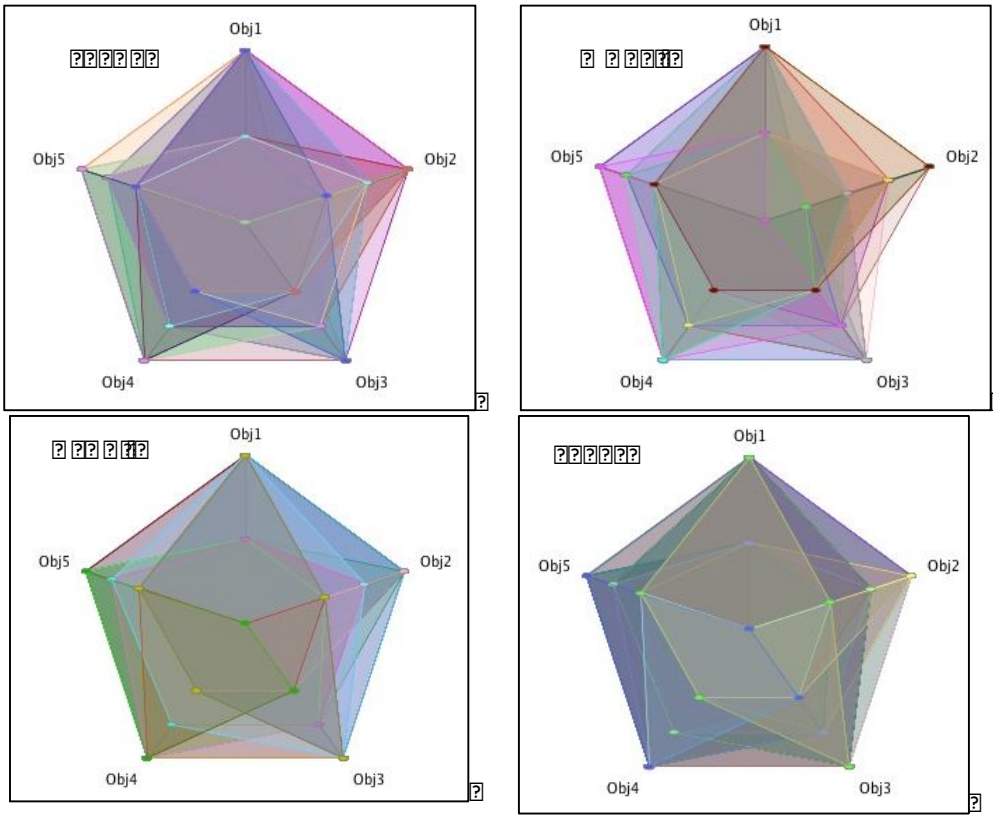


Figura 43. Comparativo dos gráficos de resultados dos algoritmos para o projeto *Parking System*.

Foram analisados os 30 melhores resultados obtidos após todas as 10 execuções para cada algoritmo (SPEA2, MOCell ou NSGAII) para fazer algumas observações sobre os gráficos gerados. Todos os três algoritmos geraram soluções de compromisso. No entanto, podemos analisar os gráficos priorizando cada objetivo e verificar quais as soluções tendem mais para um determinado objetivo. Assim, juntando os resultados

obtidos nos três algoritmos selecionados e classificando as 30 melhores soluções, chegamos à distribuição apresentada na Tabela 10 (analisando objetivo por objetivo).

Tabela 4. Distribuição das 30 melhores soluções geradas por objetivo.

Algoritmos	Obj1	Obj2	Obj3	Obj4	Obj5
Todos	2	2	2	3	1
SPEA2 e NSGAII	1	1	1	1	1
SPEA2 e MOCell	1	1	1	1	1
MOCell e NSGAII	1	1	1	0	1
SPEA2	8	7	6	8	6
NSGAII	7	9	12	14	11
MOCell	14	13	11	7	12

Priorizando o objetivo *Obj1*, observou-se que duas soluções foram geradas por todos os algoritmos e também observamos os resultados que tendem para este objetivo foram as soluções geradas pelo algoritmo MOCell, contendo 14 soluções. Priorizando o objetivo *Obj2*, observamos que 2 soluções foram geradas por todos os algoritmos e os resultados que tendem para este objetivo foram também geradas pelos algoritmos MOCell, contendo 13 soluções. Priorizando o objetivo *Obj3*, observamos que 2 soluções foram geradas por todos os algoritmos e os resultados que tendem para este objetivo foram também geradas pelos algoritmos MOCell, contendo 11 soluções. Priorizando o objetivo Obj4, observamos que 3 soluções foram geradas por todos os algoritmos e os resultados que tendem para este objetivo foram também geradas pelos algoritmos NSGA2, contendo 14 soluções. Finalmente, priorizando o objetivo Obj5, analisamos que 12 soluções foram geradas por todos os algoritmos e os resultados que tendem para este objetivo foram as soluções geradas pelo algoritmo MOCell, contendo 12 soluções.

Na Tabela 5, apresentamos os resultados de todas as rodadas e a união de todas as execuções (última coluna), a fim de realizar a comparação entre os algoritmos para várias métricas: soluções não dominadas (ND), as soluções não-dominadas distintas (DNDS – aquelas soluções em que as repetições são excluídas), o tempo de execução (TIME), o número de soluções repetidas (RS), e variedade de soluções combinadas geradas / execução (AVG COMBS).

Comparando os resultados obtidos nos experimentos (mostrado na Tabela 5), em primeiro lugar, verificou-se que o algoritmo NSGAII obteve o maior número de soluções não-dominadas, bem como soluções não-dominadas distintas. Seguidos, respectivamente, por SPEA2 e MOCell. No entanto, considerando-se outras perspectivas, descobrimos que

a abordagem mais rápida foi MOCell. Se olharmos para a quantidade de soluções repetidas, SPEA2 tive um desempenho melhor do que NSGAII e MOCell. Se analisarmos a variedade de soluções combinadas geradas / execução, as soluções geradas pelo NSGAII contém, em média, mais técnicas do que os outros dois algoritmos.

Tabela 5. Métricas do experimento para cada algoritmo em todas as rodadas.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

A escolha dos objetivos de otimização, devido à limitação do framework STSP em replicar objetivos definidos em abordagens anteriores (*Poratim-Opt* e *Porantim-GA*), não favoreceu uma avaliação comparativa entre os resultados obtidos pelo framework STSP e os dessas abordagens. Por exemplo, o objetivo esforço salvo possui uma fórmula específica para o cenário de técnicas MBT, que não poderia ser generalizada para o cenário de técnicas de Modelagem Arquitetural para Sistemas Embarcados e, portanto incluídas no framework STSP. No entanto, o framework foi instanciado para o cenário de técnicas MBT e realizaram-se experimentos com um projeto de software real da indústria. Após isso, os resultados foram coletados e podem ser avaliados por um engenheiro de software, situação que atende ao objetivo principal framework proposto.

4.3. Estudo de Caso 2: Técnicas de Modelagem Arquitetural para Sistemas Embarcados

Sistemas Embarcados vêm aumentando seu papel no cotidiano das pessoas, ainda que os mesmos não sejam percebidos, como celulares, carros, utensílios eletrodomésticos, etc., além de setores de aviação, médica e de energia (OSSADA e MARTINS, 2010). O mesmo autor conceitua Sistemas Embarcados, também conhecidos

como Sistemas Embutidos ou Sistemas Dedicados, como sistemas com a finalidade de controlar um ambiente ou dispositivo físico, diferindo-se em diversos aspectos de um computador para propósitos gerais.

Além disso, o desenvolvimento de um sistema embarcado envolve algumas características limitadoras, tais como manter o baixo consumo da potência sem o comprometimento do sistema, limitação quanto ao tamanho da memória, tamanho reduzido de circuitos integrados, projetos de hardware com baixo custo, necessidades de segurança, requisitos de tempo real, alta confiabilidade e disponibilidade, dentre outras (CANCIAN et al., 2007).

Sistemas embarcados possuem uma subcategoria denominada de sistemas embarcados críticos. Sistemas críticos quanto à missão são caracterizados pela alta dependabilidade. Isto significa que muitos operam em condições extremas, tais como funcionamento ininterrupto, longo tempo de missão, exposição a fatores ambientais rigorosos, etc. Um defeito nestes sistemas pode comprometer a vida e/ou o meio-ambiente, ou provocar prejuízos econômicos enormes. O funcionamento destes sistemas é dito seguro, isto é, uma falha não deve provocar efeitos catastróficos. O sistema, se falhar e não puder se recuperar, deve atingir um estado seguro (SIQUEIRA et al., 2006). Neste contexto, os pesquisadores responsáveis por este trabalho participam do Instituto Nacional de Ciência e Tecnologia – Sistemas Embarcados Críticos (INCT-SEC), o que contribuiu para a definição deste cenário de Engenharia de Software como estudo de caso para esta pesquisa.

O estudo de caso 2 tem como propósito a seleção de tecnologias para o cenário de técnicas de modelagem arquitetural de sistemas embarcados. Com a finalidade de formar um corpo de conhecimento sobre esta área contendo técnicas reais, foram utilizadas as abordagens da revisão sistemática realizada por GUESSI et al. (2012).

4.3.1. Configuração do Cenário

As técnicas e os respectivos atributos do cenário foram extraídos de uma revisão sistemática publicada em (GUESSI et al., 2012). Ao total, nesse estudo foram identificadas 24 técnicas que formaram o corpo de conhecimento para esse cenário (Tabela 6).

Baseado ainda em (GUESSI et al., 2012) foram identificados 6 atributos para o cenário:

a) Domínio (*Domain*): este atributo se refere à área de sistemas embarcados na qual foi relatado que a técnica foi aplicada. Os autores definem ainda os possíveis valores para este atributo: *Automation Technology*, *Automotive*, *Avionics*, *Energy Technology*,

Medical Technology, *Telecomunications*. Apesar de ser registrado para cada técnica o que foi identificado no artigo que a apresentou, é possível que essa técnica possa ser aplicada em outros domínios. O framework proposto poderia contribuir inclusive neste sentido, mantendo atualizado o corpo de conhecimento sobre as tecnologias de software a partir da sua aplicação em diversos projetos;

Tabela 6. Técnicas de modelagem arquitetural para sistemas embarcados.

Id	Ano	Referência
S1	2011	AUTOSAR. AUTOSAR (AUTomotive Open System ARchitecture). [On-line], World Wide Web. Available in http://www.autosar.org/ (Last accessed 06/19/2011).
S2	2009	Bak, S. et al. The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety. RTAS, San Francisco, USA, 99-107.
S3	2002	Bechini, A. and Prete, C.A. Performance-steered design of software architectures for embedded multicore systems. Software - Practice and Experience. 32, 12 (2002), 1155-1173.
S4	2001	Bernardo, M. et al. Detecting architectural mismatches in process algebraic descriptions of software systems. WICSA, Amsterdam, The Netherlands, 77-86.
S5	2008	Bessam, A. and Kimour, M.T. Multi-view Description of Real-Time Systems Architecture. International Journal of Electrical and Computer Engineering. 3, 11 (2008), 680-687.
S6	2010	Biehl, M. and Torngren, M. An Executable Design Decision Representation Using Model Transformations.SEAA, Lille, France, 131-134.
S7	2009	Borde, E. et al. Mode-based reconfiguration of critical software component architectures. DATE, Dresden, Germany, 1160-1165.
S8	2008	Boulanger, J.-L. and Dao, V.Q. Requirements engineering in a model-based methodology for embedded automotive software. RIVF, Ho Chi Minh, Vietnam, 263-268.
S9	2000	Champeau, J. et al. Object oriented and formal methods for AUV development. OCEANS, Providence, USA, 73-78.
S10	2000	Ciarletta, L. and Dima, A. A conceptual model for pervasive computing. ICPP Workshops, Toronto, Canada, 9-15.
S11	1998	Eixelsberger, W. and Gall, H. Describing software architectures by system structure and properties. Computer Software and Applications Conference. COMPSAC, Vienna, Austria, 106-111.
S12	2011	ESDS Reference Architecture Working Group NASA ESDS Reference Architecture v 1.0. [On-line], World Wide Web. Available in (Last accessed 11/25/2011) http://www.esdswg.org/spg/spgfolder/reference-architecture/ESDSRefArch-v1.0.pdf
S13	2010	Gilles, O. and Hugues, J. A MDE-Based Optimization Process for Real-Time Systems. ISORC, Carmona, Spain, 50-57.
S14	2009	Khalgui, M. et al. Model-checking for the functional safety of Control Component-based heterogeneous embedded systems. ETFA, Mallorca, Spain, 1-10.
S15	2002	Kru¨ger, I. et al. From scenarios to hierarchical broadcasting software architectures using UML-RT. International Journal of Software Engineering and Knowledge Engineering. 12, 2 (2002), 155-174.
S16	2010	Langsweirdt, D. et al. Architecture-Driven Development of Embedded Systems with ACOL. ISORCW, Carmona, Spain, 138-144.
S17	2000	Ledeczi, A. et al. Synthesis of self-adaptive software. AERO, Big Sky, USA, 501-507.
S18	2002	McDuffie, J.H. Using the architecture description language MetaH for designing and prototyping an embedded reconfigurable sliding mode flight controller. DASC, Irvine, USA, 8B1-1 - 8B1-17 vol.2.
S19	2010	Perseil, I. and Pautet, L. Formal methods integration in software engineering. Innovations in Systems and Software Engineering. 6, 1 (2010), 5-11.
S20	2006	Radjenovic, A. and Paige, R. Architecture description languages for high-integrity real-time systems. IEEE Software. 23, 2 (2006), 71-79.
S21	2009	Roo, A. et al. An architectural style for optimizing system qualities in adaptive embedded systems using Multi-Objective Optimization. WICSA/ECSA, Cambridge, UK, 349-352.
S22	2003	Tan, T.K. et al. Software Architectural Transformations: A New Approach to Low Energy Embedded Software. DATE, Munich, Germany, 1-6.
S23	2009	Visnevski, N.A. and Castillo-Effen, M. A UAS capability description framework: Reactive, adaptive, and cognitive capabilities in robotics. AERO, New York, USA, 1-7.
S24	2008	Ye, J. et al. A method to generate embedded real-time system test suites based on software architecture specifications. ICYCS, Hunan, China, 2325-2329.

b) **Atributos de Qualidade (*Quality Attributes*):** este atributo se refere aos atributos de qualidade (requisitos não-funcionais) e pode conter os seguintes valores: [illegible]

c) **Linguagem Arquitetural (*Architectural Language*):** este atributo se refere à linguagem que a abordagem utiliza para realizar a modelagem da arquitetura. Foram identificadas em (GUESSI et al., 2012) as seguintes linguagens:
   (1) *AADL(Architecture Analysis and Design Language);*
   (2) *ACME (An Architecture Description Interchange Language);*
   (3) *ACOL (A Model Annotation Language);*
   (4) *ASDL (Architecture Structure Description Language);*
   (5) *CCM (Common Object Request Broker Architecture Component Model);*
   (6) *Custom Informal Approaches.*
   (7) *DRTSADL(Distributed/Embedded Real-Time System ADL);*
   (8) *EAST-ADL (Electronic Architecture and* Software *Tools ADL);*
   (9) *FSM (Finite State Machines);*
   (10) *MARTE (UML Profile for Modeling and Analysis of Real-time and Embedded Systems);*
   (11) *MetaH (Avionics Architecture Description Language);*
   (12) *NCES (Net Condition/Event Systems)*
   (13) *PADL (Process Algebraic Architectural Description Language)*
   (14) *REAL (Requirement Enforcement Analysis Language);*
   (15) *SAG (*Software *Architecture Graph);*
   (16) *SDL (Specification and Description Language);*
   (17) *SysML (Systems Modeling Language);*
   (18) *UML (Unified Modeling Language);*
   (19) *UML-RT (UML Profile for Real-time Systems);*

d) **Tipo de Arquitetura (*Type of Architectural*):** este atributo se refere ao nível de formalismo que se necessita para a modelagem arquitetural. Este pode ser de três tipos diferentes:
   (1) Software *Architecture*: abordagem formal, onde geralmente possuem geração automática de código e recursos para verificação;
   (2) *Reference Architecture*: abordagens semiformais ou uma mistura de abordagem semiformais e informais;

(3) *Reference Model*: usadas apenas para abordagens informais.

e) Linguagem de programação (*Programming Language*): este atributo se refere à linguagem de programação utilizada na geração automática de código para alguma abordagem;

f) Ferramenta de Apoio (*Tool*): este atributo se refere à existência de alguma ferramenta que implemente a abordagem.

Além desses atributos, também incluímos outro atributo e classificamos as técnicas para sistemas críticos quando possuírem os atributos de qualidade *safety* ou *fault-tolerance*. Apesar de parecer uma replicação de informações, a existência desse atributo é necessária e em trabalhos futuros esse critério pode ser mudado:

g) Sistema Crítico (*Critical System*): este atributo se refere à criticidade do sistema para o qual a técnica é aplicada.

Por fim, para completar a configuração do cenário precisou-se definir a valoração para os atributos. Isso está definido na Tabela 7.

Tabela 7. Caracterização dos Atributos de Tecnologia e Projeto.

Projeto de Software		**Técnicas ADES**	
Atributos	**Valoração**	**Atributos**	**Valoração**
Domínio de Aplicação	Multi	Domínio onde pode ser aplicada.	Multi
Atributos de Qualidade requeridos no projeto	Multi	Atributos de Qualidade em que pode ser aplicada.	Multi
Linguagem Arquitetural usada s/conhecidas no projeto	Multi	Linguagem Arquitetural em que pode ser aplicada.	Multi
Tipo de Arquitetura de software	Mono	Tipo de Arquitetura em que pode ser aplicada	Mono
Linguagem de Programação usada no projeto	Multi	Linguagem de Programação em que pode ser aplicada.	Multi
Ferramentas desejadas/ conhecidas no projeto	Multi	Ferramentas utilizadas e que implementam a técnica	Mono
Se o projeto é considerado crítico	Mono	Se a técnica é aplicada em projetos de sistemas críticos.	Mono

O conjunto de valores para cada atributo pode aumentar conforme sejam identificados na literatura técnica outros valores usados pelas abordagens. Após definir os atributos usados para caracterização de uma técnica de modelagem arquitetural, foram instanciados os mesmos para cada uma das 24 técnicas identificadas. O resultado encontra-se na Tabela 8. Quando não foi possível identificar o valor de um atributo para uma determinada

técnica a partir do seu artigo de referência, foi colocado o valor “Não Definido” em sua caracterização.

Tabela 8. Atributos das Técnicas para Modelagem Arquitetural de Sistemas Embarcados.

Id	Atributo de Qualidade	Domínio	Linguagem Arquitetural	Tipo de Arquitetura	Linguagem de Programação	Ferramenta de Apoio
S1	• Interoperability • Maintainability • Performance • Safety	• Automotive	• UML • Custom	Reference Architecture	Não Definido	Não Definido
S2	• Fault-Tolerance • Safety	• Automation	• AADL • ACOL • REAL • Metah • FSM	Não Definido	VHDL	UPPAAL
S3	• Performance • Power • Consumption	• Energy	• UML	Não Definido	Não Definido	Não Definido
S4	• Interoperability	• Automation	• PADL	Software Architecture	Não Definido	Não Definido
S5	• Dependability	• Energy • Automation • Avionics	• UML	Não Definido	Não Definido	Não Definido
S6	• Knowledge Reuse	• Automation	• EAST-ADL	Não Definido	Não Definido	Não Definido
S7	• Adaptability • Safety	• Automation	• AADL • CCM	Não Definido	Não Definido	Não Definido
S8	• Não Definido	• Automotive	• Sysml • EAST-ADL	Não Definido	Não Definido	Não Definido
S9	• Maintainability • Reliability	• Automotive • Avionics	• UML • SDL	Não Definido	Não Definido	Não Definido
S10	• Não Definido	• Telecomunications	• Custom	Reference Model	Não Definido	Não Definido
S11	• Fault-Tolerance • Safety	• Automotive • Avionics	• ASDL	Software Architecture	Não Definido	Não Definido
S12	• Interoperability • Reliability	• Avionics	• UML	Reference Architecture	Não Definido	Não Definido
S13	• Correctness • Timing	• Avionics	• AADL • REAL	Não Definido	ADA	Não Definido
S14	• Adaptability • Safety	• Automation	• NCES	Software Architecture	Não Definido	Não Definido
S15	• Adaptability • Correctness	• Automotive • Telecomunications • Avionics	• UML-RT	Não Definido	Não Definido	Não Definido
S16	• Reliability	• Automotive • Avionics	• ACOL	Software Architecture	Não Definido	Não Definido
S17	• Performance	• Automation	• UML	Não Definido	Não Definido	Não Definido
S18	• Performance	• Avionics	• Metah	Software Architecture	Não Definido	Não Definido
S19	• Safety	• Avionics	• AADL • MARTE • Metah	Não Definido	Não Definido	SIMULINK
S20	• Dependability • Safety	• Automotive • Energy • Medical • Avionics	• UML • ACME • Metah	Não Definido	Não Definido	Não Definido
S21	• Não Definido	• Automation	• UML	Não Definido	Não Definido	Não Definido
S22	• Power Consumption	• Energy	• SAG	Não Definido	Não Definido	Não Definido
S23	• Knowledge Reuse	• Automotive • Medical • Avionics	• Custom	Reference Model	Não Definido	Não Definido
S24	• Correctness • Timing	• Automotive • Medical • Avionics	• DRTSADL	Software Architecture	Não Definido	Não Definido

4.3.2. Arquitetura do Experimento

O experimento foi realizado instanciando as seguintes dimensões: como dimensão *Origem*, definiu-se a possibilidade de combinação de técnicas a serem selecionadas (*DSour2*); como dimensão *Destino*, um único projeto de software (*DDest1*) chamado ***Núcleo de Excelência em Desenvolvimento de Sistemas Embarcados para Veículos Aéreos Não-Tripulados (VANTS) e Robôs Táticos Móveis*** realizado pela Universidade Federal do Amazonas em convênio com o INCT-SEC[7], que foi caracterizado através do formulário disponível no Apêndice B e mostrado na Tabela 9; e como dimensão *Objetivo*, dois objetivos de otimização foram definidos (*DObj2* - multiobjetivo). Os objetivos foram definidos como:

- **Objetivo de Requisitos (RO)**: compreende os atributos domínio de aplicação, atributos de qualidade e sistema crítico;
- **Objetivo Modelagem Técnica (TDO)**: compreende os atributos linguagem arquitetural e tipo de arquitetura;
- **Objetivo Infraestrutura de Codificação (ClO)**: compreende os atributos linguagem de programação e ferramentas.

Tabela 9. Caracterização do projeto VANTS.

Projeto de Software VANTS	
Atributos	**Caracterização**
Domínio de Aplicação	Aviônico, Tecnologia de Energia, Telecomunicações e Robótica.
Atributos de Qualidade requeridos no projeto	Corretude,
Linguagem Arquitetural usada s/conhecidas no projeto	
Tipo de Arquitetura de software	
Linguagem de Programação usada no projeto	C, C++ e Java.
Ferramentas desejadas/ conhecidas no projeto	
Se o projeto é considerado crítico	Sim.

O *benchmark* usado no experimento contendo 24 possibilidades de solução pode parecer pequeno, mas quando se combinam diferentes técnicas este número pode aumentar exponencialmente. Além disso, o número de técnicas identificadas na literatura técnica pode aumentar ao longo dos anos na medida em que novas abordagens forem propostas por outros pesquisadores. Neste caso, seria difícil de utilizar algoritmos exatos, uma vez que o STSP foi modelado como o problema do Conjunto Dominante Mínimo, que é um problema NP-Difícil (tal como descrito na seção 3.2.2).

[7] http://www.icomp.ufam.edu.br/inctsec/

4.3.3. Execução do Experimento e Resultados

Para a execução do experimento, também foram utilizadas duas meta-heurísticas disponíveis no framework JMetal e que estão integradas ao framework STSP: NSGAII, SPEA2 e MoCell. Os parâmetros utilizados para os algoritmos de otimização estão listados abaixo. Foi considerado o tamanho da população como 10 e número de soluções não-dominadas igual a 5 devido ao total de possibilidades ser igual ao número de técnicas (no caso 24):

- Tamanho da População = 10;
- Máximo de Avaliações = 10;
- Probabilidade de Cruzamento *(Crossover)* = 0.9;
- *Probabilidade de Mutação (Mutation)* = 0.1;
- Índice de Distribuição = 20;
- Número de Variáveis (Técnicas em uma solução) = 2;
- Número de Restrições (Repetições de uma técnica dentro da solução) = 1;
- Quantidade de Soluções não-dominadas arquivadas (*Archive Size*) = 5;
- Soluções Retornadas (*Feedback*) = 10.

Portanto, uma vez executado cada algoritmo, os resultados obtidos foram apresentados em um gráfico radar, pois como dito anteriormente esta é a forma de representação quando se tem mais de dois objetivos. O resultado está apresentado graficamente na (Figura 44) contendo duas soluções não-dominadas.

Os números obtidos e mostrados na listagem abaixo gráfico da Figura 44, representam a porcentagem de adequação das soluções com os respectivos objetivos definidos em relação às características do projeto. Na listagem podemos verificar que para essa execução tivemos apenas uma solução não-dominada. Na próxima seção iremos analisar o conjunto contendo todas as soluções geradas em todas as outras execuções.

4.3.4. Análise do Estudo de Caso 2

Para a execução do experimento, foram utilizadas três meta-heurísticas disponíveis no framework *JMetal* e integrado ao framework STSP: NSGAII, SPEA2 e MOCell.

Cada algoritmo foi executado 10 vezes e os resultados obtidos com as soluções não-dominadas foram apresentados em um gráfico de radar, porque três objetivos foram definidos para este experimento. Cada execução foi armazenada no banco de dados a ser consultado em um momento futuro. Os engenheiros de software pode reexecutar este experimento quantas vezes quiserem.

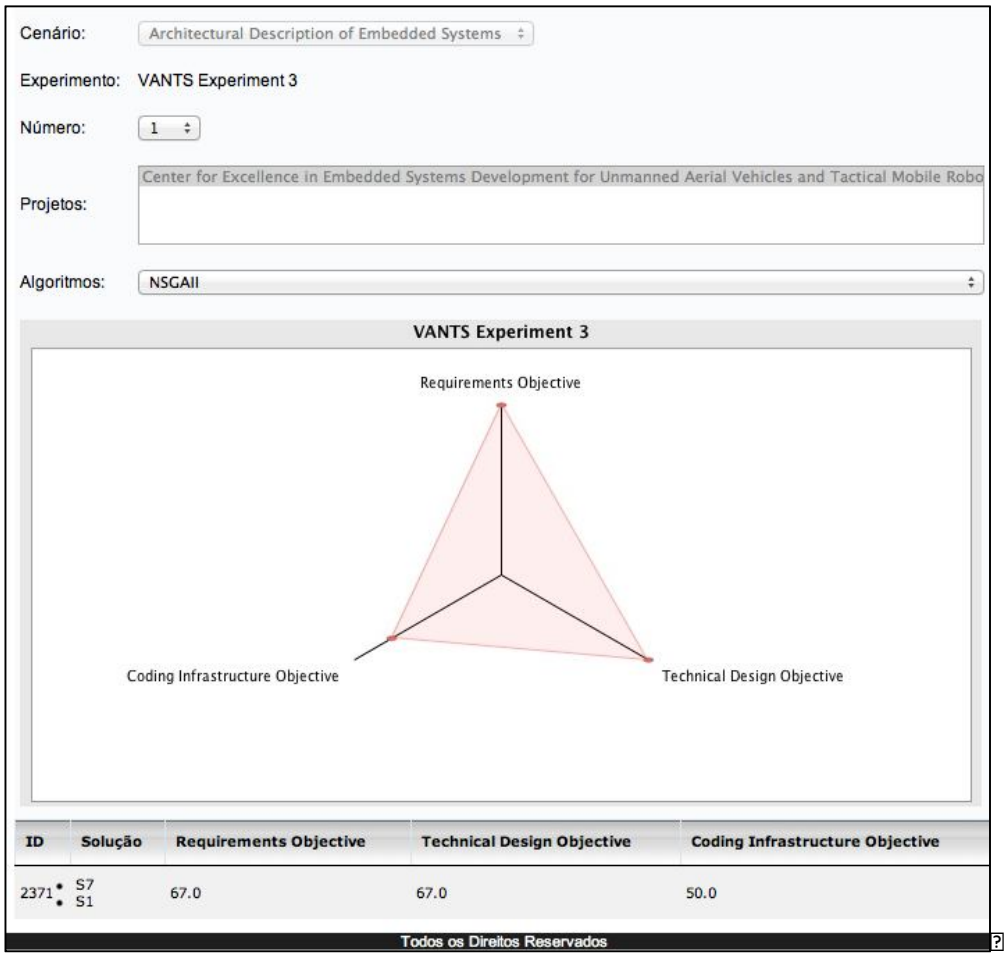


Figura 44. Resultados do Experimento para o Projeto VANTS

Usando o framework STSP, engenheiros de software podem escolher qual algoritmo (que também pode selecionar todos eles) e qual número de execução querem exibir as soluções geradas para comparar os resultados entre os algoritmos. Além da representação gráfica, os engenheiros de software podem ver a lista de soluções (combinação de técnicas ADES) com os respectivos valores calculados para cada objetivo. A partir dessa lista, os engenheiros de software terão mais informações para apoio para decisão sobre quais técnicas ADES usar no projeto de software.

A fim de executar a comparação de algoritmos, os resultados de todos os 10 ensaios foram unidos gerando um gráfico de radar para cada algoritmo (apenas com as soluções não-dominadas). Além disso, podemos também montar os resultados de todos os algoritmos no mesmo gráfico (Figura 45).

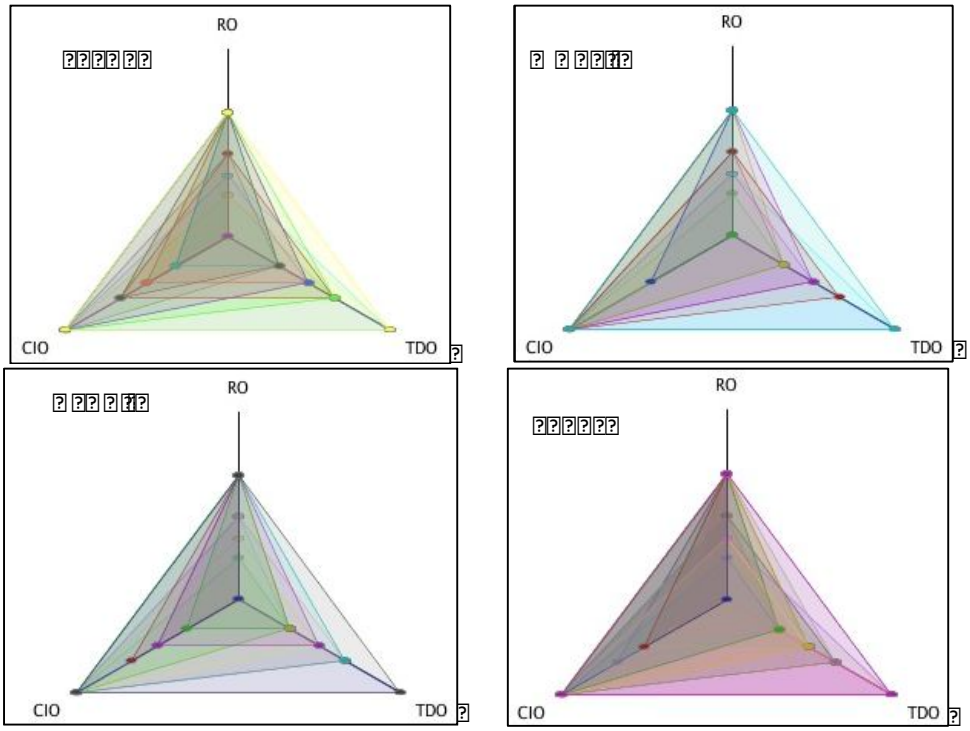


Figura 45. Comparativo dos gráficos de resultados dos algoritmos para o projeto VANTS.

Tabela 10. Distribuição das 30 melhores soluções geradas por objetivo.

Algoritmos	RO	TDO	CIO
Todos	4	3	5
SPEA2 e NSGAII	4	3	1
SPEA2 e MOCell	3	3	3
MOCell e NSGAII	2	2	2
SPEA2	12	13	10
NSGAII	10	10	9
MOCell	7	9	11

Foram analisados os 30 melhores resultados obtidos após todas as 10 execuções para cada algoritmo (SPEA2, MOCell ou NSGAII) com o intuito de fazer algumas observações sobre os gráficos gerados. Todos os três algoritmos geraram soluções de compromisso. No entanto, podemos analisar os gráficos priorizando cada objetivo e verificar quais as soluções tendem mais para um determinado objetivo. Assim, juntando os resultados obtidos nos três algoritmos selecionados e classificando as 30 melhores soluções, chegamos à distribuição apresentada na [illegible] (analisando objetivo por objetivo).

Priorizando o objetivo RO, observou-se que quatro soluções foram geradas por todos os algoritmos e os resultados que tendem para este objetivo foram as soluções geradas pelo algoritmo SPEA2, contendo 12 soluções. Priorizando o objetivo TDO, observamos que

3 soluções foram geradas por todos os algoritmos e os resultados que tendem para este objetivo foram também geradas pelos algoritmos SPEA2, contendo 13 soluções. Finalmente, priorizando o objetivo CIO, analisamos que 5 soluções foram geradas por todos os algoritmos e os resultados que tendem a este objetivo foram as soluções geradas pelo algoritmo MOCell, contendo 11 soluções.

Na Tabela 11, são apresentados os resultados de todas as rodadas e a união de todas as execuções (última coluna), a fim de realizar a comparação entre os algoritmos para várias métricas: soluções não dominadas (ND), as soluções não-dominadas distintas (DNDS – aquelas soluções em que as repetições são excluídas), o tempo de execução (TIME), o número de soluções repetidas (RS), e variedade de soluções combinadas geradas / execução (AVG COMBS).

Tabela 11. Métricas do experimento para cada algoritmo em todas as rodadas.

Comparando os resultados obtidos nos experimentos (mostrado na Tabela 11), em primeiro lugar, verificou-se que o algoritmo NSGAII obteve o maior número de soluções não-dominadas, bem como soluções não-dominadas distintas. Seguidos, respectivamente, por SPEA2 e MOCell. No entanto, considerando-se outras perspectivas, descobriu-se que a abordagem mais rápida foi SPEA2. Se for analisada a quantidade de soluções repetidas, SPEA2 e MOCell tiveram um desempenho muito semelhante e melhor do que NSGAII. Se for analisada a variedade de soluções combinadas geradas/execução, as soluções geradas pelo NSGAII contém, em média, mais técnicas do que os outros dois algoritmos.

Assim, finalizou-se o uso do framework STSP neste estudo de caso. Os passos apresentados nesta seção sugerem a viabilidade de utilizar o framework proposto para

instanciar cenários de seleção de tecnologias de software. Estão sendo planejadas novas instâncias para outros cenários de engenharia de software. Neste estudo de caso, o framework foi instanciado para o cenário de técnicas de modelagem arquitetural para sistemas embarcados e realizaram-se experimentos com um projeto de software real. O projeto foi caracterizado pelo responsável técnico através do uso de questionário de caracterização (Apêndice B). Os resultados foram coletados e apresentados ao responsável pelo projeto, que analisou os resultados e informou que as técnicas selecionadas poderiam ser aplicadas no projeto.

A partir deste resultado, definiu-se como próximos passos neste trabalho a avaliação formal da técnica instanciada neste projeto de software e em outros projetos desenvolvidos por outros subgrupos que pertencem ao INCT-SEC.

4.4. Considerações Finais

Neste capítulo foram realizados estudos de caso da aplicação do framework desenvolvido com o intuito de avaliar sua viabilidade no que diz respeito à instanciação de técnicas de apoio à seleção de tecnologias de software em cenários reais.

Devido à avaliação do framework ter sido realizada utilizando a técnica de análise de características através de estudos de casos (*feature analysis – case study*) da metodologia DESMET, os resultados não podem ser considerados conclusivos, haja visto que os experimentos foram realizados para projetos reais únicos em cenários específicos. No entanto, o resultado da avaliação foi considerado positivo pelos pesquisadores por ter atingido aos objetivos definidos, pois foi possível instanciar o framework para dois cenários distintos, realizar experimentos e obter resultados que podem ser validados por engenheiros de software ligados aos projetos escolhidos.

Portanto, a análise através de estudos de caso apresentou indícios da viabilidade da utilização do framework STSP para aquilo que este foi proposto. Em relação a este ponto, é necessária uma análise mais formal, com uma análise quantitativa, com a utilização do sistema pelos próprios engenheiros em um ambiente controlado no qual seja possível a coleta de dados para futuras análises. Como citado anteriormente, trabalhos futuros já estão sendo planejados no que diz respeito a estes aspectos.

CAPÍTULO 5: CONCLUSÃO

Neste capítulo serão apresentadas as conclusões desta dissertação, apresentando as suas contribuições e trabalhos futuros que fornecem a direção para que seja dada continuidade a esta pesquisa, além de suas limitações.

5.1. Considerações Finais

Este trabalho apresentou um framework para o problema de seleção de software tecnologias (STSP) que foi modelado como um problema de otimização combinatória. Tal framework funciona como uma ponte entre os engenheiros de software, que geralmente têm conhecimento limitado sobre pesquisa com base em algoritmos (meta-heurística) e os frameworks de otimização disponíveis, que implementam estes algoritmos e problemas de otimização combinatória.

Como contribuição, este framework pode ser um instrumento para engenheiros de software instanciarem novas abordagens para a seleção de tecnologias de software em diferentes cenários de engenharia de software. Além disso, repositórios de tecnologias de software podem ser criados utilizando os dados extraídos da literatura técnica. Assim, nas simulações e experimentos, engenheiros de software podem usar dados reais para avaliação de estratégias baseadas em busca e para a seleção de tecnologias de software mais adequadas para um projeto de software.

Extensibilidade é uma das características mais importantes do framework proposto. Novos frameworks e algoritmos de otimização podem ser integrados a ele para fornecer mais opções de seleção pelos engenheiros de software durante a seleção de tecnologias para projetos de software. Neste momento, ele está integrado ao framework JMetal e fornece três algoritmos evolucionários implementados em JMetal.

Nesta pesquisa, o framework proposto foi instanciado para dois cenários: seleção de técnicas de teste baseados em modelo (MBT) e seleção de técnicas de modelagem arquitetural para sistemas embarcados. Foi criado ainda um repositório de técnicas de MBT com dados reais extraídos da literatura técnica (DIAS-NETO e TRAVASSOS, 2010), além de criar um segundo repositório para técnicas de modelagem arquitetural para sistemas embarcados extraídos de (GUESSI et al., 2012). Após a configuração destes cenários, foram realizados alguns experimentos usando diferentes meta-heurísticas e

objetivos. Os resultados sugerem a viabilidade do framework proposto em instanciar estratégias de seleção de tecnologias de software.

Esta pesquisa possibilitou a publicação de um artigo no simpósio brasileiro de inteligência artificial (SBIA 2012), que serviu como prova de conceito do trabalho. Além deste, foi feita a submissão de um artigo científico para o evento *International Conference on Search-Based Software Engineering* (SSBSE 2013) no qual o resultado ainda não foi divulgado.

5.2. Contribuições

As principais contribuições oferecidas por esta pesquisa à comunidade de SBSE são:

- Desenvolvimento e disponibilização do framework STSP de apoio à tomada de decisões relativas à escolha de tecnologias de engenharia de software;
- Integração do Framework implementado com o framework de otimização JMetal e flexibilidade de integração com outros frameworks existentes;
- Mapeamento de técnicas de modelagem arquitetural para sistemas embarcados contendo 24 técnicas extraídas de uma revisão sistemática (GUESSI et al., 2012) e identificação de 7 atributos de caracterização do cenário, que formam um corpo de conhecimento para o cenário de técnicas ADES e que podem ser utilizadas em futuros experimentos.

5.3. Limitações

Algumas limitações foram observadas no decorrer da pesquisa:

- Para o cenário de técnicas de MBT, não foi possível realizar comparações dos resultados obtidos no framework STSP com as abordagens anteriores (DIAS-NETO et al., 2011) e (GRANDE et al., 2012). O motivo é que alguns objetivos descritos nos estudos anteriores possuem diferentes formas de cálculo, por exemplo: o objetivo esforço de modelagem salvo que possui uma fórmula específica de cálculo para o cenário de técnicas MBT, o que tornaria necessária a criação de um mecanismo de editor de fórmula para eles com a finalidade de deixar o framework genérico para diversos cenários;
- Foram integrados no framework STSP apenas três meta-heurísticas baseadas em algoritmos evolutivos disponíveis no JMetal devido à restrição de tempo. No entanto, o framework foi desenvolvido possibilitando a sua integração a outros frameworks e a inserção de novas meta-heurísticas;

- A forma atual de criação de repositórios de tecnologias ocorre de forma manual, realizando-se os cadastros dos atributos de caracterização um-a-um. Poderia ser desenvolvido um mecanismo de importação de arquivos BibText editados por ferramentas como JabRef, nas quais podem ser incluídos campos representando os atributos, e importados para o framework STSP;
- A avaliação do framework STSP foi feita via estudos de caso que simulam cenários reais de seleção de tecnologias para projetos de software. No entanto, esta categoria de estudos possui limitações em relação às possibilidades de conclusões a serem extraídas a respeito do framework desenvolvido.

5.4. Trabalhos Futuros

Como próximos passos estão previstos:

- Inclusão de novas meta-heurísticas disponíveis nos frameworks de otimização e/ou implementação de novas abordagens;
- Implementar um mecanismo de importação de cenários através de arquivos BibText;
- Realizar estudos comparativos entre os resultados obtidos através dos experimentos no framework STSP e os obtidos em trabalhos anteriores.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBA, E. and CHICANO, F. (2007). Finding Safety Errors with ACO. In Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO ’07), pages 1066–1073, London, England. ACM.

ANTONIOL, G., DI PENTA, M., and HARMAN, M. (2005). Search-based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project. In Proceedings of the 21st IEEE International Conference on Software Maintenance, pages 240–249, Los Alamitos, California, USA. IEEE Computer Society.

ARANDA, G. N., VIZCAINO, A., CECHICH, A., PIATTINI, M. (2006), “Technology Selection to Improve Global Collaboration”, In: International Conference on Global Software Engineering (ICGSE), October, pp. 223-232.

AZUMA, R. M. (2011). Otimização multiobjetivo em problema de estoque e roteamento gerenciados pelo fornecedor. Dissertação de Mestrado. UNICAMP, Campinas-SP.

BAGNALL, A. J., RAYWARD-SMITH, V. J., and WHITTLEY, I. M. (2001). The Next Release Problem. Information and Software Technology, 43(14), 883–890.

BARROS M. O. (2011). Evaluating Modularization Quality as an Extra Objective in Multiobjective Software Module Clustering, Proceedings of the 3rd International Symposium on Search Based Software Engineering (SSBSE '11), Springer, volume 6956, pages 267-267, Szeged, Hungary, 10-12 September 2011.

BASILI, V. CALDIEIRA, G. ROMBACH, H. (1994) "The Goal Question Metric Approach". Encyclopedia of Software Engineering (J.Willey & Sons).

BASILI, V. R., ROMBACH, H. D. (1991), “Support for comprehensive reuse”. Software Engineering Journal 6(5): September, 303-316.

BAUERSFELD S., WAPPLER S., WEGENER J. (2011). An Approach to Automatic Input Sequence Generation for GUI Testing using Ant Colony Optimization, Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11), ACM, pages 1915-1922, Dublin, Ireland, 12-16 July 2011.

BERTOLINO A. (2004). Guide to the knowledge area of software testing. Software engineering body of knowledge, in: IEEE Computer Society, February 2004.

BIRK, A. (1997), “Modeling the application domains of software engineering technologies”, In: International Conference on Automated Software Engineering (ASE). Lake Tahoe, CA, November.

BONDY, J., MURTY, U. (2008), Graph Theory: Graduated Text in Mathematics. Springer-Verlag.

BURGESS, C. J. and LEFLEY, M. (2001). Can Genetic Programming Improve Software Effort Estimation? A Comparative Evaluation. Information & Software Technology, 43(14), 863–873.

CANCIAN R.; STEMMER M.; FROLICH A. (2007) “New Developments in EPOS Tools for Configuring and Generating Embedded Systems.”, In Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation, pp. 776-779, Patras, Greece: 2007.

CANTÚ-PAZ, E. (2000). Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, 2000.

CHANG, C. K. (1994). Changing Face of Software Engineering. IEEE Software, 11(1), 4–5.

CHEN W. and ZHANG J. (2013). Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler, IEEE Transactions on Software Engineering, volume 39, number 1, pages 1-17, January 2013.

CHEN, D.-Y.a, CHUANG, T.-R.a, TSAI, S.-C.bc. (2001). “JGAP: A Java-based graph algorithms platform “ . Software - Practice and Experience, Volume 31, Issue 7, June 2001, Pages 615-635.

CHENG, B. and ATLEE, J. (2007). From state of the art to the future of requirements engineering. In L. Briand and A. Wolf, editors, Future of Software Engineering 2007, Los Alamitos, California, USA. IEEE Computer Society Press.

CHICANO, F. and ALBA, E. (2008). Searching for Liveness Property Violations in Concurrent Systems with ACO. In M. Keijzer, editor, Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO ’08), pages 1727–1734, Atlanta, GA, USA. ACM.

COLANZI T. E. and VERGILIO S. R. (2012). Applying Search Based Optimization to Software Product Line Architectures: Lessons Learned, Proceedings of the 4th International Symposium on Search Based Software Engineering (SSBSE '12), Springer, volume 7515, pages 259-266, Riva del Garda, Italy, 28-30 September

2012.

DAVIES, E., MCMASTER, J., and STARK, M. (1994). The Use of Genetic Algorithms for Flight Test and Evaluation of Artificial Intelligence and Complex Software Systems. Technical Report AD-A284824, Naval Air Warfare Center, Patuxent River.

DE FREITAS, F. G., MAIA, C. L. B., COUTINHO, D. P., DE CAMPOS, G. A. L., DE SOUZA, J. T. “Aplicação de Meta-heurísticas em Problemas da Engenharia de Software: Revisão de Literatura”, II Congresso Tecnológico Infobrasil, 2009.

DEB K. (2001). Multi-objective optimization using evolutionary algorithms, John Wiley & Sons, 2001.

DEB K., PRATAP A., AGARWAL S., MEYARIVAN T., A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

DEMŠAR, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets, J. Mach. Learn. Res. 7 (2006) 1–30.

DIAS-NETO, A. C.; TRAVASSOS, G. H. (2010). Evolving a Computerized Infrastructure to support the Selection of Model-Based Testing Techniques. In: IFIP International Conference on Testing Software and Systems (ICTSS'10), 2010, Natal.

DIAS-NETO, A.C.; RODRIGUES, R.F.; TRAVASSOS, G. H. (2011), Porantim-Opt: Optimizing the Combined Selection of Model-based Testing Techniques, In: 4th International Workshop on Search-Based Software Testing (SBST), March, Berlin, Germany.

DIAS-NETO, A.C.; TRAVASSOS, G.H. (2009). “Model-based Testing Approaches Selection for Software Projects”, In Automation of Software Test, 2009. AST '09. ICSE Workshop, on 18-19 May 2009, p. 1487-1504.

DIAS-NETO, A.C.; TRAVASSOS, G.H. (2009a). “Porantim: An Approach to Support the Combination and Selection of Model-Based Testing Techniques”, In: Information and Software Technology, v. 51, p. 1-9.

DURILLO, J.J.; NEBRO, A.J.; (2011); “jMetal: a Java Framework for Multi-Objective Optimization”. Advances in Engineering Software, v. 42, pp. 760-771.

EDGEWORTH, F. Y. (1881). Mathematical physics. P. Keagan, London, England.

FARZAT F. A., BARROS M. O. (2010). Test Case Selection Method for Emergency Changes (in Portuguese), Proceedings of the Brazilian Workshop on Optimization in Software Engineering (WOES '10), Salvador, Brazil, 30-30 September 2010.

FEATHER, M. S. and MENZIES, T. (2002). Converging on the Optimal Attainment of Requirements. In Proceedings of the 10th IEEE International Conference on Requirements Engineering (RE ’02), pages 263–270, Essen, Germany. IEEE.

FERGUSON, R. and KOREL, B. (1996). The Chaining Approach for Software Test Data Generation. ACM Transactions on Software Engineering and Methodology (TOSEM), 5(1), 63–86.

FIGUEIREDO F. and BARROS M. O. (2011). Heuristic Optimization of a Technique for Selection and Prioritization balanced portfolio of Software Projects (in Portuguese), Proceedings of the 2nd Brazilian Workshop on Search Based Software Engineering (WESB '11), Sao Paulo, Brazil, 26-26 September 2011.

FREITAS F. G., COUTINHO D. P., J. T. SOUZA (2011). Software Next Release Planning Approach through Exact Optimization. International Journal of Computer Applications, May 2011, volume 22, number 8.

GAREY, M., JOHNSON, D. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York, NY, 1979.

GHAITH S. and Ó CINNÉIDE M. (2012). Improving Software Security Using Search-Based Refactoring, Proceedings of the 4th International Symposium on Search Based Software Engineering (SSBSE '12), Springer, volume 7515, pages 121-135, Riva del Garda, Italy, 28-30 September 2012.

GLOVER F. (1986), “Future paths for integer programming and links to artificial intelligence”, Computer Operational Research 13, pp. 533-549, 1986.

GODEFROID, P. (1997). Model Checking for Programming Languages using Verisoft. In Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, pages 174–186, Paris, France. ACM.

GONÇALVES , E. (2008). Dominando Java Server Faces e Facelets Utilizando Spring 2.5, Hibernate e JPA. 1.ª edição – Editora Ciência Moderna, 2008, ISBN: 9788573937114.

GONZALEZ T. F. (2007). Handbook of Approximation Algorithms and Metaheuristics. Chapter Basic Methodologies. University of California, Santa Barbara by Taylor & Francis Group, LLC.

GRAHAM, R. L. (1966), Bounds for certain multiprocessing anomalies, Bell System Tech. J., 45, 1563, 1966.

GRANDE, A.; DIAS-NETO, A.C.; RODRIGUES, R. (2012), “Providing trade-off techniques subsets to improve software testing effectiveness: using evolutionary algorithm to support software testing techniques selection by a web tool”, In: Brazilian Symposium on Artificial Intelligence, Curitiba, pp. 1-10, October.

GREER, D. and RUHE, G. (2004). Software Release Planning: An Evolutionary and Iterative Approach. Information & Software Technology, 46(4), 243–253.

GUESSI M., NAKAGAWA E. Y., OQUENDO F., MALDONADO J. C. (2012). Architectural Description of Embedded Systems: A Systematic Review. Proceeding ISARCS '12 Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems. Pages 31-40 ACM New York, NY, USA ©2012.

HAAS, J., PEYSAKHOV, M., and MANCORIDIS, S. (2005). GA-based Parameter Tuning for Multi-Agent Systems. In Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO ’05), pages 1085–1086, Washington, D.C., USA. ACM.

HARMAN, M. and CLARK, J. A. (2004). Metrics Are Fitness Functions Too. In Proceedings of the 10th IEEE International Symposium on Software Metrics (METRICS’04), pages 58–69, Chicago, USA. IEEE Computer Society.

HARMAN, M., HIERONS, R., and PROCTOR, M. (2002). A New Representation and Crossover Operator for Search based Optimization of Software Modularization. In Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO ’02), pages 1351–1358, New York, USA. Morgan Kaufmann Publishers.

HARMAN, M., JONES, B.F. (2001), Search-based software engineering, Information and Software Technology, 2001, pp. 833-839.

HARMAN, M., MANSOURI S. A. and ZHANG Y. (2009), Search-based software engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. Technical Report TR-09-03, 2009.

HOSTE, K. and EECKHOUT, L. (2008). COLE: Compiler Optimization Level Exploration.

In Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization, pages 165–174, Boston, MA, USA. ACM.

JACCARD, P. (1901), Étude comparative de la distribution florale dans une portion des Alpes et des Jura, Bulletin del la Société Vaudoise des Sciences Naturelles 37, pp. 547-579.

JENSEN A. C. and CHENG B. H.C. (2010). On the Use of Genetic Programming for Automated Refactoring and the Introduction of Design Patterns, Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10), ACM, pages 1341-1348, Portland, Oregon, USA, 7-11 July 2010.

JIANG H., ZHANG J., XUAN J., RE Z., HU Y. (2010), A Hybrid ACO Algorithm for the Next Release Problem, Proceedings of the 2nd International Conference on Software Engineering and Data Mining (SEDM '10), IEEE, 23-25 June 2010, pages 166-171, Chengdu, China.

JOHNSON, C. (2007). Genetic Programming with Fitness based on Model Checking. In Proceedings of the 10th European Conference on Genetic Programming, volume 4445 of LNCS, pages 114–124, Valencia, Spain. Springer.

KATZ G. and PELED D. (2010). Code Mutation in Verification and Automatic Code Correction, Proceedings of 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '10) Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS '10), Springer, volume 6015, pages 435-450, Paphos, Cyprus, 20-28 March 2010.

KATZ, G. and PELED, D. (2008). Model Checking-Based Genetic Programming with an Application to Mutual Exclusion. In Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08), pages 141–156, Budapest, Hungary. Springer.

KITCHENHAM B., LINKMAN S. and LAW D. (1997). DESMET: A method for evaluating software engineering methods and tools. Computing Control Engineering Journal, June 1997.

KNOWLES J. and CORNE D., "The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization," in Proceedings of the 1999 Congress on Evolutionary Computation. Piscataway, NJ: IEEE Press, 1999, pp. 98–105.

KNOWLES J., THIELE L., ZITZLER E. (2006). A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers, Tech. Rep. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich (2006).

KOREL, B. (1990). Automated Software Test Data Generation. Transactions on Software Engineering, SE-16(8), 870–879.

KPODJEDO S., RICCA F. ANTONIOL G., GALINIER P. (2009). Evolution and Search Based Metrics to Improve Defects Prediction, Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE '09), IEEE, pages 23-32, Cumberland Lodge, Windsor, UK, 13-15 May 2009.

KRONFELD, M.; PLANATSCHER, H.; ZELL, A.; (2010); "The {EvA2} Optimization Framework. Learning and Intelligent Optimization Conference", Special Session on Software for Optimization (LION-SWOP). Lecture Notes in Computer Science, LNCS. Pp: 247-250. Venice, Italy.

LANGE, R. and MANCORIDIS, S. (2007). Using Code Metric Histograms and Genetic Algorithms to Perform Author Identification for Software Forensics. In Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07), pages 2082–2089, London, England. ACM.

LEE S., BAE G., CHAE H. S., BAE D., KWON Y. R., Automated Scheduling for Clone-based Refactoring using a Competent GA, Software: Practice and Experience, volume 41, number 5, pages 521-550, April 2011.

LEFTICARU R., IPATE F., TUDOSE C. (2009). Automated Model Design using Genetic Algorithms and Model Checking, Proceedings of the 4th Balkan Conference in Informatics (BCI '09), IEEE, pages 79-84, Thessaloniki, Greece, 17-19 September.

LEMNARU, C. , CUIBUS, M. , BONA, A., ALIC, A., POTOLEA, R. (2012). "A distributed methodology for imbalanced classification problems". 11th International Symposium on Parallel and Distributed Computing (ISPDC). Pp. 164-171.

LUKASIEWYCZ M.; GLAß, M.; REIMANN, F.; TEICH J. (2011); "Opt4J - A Modular Framework for Meta-heuristic Optimization". Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011), Dublin, Ireland, pp: 1723-1730.

LUKE S. (2010). ECJ Evolutionary Computation System. Copyright ©2000 by Sean Luke, All Rights Reserved http://www.cs.umd.edu/users/seanl/ec/ecj/.

LUKE S. (2011). Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes. Department of Computer Science George Mason University, First Edition. (Rev C) Online Version 1.2 July 2011.

MAHDAVI, K., HARMAN M., HIERONS, R. M. (2003). A Multiple Hill Climbing Approach to Software Module Clustering. In Proceedings of the International Conference on Software Maintenance ICSM'03, page 315, IEEE Computer Society Washington, DC, USA ©2003, ISBN:0-7695-1905-9.

MAIDEN, N. A. M.; RUGG, G. (1996), "ACRE: Selecting methods for requirements acquisition", Software Engineering Journal 11(3): 183-192.

MANCORIDIS, S., MITCHELL, B. S., CHEN, Y., and GANSNER, E. R. (1999). Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99), pages 50–59, Oxford, England, UK. IEEE.

MANCORIDIS, S., MITCHELL, B. S., RORRES, C., CHEN, Y., and GANSNER, E. R. (1998). Using Automatic Clustering to Produce High-Level System Organizations of Source Code. In Proceedings of the 6th International Workshop on Program Comprehension (IWPC '98), pages 45–52, Ischia, Italy. IEEE Computer Society Press.

MANDERICK B. and SPIESSENS P. Fine-grained parallel genetic algorithm. In Proc. of the Third Int. Conf. on Genetic Algorithms (ICGA), pages 428–433, 1989.

MCMINN P. (2004), "Search-Based Software Test Data Generation: A Survey," Software Testing, Verification and Reliability, vol. 14, pp. 105-156, 2004.

MILLER, W. and SPOONER, D. L. (1976). Automatic Generation of Floating-Point Test Data. IEEE Transactions on Software Engineering, 2(3), 223–226.

NEBRO A. J., DURILLO J. J., LUNA F., DORRONSORO B. and ALBA E. A Cellular Genetic Algorithm for Multiobjective Optimization, in David A. Pelta and Natalio Krasnogor (editors), Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO 2006), pp. 25--36, Granada, Spain, 2006.

O'KEEFFE, M. and Ó CINNÉIDE, M. (2004). Towards Automated Design Improvement Through Combinatorial Optimization. In Proceedings of the 26th International Conference on Software Engineering and Workshop on Directions in Software Engineering Environments (WoDiSEE'04), pages 75–82, Edinburgh, UK. ACM.

OSSADA, J.C.A AND MARTINS, L.E.G.B (2010), A field study on the state of practice of requirements elicitation in embedded systems [Um estudo de campo sobre o estado da prática da elicitação de requisitos em sistemas embarcados]}. WER, pages 41-52, Cuenca.

PARETO, V. (1896). Cours d'economie politique, volume i and ii. F. Rouge, Lausanne.

PMBOK (2004). A Guide To The Project Management Body Of Knowledge (PMBOK Guides), Project Management Institute, 2004.

RÄIHÄ O. (2010). A Survey on Search-based Software Design, Computer Science Review, volume 4, number 4, pages 203-249, November 2010.

RÄIHÄ, O., KOSKIMIES, K., MÄKINEN, E., and SYSTÄ, T. (2008). Pattern-Based Genetic Model Refinements in MDA. In Proceedings of the Nordic Workshop on Model-Driven Engineering (NW-MoDE '08), pages 129–144, Reykjavik, Iceland. University of Iceland.

REFORMAT, M., CHAI, X., and MILLER, J. (2007). On the Possibilities of (Pseudo-) Software Cloning from External Interactions. Soft Computing - A Fusion of Foundations, Methodologies and Applications, 12(1), 29–49.

RIFKI O. and ONO H. (2012). A survey of computational approaches to portfolio optimization by genetic algorithms. Department of Economic Engineering Kyushu University Fukuoka 812-8581, Japan.

SALIU, M. O. and RUHE, G. (2007). Bi-Objective Release Planning for Evolving Software Systems. In Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the Acm Sigsoft Symposium on the Foundations of Software Engineering, pages 105–114, Dubrovnik, Croatia. ACM.

SCHOENAUER, M. and XANTHAKIS, S. (1993). Constrained GA Optimization. In Proceedings of the 5th International Conference on Genetic Algorithms (ICGA '93), pages 573– 580, San Mateo, CA, USA. Morgan Kauffmann.

SILVERMAN B. W. Density estimation for statistics and data analysis. Chapman and Hall, London, (1986).

SIMONS C. L., PARMEE I. C. (2009). An Empirical Investigation of Search-based Computational Support for Conceptual Software Engineering Design, Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC '09), IEEE, pages = 2503-2508, San Antonio, USA, 11-14 October 2009.

SIMONS, C. L. and PARMEE, I. C. (2008). User-centered, Evolutionary Search in Conceptual Software Design. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC '08) (World Congress on Computational Intelligence), pages 869–876, Hong Kong, China. IEEE.

SIQUEIRA T. F., MENEGOTO, C. C., WEBER, T. S., NETTO, J. C., WAGNER, F. R. (2006). Desenvolvimento de sistemas embarcados para aplicações críticas. CD-ROM, Anais. 4ª Escola Regional de Redes de Computadores, Passo Fundo-RS.

SPIETH, C. , SUPPER, J., STREICHERT, F., SPEER, N., ZELL, A. (2006). "JCell - A Java-based framework for inferring regulatory networks from time series data". Bioinformatics, Volume 22, Issue 16, 15 August, Pages 2051-2052.

SRINIVAS N. and DEB K., "Multiobjective function optimization using nondominated sorting genetic algorithms," Evol. Comput., vol. 2, no. 3, pp. 221–248, Fall 1995.

STREICHERT, F., ULMER, H. (2005), JavaEvA - A Java Framework for Evolutionary Algorithms. Technical Report WSI-2005-06, Center for Bioinformatics Tübingen, Uni- versity of Tübingen.

TALBI, El-G. (2009). Metaheuristics: from design to implementation. University of Lille – CNRS – INRIA by John Wiley & Sons.

TOMASI A., MARCHETTO A., FRANCESCOMARINO C. (2012). Domain-Driven Reduction Optimization of Recovered Business Processes, Proceedings of the 4th International Symposium on Search Based Software Engineering (SSBSE '12), Springer, volume 7515, pages 228-243, Riva del Garda, Italy, 28-30 September.

VAN VELDHUIZEN, D. A. e LAMONT, G. B (1998). Multiobjective evolutionary algorithm research: A history and analysis, Tech. Rep. TR-98-03, Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst.Technol., Wright-Patterson, AFB, OH.

VEGAS, S.; BASILI, V. (2005). "A Characterization Schema for Software Testing Techniques", Journal of Empirical Software Engineering, v.10 n.4, p.437-466.

VENTURA, S. , ROMERO, C. , ZAFRA, A. , DELGADO, J.A. , HERVÁS, C. (2008). JCLEC: A Java framework for evolutionary computation. Soft Computing, Volume 12, Issue 4, February, Pages 381-392.

VICTOR, M. and UPADHYAY, N. (2011), “Selection of Software Testing Technique: A Multi Criteria Decision Making Approach”, Communications in Computer and Information Science, Vol. 204, Part 1, 453-462, DOI: 10.1007/978-3-642-24043-0_46.

VIVANCO, R. and JIN, D. (2008). Enhancing Predictive Models using Principal Component Analysis and Search Based Metric Selection: A Comparative Study. In Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM’08), pages 273–275, Kaiserslautern, Germany. ACM.

VOS T. E. J., LINDLAR F. F., WILMES B., WINDISCH A., BAARS A. I., KRUSE P. M., GROSS H., WEGENER J. (2013). Evolutionary Functional Black-box Testing in an Industrial Setting, Software Quality Journal, volume 21, pages 259-288, June 2013.

WILSON G. C., MC INTYRE A., HEYWOOD M. I. (2004). Resource Review: Three Open Source Systems for Evolving Programs–Lilgp, ECJ and Grammatical Evolution. Genetic Programming and Evolvable Machines March 2004, Volume 5, Issue 1, pp 103-105.

WOJCICKI, M. A.; STROOPER, P. (2007). “An Iterative Empirical Strategy for the Systematic Selection of a Combination of Verification and Validation Technologies”. In: international Workshop on Software Quality (May 20 - 26, 2007).

XANTHAKIS, S., ELLIS, C., SKOURLAS, C., LE GALL, A., KATSIKAS, S., and KARAPOULIOS, K. (1992). Application of Genetic Algorithms to Software Testing. In Proceedings of the 5th International Conference on Software Engineering and Applications, pages 625–636, Toulouse, France.

XUAN J., JIANG H., REN Z., LUO Z. (2012). Solving the Large Scale Next Release Problem with a Backbone Based Multilevel Algorithm, IEEE Transactions on Software Engineering, Sept.-Oct. 2012, volume 38, number 5, pages 1195-1212.

YOO S. and HARMAN M. (2012). Test Data Regeneration: Generating New Test Data from Existing Test Data, Journal of Software Testing, Verification and Reliability, volume 22, number 3, pages 171-201, May 2012.

ZHANG Y., HARMAN M., LIM S. L. (2013). Empirical Evaluation of Search Based Requirements Interaction Management, Information and Software Technology, volume 55, number 1, pages 126-152, January, 2013.

ZHANG, Y., HARMAN, M., and MANSOURI, S. A. (2007).The Multi-Objective Next Release Problem. In Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07), pages 1129–1137, London, UK. ACM.

ZHOU A, JIN Y, ZHANG Q, SENDHOFF B, TSANG E (2006). Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion. In: 2006 IEEE Congress on evolutionary computation; 2006. p. 3234–41.

ZITZLER E. and THIELE L., "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation, 3*(4), 257-271, (1999).

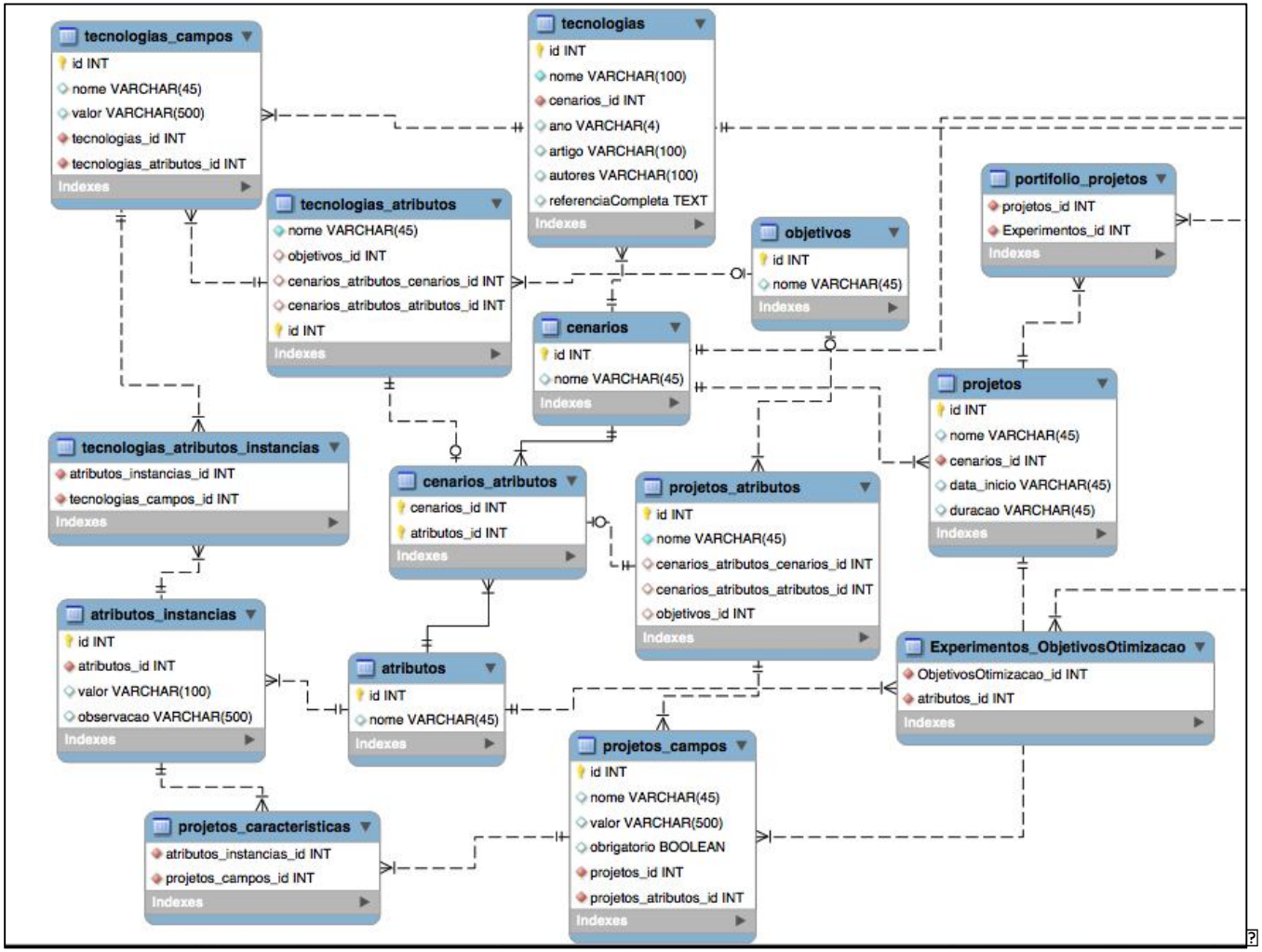
ZITZLER E., "Evolutionary algorithms for multiobjective optimization: Methods and applications," Doctoral dissertation ETH 13398, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1999.

ZITZLER E., LAUMANNS M., THIELE L. (2002), SPEA2: Improving the strength pareto evolutionary algorithm, in: K. Giannakoglou, D. Tsahalis, J. Pe- riaux, P. Papailou, T. Fogarty (Eds.), EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece, pp. 95–100.

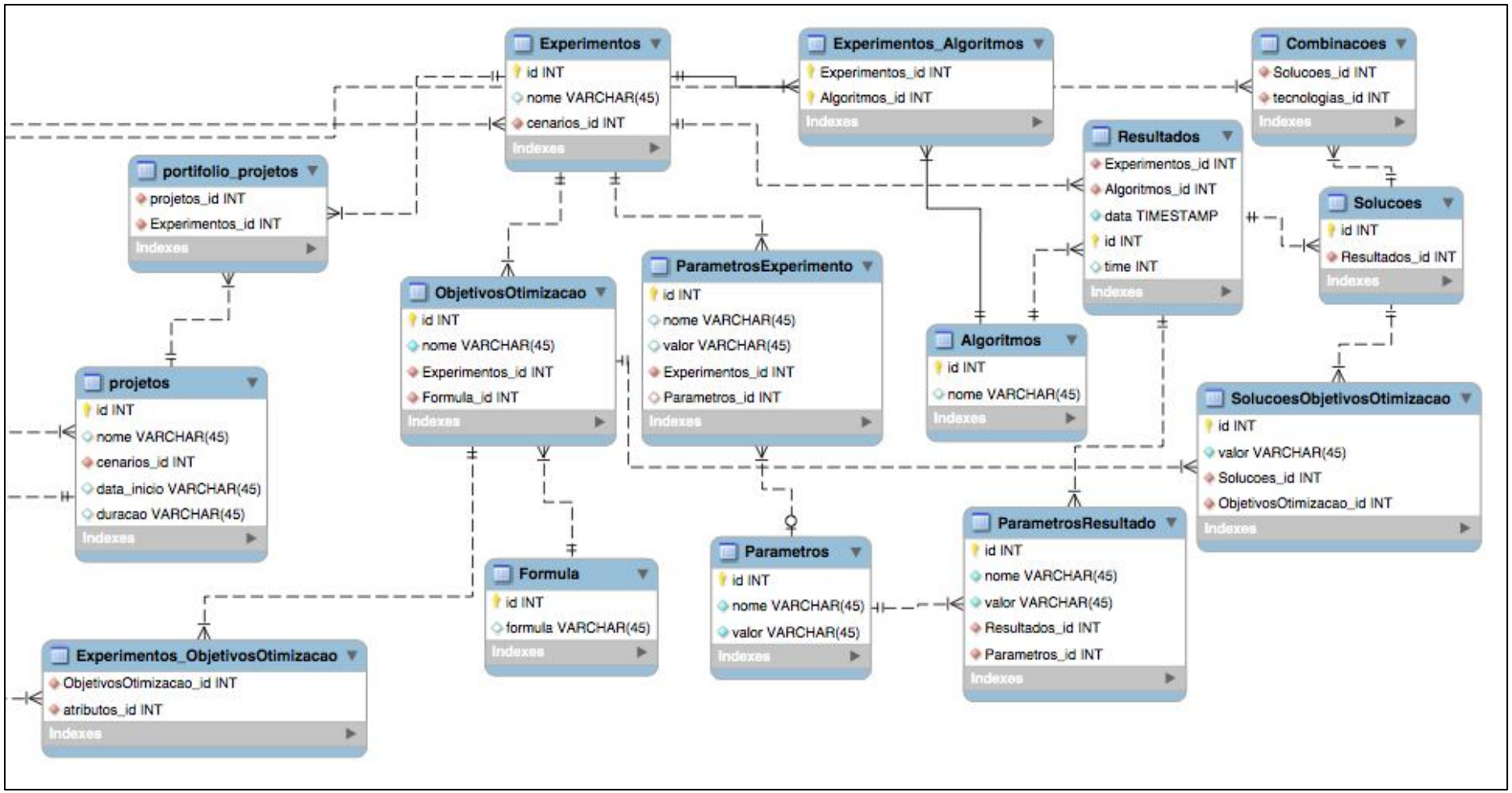
ZITZLER E., THIELE L. (1999). Multiobjective evolutionary algorithms: a com- parative case study and the strength pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271.

Apêndice A – Modelagem do Banco para Framework STSP

A.1. Modelagem do Banco para STSP



A.2. Modelagem do Banco para Experimentos



Apêndice B – Formulário de Caracterização de Projetos

