THELMA ELITA COLANZI

UMA ABORDAGEM DE OTIMIZAÇÃO MULTIOBJETIVO PARA PROJETO ARQUITETURAL DE LINHA DE PRODUTO DE SOFTWARE

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Silvia Regina Vergilio

CURITIBA

2014

THELMA ELITA COLANZI

UMA ABORDAGEM DE OTIMIZAÇÃO MULTIOBJETIVO PARA PROJETO ARQUITETURAL DE LINHA DE PRODUTO DE SOFTWARE

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Silvia Regina Vergilio

CURITIBA

2014

C683a Colanzi, Thelma Elita
Uma abordagem de otimização multiobjetivo para projeto arquitetural de linha de produto de software / Thelma Elita Colanzi. – Curitiba, 2014.
195f. : il. color. ; 30 cm.

Tese (doutorado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2014.

Orientador: Silvia Regina Vergilio.
Bibliografia: p. 140-153.

1. Engenharia de software. 2. Planejamento da produção. 3. Otimização. I. Universidade Federal do Paraná. II. Vergilio, Silvia Regina. III. Título.

CDD: 005.12

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa da aluna de Doutorado em Ciência da Computação, Thelma Elita Colanzi Lopes, avaliamos a tese de doutorado intitulada *"Uma abordagem de otimização multiobjetivo para projeto arquitetural de linha de produto de software"*, cuja defesa pública foi realizada no dia 21 de março de 2014, às 09:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após avaliação, decidimos pela:

(X) **aprovação** da candidata. ( ) **reprovação** da candidata.

Curitiba, 21 de março de 2014.

Profa. Dra. Silvia Regina Vergilio
DINF/UFPR – Orientadora

Prof. Dr. Alessandro Fabricio Garcia
PUC/RJ – Membro Externo

Profa. Dra. Itana Maria de Souza Gimenes*
UEM – Membro Externo

Prof. Dr. Jerffeson Teixeira de Souza
UECE – Membro Externo

Profa. Dra. Aurora Trinidad Ramirez Pozo
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

* Participação por videoconferência

AGRADECIMENTOS

Durante esses quatro anos da minha vida, muitas pessoas contribuíram das mais variadas formas para que eu realizasse esse trabalho de pesquisa. Por isso, é necessário agradecer...

a Deus, pela vida e pela saúde física e mental.

ao meu esposo Leonardo, pelo companheirismo, paciência, compreensão e incentivo, por tentar me ajudar nos momentos difíceis mesmo sem entender das questões técnicas, e por despertar em mim, a cada dia, a vontade de ser uma pessoa e profissional melhor.

aos meus pais por me estimularem desde cedo a sempre dar o melhor de mim em tudo o que eu fizesse; à minha sogra Sonia, que pelo seu exemplo de fibra, me ensinou que é possível se dedicar à profissão sem deixar de lado a família; e aos meus irmãos, sogro e demais familiares, pelos momentos de fraternidade que deixaram os dias da minha jornada mais leves.

a Silvia, pela dedicação e disponibilidade em me orientar; pelas valiosas lições e pela amizade que construímos além da relação aluno-orientador.

ao Masiero, pelas lições que me prepararam como pesquisadora e professora durante o mestrado, e que no doutorado percebi o quanto foram primordiais.

ao Wesley, pelos bons momentos, troca de ideias, parceria em vários trabalhos durante essa jornada e pela amizade.

a Aurora, pela disponibilidade em esclarecer minhas dúvidas, pela troca de ideias e amizade.

a Itana pelo incentivo, pela rica troca de ideias, pela disposição de querer ajudar sempre e pela amizade.

ao Alessandro pelas boas ideias trocadas no exame de qualificação que serviram tanto para delimitar como para ampliar os horizontes em relação ao tema da pesquisa.

a Sandra pela revisão do texto, por conduzir o projeto de pesquisa e acolher meus alunos durante meu afastamento e pela amizade.

a Lucília e aos demais colegas do CBio-GrES e da UFPR: Arion, Luiz Camargo, André, Olacir, Daday, Giovani, Édipo, Murilo, Thainá, Rui, Luiz Leandro, Pedro, João Eugênio, pelos cafés, almoços e papos que tornaram meus dias em Curitiba muito mais interessantes.

as minhas amigas de tantos anos e momentos Aline, Lu, Tati e Val, e a todos os demais amigos, cujos nomes não vou mencionar devido ao tamanho da lista, obrigada por cada momento de descontração que desfrutamos.

aos meus ex-alunos Willian, Marcos e Rômulo pelo apoio técnico imprescindível durante a fase de implementação.

ao Édipo por todo o apoio na execução dos experimentos da tese.

aos meus colegas de trabalho do DIN/UEM, cuja colaboração foi imprescindível para que eu pudesse me dedicar integralmente ao doutorado.

ao pessoal do DInf/UFPR: coordenação, professores, e secretários Jucélia e Rafael.

e ao CNPq e à Fundação Araucária pelo apoio financeiro.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACO *Ant Colony Optimization*

AGM *Arcade Game Maker*

ATMR Elegância da razão entre atributos e métodos

CDAC *Concern Diffusion over Architectural Components*

CDAI *Concern Diffusion over Architectural Interfaces*

CDAO *Concern Diffusion over Architectural Operations*

CDepIn Dependências de Entrada de uma Classe

CDepOut Dependências de Saída de uma Classe

CM Métricas convencionais

CIBC *Component-level Interlacing Between Concerns*

CPSU *Compound Provided Service Utilization*

CRA *Class Responsability Assignment*

CRSU *Compound Required Service Utilization*

DepIn Dependências de Entrada

DepOut Dependências de Saída

DepPack Dependência de Pacotes

EC Elegância de acoplamentos externos

ED Distância Euclidiana à Solução Ideal

FM Métricas dirigidas a características

GA Algoritmos Genéticos

H Coesão Relacional

HV Hipervolume

IIBC *Interface-level Interlacing Between Concerns*

LCC *Lack of Concern-based Cohesion*

LPS Linha de Produto de Software

MM *Mobile Media*

MOA4PLA *Multi-Objective Approach for Product-Line Architecture Design*

MOEA *Multi-objetive Evolutionary Algorithms*

MSI Métricas Sensíveis a Interesses

NAC Elegância de números entre classes

NSGAII *Non-dominated Sorting Genetic Algorithm*

NumOps Número de Operações por Interface

OO Orientado a Objetos

OOBC *Operation-level Overlapping Between Concerns*

PACO *Pareto Ant Colony Optimization*

PAES *Pareto Archived Evolution Strategy*

PLA *Product Line Architecture*

PSO *Particle Swarm Optimization*

PSU *Provided Service Utilization*

RSU *Required Service Utilization*

SBSE *Search Based Software Engineering*

SEI *Software Engineering Institute*

SPEA2 *Strength Pareto Evolutionary Algorithm*

SPL *Software Product Line*

UML *Unified Modeling Language*

XMI *XML Metadata Interchange*

LISTA DE PUBLICAÇÕES

Durante o período do doutorado vários artigos científicos foram publicados em conferências e periódicos. Alguns são resultados de trabalhos desenvolvidos em disciplinas cursadas, que serviram de subsídio para o desenvolvimento da tese. Dentre eles estão os estudos sobre teste de linha de produto de software, aplicação de algoritmos de busca para problema de clusterização de dados e aplicação de algoritmos evolutivos multiobjetivos para o problema de integração e teste de classes e aspectos.

Outros estudos envolveram projetos de pesquisa não vinculados diretamente ao tema da tese, mas que se relacionam com algum tópico envolvido na tese. Esses estudos envolvem métricas arquiteturais, linhas de produto de software e a produção da comunidade brasileira de *Search Based Software Engineering*.

Esses artigos são listados a seguir e estão agrupados por assunto. Ao final, mas não menos importantes, são listados os artigos relacionados à tese que já foram publicados. Outros artigos já submetidos se encontram em fase de avaliação.

Estudos sobre teste de linha de produto de software:

1. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; TRINDADE, D. F. G.; ZORZO, C. A.; VERGILIO, S. R. Evaluating different strategies for testing software product lines. *Journal of Electronic Testing: Theory and Applications (JETTA)*. v. 29, p. 9-24, 2013.

2. FÉDERLE, E. L.; GUIZZO, G.; COLANZI, T. E.; VERGILIO, S. R.; SPINOSA, E. J. . Seleção de produto baseada em algoritmos multiobjetivos para o teste de mutação de variabilidades. In: *IV Workshop de Engenharia de Software Baseada em Busca (WESB)*, Brasília. 2013.

3. ASSUNÇÃO, W. K. G.; TRINDADE, D. F. G.; COLANZI, T. E.; VERGILIO, S. R. Evaluating test reuse of a software product line oriented strategy. In: *12th IEEE Latin-American Test Workshop (LATW)*, Porto de Galinhas - PE. 2011.

4. ZORZO, C. A.; SILVA, R. A. L.; COLANZI, T. E.; VERGILIO, S. R. Aplicação de uma Estratégia Incremental para o Teste de Linha de Produto de Software. In: *Simpósio Brasileiro de Qualidade de Software*, Curitiba. 2011.

Estudos sobre aplicação de algoritmos de busca para problema de clusterização de dados:

1. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; POZO, A. T. R.; VENDRAMIN, A. C. B. K.; PEREIRA, D. A. B.; PAULA FILHO, P. L.; ZORZO, CARLOS ALBERTO. Application of Bio-inspired Metaheuristics in the Data Clustering Problem. *CLEI Electronic Journal*, v. 14, p. 5, 2011.

2. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; VENDRAMIN, A. C. B. K.; PEREIRA, D. A. B.; POZO, A. T. R. Empirical Studies on Application of Genetic Algorithms and Ant Colony Optimization for Data Clustering. In: *XXIX International Conference of the Chilean Computer Science Society (SCCC 2010)*, 2010, Antofagasta/Chile. 2010. p. 1-10.

Estudos sobre aplicação de algoritmos evolutivos multiobjetivos para o problema de integração e teste de classes e aspectos:

1. ASSUNCAO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. . A Multi-Objective Optimization Approach for the Integration and Test Order Problem. *Information Sciences*, v. 267, p. 119-139, 2014. http://dx.doi.org/10.1016/j.ins.2013.12.040.

2. ASSUNCAO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. . Evaluating Different Strategies for Integration Testing of Aspect-Oriented Programs. *Journal of The Brazilian Computer Society (Online)*, v. 20 (9), 2014. http://dx.doi.org/10.1186/1678-4804-20-9.

3. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. Generating Integration Test Orders for Aspect Oriented Software with Multi-objective Algorithms. *Revista de Informática Teórica e Aplicada (RITA)*. v. 20, p. 301-327, 2013.

4. ASSUNCAO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. On the Application of the Multi-Evolutionary and Coupling-Based Approach with Different Aspect-Class Integration Testing Strategies. In: *5th Symposium on Search-Based Software Engineering (SSBSE 2013)*, Saint Petersburg, Russia. 2013.

5. ASSUNCAO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R.. Determining Integration and Test Orders in the Presence of Modularization Restrictions. In: *Simpósio Brasileiro de Engenharia de Software (SBES)*, Brasília. 2013.

6. ASSUNCAO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. . Evaluating Different Strategies for Integration Testing of Aspect-Oriented Programs. In: *Latin American*

Workshop on Aspect-Oriented Software Development (LA-WASP) (Advanced Modularization Techniques), Natal. 2012.

7. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; POZO, A. T. R.; VERGILIO, S. R. . Establishing Integration Test Orders of Classes with Several Coupling Measures. In: *2011 Genetic and Evolutionary Computation Conference (GECCO)*, Dublin, Ireland. ACM, 2011. p. 1867-1874.

8. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; VERGILIO, S. R.; POZO, A. T. R. Estabelecendo Sequências de Teste de Integração de Classes: Um Estudo Comparativo da Aplicação de Três Algoritmos Evolutivos Multiobjetivos. In: *XXIX Simpósio Brasileira de Redes de Computadores e Sistemas Distribuídos (SBRC) - XII Workshop de Testes e Tolerância a Falhas (WTF)*, Campo Grande - MT. 2011.

9. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; POZO, A. T. R.; VERGILIO, S. R. Reduzindo o Custo do Teste de Integração com Algoritmos Evolutivos Multiobjetivos e Diferentes Medidas de Acoplamento. In: *XXXI Congresso da Sociedade Brasileira de Computação - VIII Encontro Nacional de Inteligência Artificial (ENIA)*, Natal. 2011.

10. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; VERGILIO, S. R.; POZO, A. T. R. Integration Test of Classes and Aspects with a Multi-Evolutionary and Coupling-Based Approach. In: *International Symposium on Search Based Software Engineering (SSBSE'2011)*, Szeged / Hungria. 2011.

11. COLANZI, T. E.; ASSUNÇÃO, W. K. G.; VERGILIO, S. R.; POZO, A. T. R. Generating Integration Test Orders for Aspect-Oriented Software with Multi-objective Algorithms. In: *V Workshop Latino-Americano em Desenvolvimento de Software Orientado a Aspectos (LA-WASP'2011)*, São Paulo - SP. 2011.

12. ASSUNÇÃO, W. K. G.; COLANZI, T. E.; POZO, A. T. R.; VERGILIO, S. R. Uma Avaliação do Uso de Diferentes Algoritmos Evolutivos Multiobjetivos para Integração de Classes e Aspectos. In: *II Workshop de Engenharia de Software Baseada em Buscas (WESB'2011)*, São Paulo - SP. 2011.

Estudos envolvendo métricas arquiteturais e linhas de produto de software:

1. OIZUMI, W. N.; CONTIERI JUNIOR, A. C.; CORREIA, G. G.; COLANZI, T. E.; FERRARI, S.; GIMENES, I. M. S.; OLIVEIRA JUNIOR, E. A.; GARCIA, A. F.; MASIERO, P. C.. On the Proactive Design of Product-Line Architectures with Aspects: An Exploratory Study. In: *2012 IEEE 36th Annual Computer Software and Applications Conference COMPSAC 2012*, Izmir. 2012. p. 273-278.

2. CONTIERI JUNIOR, A. C.; CORREIA, G. G.; COLANZI, T. E.; GIMENES, I.; OLIVEIRA JUNIOR, E. A.; MASIERO, P. C.; GARCIA, A. F. . Extending UML Components to Develop Software Product-Line Architectures: Lessons Learned. In: *5th European Conference on Software Architecture (ECSA)*, Essen / Alemanha. 2011.

3. OIZUMI, W. N.; COLANZI, T. E. Automatização da aplicação de métricas arquiteturais convencionais e sensíveis a interesses. In: *9o. Fórum de Informática e Tecnologia de Maringá (FITEM)*, Maringá - PR. 2010. p. 6-18.

4. MORAES, A. L. S.; BRITO, R. C.; CONTIERI JUNIOR, A. C.; RAMOS, M. C.; COLANZI, T. E.; MASIERO, P. C.; GIMENES, I. Using Aspects and the Spring Framework to Implement Variabilities in a Software Product Line. In: *XXIX International Conference of the Chilean Computer Science Society (SCCC 2010)*, Antofagasta/Chile. 2010. p. 71-80.

Estudos sobre a Comunidade Brasileira de *Search Based Software Engineering*:

1. ASSUNCAO, W. K. G.; BARROS, M. O.; COLANZI, T. E.; DIAS NETO, A. C.; PAIXAO, M. H. E.; SOUZA, J. T.; VERGILIO, S. R. . A mapping study of the Brazilian SBSE community. *Journal of Software Engineering Research and Development*, 2014, v. 2:3, 2014. http://dx.doi.org/10.1186/2195-1721-2-3.

2. COLANZI, T. E.; VERGILIO, S. R.; ASSUNÇÃO, W. K. G.; POZO, A. T. R. Search Based Software Engineering: Review and Analysis of the Field in Brazil. *Journal of Systems and Software (JSS) - Special Issue: Software Engineering in Brazil.* v. 86, p. 970-984, 2013.

3. ASSUNCAO, W. K. G.; BARROS, M. O.; COLANZI, T. E.; DIAS NETO, A. C.; PAIXAO, M. H. E.; SOUZA, J. T.; VERGILIO, S. R. . Mapeamento da Comunidade Brasileira de SBSE. In: *IV Workshop de Engenharia de Software Baseada em Busca (WESB)*, Brasília. 2013.

4. VERGILIO, S. R.; COLANZI, T. E.; POZO, A. T. R.; ASSUNCAO, W. K. G. . Search Based Software Engineering: A Review from the Brazilian Symposium on Software Engineering. In: *XXV Simpósio Brasileiro de Engenharia de Software (SBES) - Trilha Especial: SBES é 25*, São Paulo. 2011. p. 50-55.

Estudos relacionados à tese:

1. COLANZI, T. E.; VERGILIO, S. R.. A Feature-Driven Crossover Operator for Product Line Architecture Design Optimization. In: *2014 IEEE 38th Annual Computer Software and Applications Conference COMPSAC 2014*, Vasteras/Sweden. 2014.

2. COLANZI, T. E.; VERGILIO, S. R.. Representation of Software Product Line Architectures for Search-Based Design. In: *1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, San Francisco. Proceeding of the 35th International Conference on Software Engineering (ICSE), San Francisco, CA - USA. p. 28-33. 2013.

3. GUIZZO, G.; COLANZI, T. E.; VERGILIO, S. R.. Applying design patterns in product line search-based design: feasibility analysis and implementation aspects. In: *XXXII International Conference of the Chilean Society of Computer Science*, Temuco - Chile. 2013.

4. GUIZZO, G.; COLANZI, T. E.; VERGILIO, S. R.. Otimizando arquiteturas de LPS: uma proposta de operador de mutação para a aplicação automática de padrões de projeto. In: *IV Workshop de Engenharia de Software Baseada em Busca (WESB)*, Brasília. 2013.

5. COLANZI, T. E.; VERGILIO, S. R.. Direções de Pesquisa para a Otimização de Arquiteturas de Linhas de Produto de Software Baseada em Busca. In: *Workshop de Engenharia de Software Baseada em Buscas (WESB 2012)*, Natal. 2012.

6. COLANZI, T. E.. Search Based Design of Software Product Lines Architectures. In: Doctoral Symposium of the *34th International Conference on Software Engineering (ICSE)*, Zurich - Switzerland. p. 1507 - 1510. 2012.

7. COLANZI, T. E.; VERGILIO, S. R.. Applying Search Based Optimization to Software Product Line Architectures: Lessons Learned. In: *4th Symposium on Search Based Software Engineering (SSBSE)*, Riva del Garda - Italy. p. 259-266. 2012.

RESUMO

A indústria de software tem adotado a abordagem de Linha de Produto de Software (LPS) com o objetivo de aumentar o reúso de software e diminuir o tempo de produção e os custos de desenvolvimento dos produtos. Nessa abordagem, o principal artefato é a arquitetura de LPS (PLA - *Product Line Architecture*). No entanto, obter uma PLA modular, extensível e reusável é uma tarefa não trivial. O arquiteto pode se apoiar em métricas arquiteturais para definir e melhorar o projeto da PLA. Contudo, essa tarefa pode envolver vários fatores, muitas vezes conflitantes entre si, e encontrar o melhor *trade-off* entre as métricas utilizadas para avaliar o projeto transforma o projeto de PLA em uma tarefa que demanda grande esforço humano. Nesse contexto, o projeto de PLA pode ser formulado como um problema de otimização com vários fatores. Porém, elaborar um projeto que atenda a todos os fatores envolvidos pode ser mais difícil do que reconhecer um bom projeto. Problemas da Engenharia de Software similares a esse têm sido eficientemente resolvidos com algoritmos de busca em um campo de pesquisa conhecido como Engenharia de Software Baseada em Busca (SBSE - *Search Based Software Engineering*). Entretanto, as abordagens existentes utilizadas para otimizar arquiteturas de software não são apropriadas para projeto de PLAs, pois não consideram características específicas de LPS. Desse modo, este trabalho propõe uma abordagem de otimização multiobjetivo automatizada para avaliar e melhorar um projeto de PLA no que tange a modularização de características, estabilidade do projeto e extensibilidade de LPS. A abordagem proposta inclui: (a) um processo sistemático para conduzir a otimização de projeto de PLA por meio de algoritmos de busca; (b) um metamodelo que permite que esses algoritmos manipulem projetos de PLA; (c) novos operadores de busca para evoluir projetos de PLA em termos de modularização de características; e (d) um tratamento multiobjetivo para o problema de projeto de PLA. Esse tratamento multiobjetivo engloba métricas que indicam a modularização de características e a extensibilidade de LPS, além de métricas convencionais para medir princípios básicos de projeto como coesão e acoplamento. Ao final do processo de otimização, um conjunto de possíveis soluções de projeto de PLA que representam os melhores *trade-off* entre os objetivos otimizados é retornado. O arquiteto deve selecionar uma solução de acordo com as suas prioridades. A ferramenta OPLA-Tool foi desenvolvida para instanciar a abordagem usando algoritmos evolutivos multiobjetivos, os quais têm sido usados com sucesso na área de SBSE. Utilizando a OPLA-Tool, quatro estudos empíricos foram realizados com nove PLAs para avaliar: os operadores de busca propostos; o uso das métricas de LPS; e os algoritmos escolhidos. Em comparação às PLAs originais, os resultados mostraram que a abordagem proposta consegue gerar projetos mais estáveis, mais elegantes e com melhor modularização de características.

ABSTRACT

The Software Product Line (SPL) approach has been adopted by the software industry since it increases the software reuse and decreases both the time to market and the development costs. The Product-Line Architecture (PLA) is the main artefact of a SPL. However, obtaining a modular, extensible and reusable PLA is a non-trivial task. SPL architects may rely on architectural metrics to define and improve a PLA design. But, this task is often related to different and possible conflicting factors. Finding the best trade-off between the metrics values used to evaluate the design turns PLA design in a people-intensive task. Hence, the PLA design can be formulated as an optimization problem with many factors. Although developing a PLA design that encompasses the involved factors may be more difficult than recognizing a good design. Similar Software Engineering problems have been efficiently solved by search-based algorithms in the field known as Search Based Software Engineering (SBSE). However, existing approaches used to optimize software architectures are not suitable to PLA design since they do not encompass specific characteristics of SPL. In this sense, this thesis introduces a multi-objective optimization approach to automate the evaluation and improvement of PLA design in terms of feature modularization, design stability and SPL extensibility. This approach (a) encompasses a process to conduct the PLA design optimization through search-based algorithms; (b) includes a metamodel to allow the PLA manipulation by search-based algorithms; (c) addresses the feature modularization by novel search operators; and (d) introduces a multi-objective treatment to the PLA design problem. The multi-objective treatment includes feature-driven and SPL extensibility metrics in addition to conventional metrics to measure basic design principles like coupling and cohesion. After the optimization process, a set of potential solutions that represent the best trade-off between the objectives to be optimized is returned. So, the architect should choose the best solution according to his or her priorities. OPLA-Tool was developed to allow the approach instantiation by using multi-objective evolutionary algorithms, which have been successfully used in the SBSE area. Four empirical studies were performed with nine PLA designs to evaluate: the proposed search operators, SPL metrics, and the chosen search-based algorithms. The results have shown the proposed approach can generate PLA designs more stable, more elegant and with better feature modularization than the original ones.

CAPÍTULO 1

INTRODUÇÃO

1.1 Contextualização e Descrição do Problema

Linha de Produto de Software (LPS) é uma abordagem sistemática para a reutilização de software aplicada a uma família de produtos dentro de um domínio bem definido [96]. As organizações têm adotado a abordagem de LPS a fim de migrar do reúso de componentes individuais para o reúso de software de uma arquitetura de LPS em larga escala. Um dos principais ativos para garantir o reúso em uma LPS é a arquitetura da LPS (PLA - *Product Line Architecture*) porque ela engloba um projeto que é comum a todos os produtos derivados da LPS. Para isso, uma PLA deve abranger os componentes que realizam as características obrigatórias e variáveis em um domínio [23]. As características variáveis (variabilidades) incluem pontos de variação, os quais estão associados às diferentes alternativas de projeto daquela variabilidade [96].

Arquitetos geralmente raciocinam sobre uma LPS em termos de características, as quais têm uma importância crucial na abordagem de LPS. Uma característica (do inglês *feature*) é uma capacidade do sistema que é relevante e visível para o usuário final [48]. Uma condição essencial para o projeto bem sucedido de uma PLA é a identificação de elementos arquiteturais que permitam a melhor modularização das características da LPS. Caso contrário, a PLA fica suscetível a revisões precoces e grandes refatorações desde o início do projeto, o que pode atrasar o processo de desenvolvimento. Uma arquitetura de LPS não modular também poderá sofrer alterações quando novos produtos precisarem ser acomodados, tornando-se difícil manter a estabilidade do projeto ao longo do tempo, já que um projeto é dito estável se os seus módulos se mantêm estáveis ao longo do tempo sem precisar de alterações [62].

A modularidade consiste na divisão de um software em componentes nomeados separadamente e endereçáveis [73], o que em outras palavras significa dividir o software em

módulos que realizam uma função completa independente de outras funções. Dado esse conceito, a alta modularização de características implica em baixa difusão das funcionalidades de uma mesma feature pelo projeto da PLA. Esta modularização também outros tópicos relacionados a modularidade, tais como: coesão, acoplamento e entrelaçamento de características [24]. No entanto, características são raramente modularizadas em PLAs [59]. É difícil obter a modularização de características porque o código específico de características está frequentemente disperso pelas classes de fronteira - aquelas que propiciam a comunicação tanto entre componentes do software como entre os componentes e os atores. Apesar disso, é natural considerar a modularização de características como uma forma de modularizar os programas [59].

Outras propriedades particularmente importantes para projeto de PLA são reusabilidade e extensibilidade da PLA. A extensibilidade é medida em termos da abstração da PLA, uma PLA extensível permite a geração de um número maior de produtos para a LPS, porque um alto nível de abstração propicia a inserção de novas características, facilitando sua evolução. A extensibilidade também influencia na reusabilidade e na longevidade da PLA. Quanto mais extensíveis forem os componentes da arquitetura, maior sua taxa de reusabilidade [69].

Além da maximização de propriedades como modularidade, reusabilidade, extensibilidade e estabilidade, a presença de pontos de variação inter-relacionados dificulta a especificação inicial da PLA assim como a sua subsequente evolução [58]. Por isso, a definição e a avaliação do projeto de PLA é uma atividade importante ao longo do ciclo de vida da LPS [30, 70] e é ainda mais complexa do que o projeto de software único.

Além de métodos para a análise qualitativa de PLAs, estudos sobre avaliação estrutural de PLA frequentemente consideram um conjunto de métricas para fornecer indicadores sobre diferentes propriedades [70, 4, 13]. Métricas podem auxiliar na detecção de pontos do projeto que são propícios a apresentarem problemas - os chamados *bad smells*. Um dos *bad smells* arquiteturais que impacta negativamente a reusabilidade e extensibilidade da PLA é o *Scattered Functionality* [36]. Porém, para melhorar o projeto e identificar possíveis *bad smells*, arquitetos precisam analisar uma grande quantidade de valores de métricas.

Além das métricas convencionais para projeto de software que fornecem indicadores de coesão e acoplamento dos componentes arquiteturais [11, 62], há ainda métricas para medir atributos de qualidade e avaliar PLAs [13, 60, 69, 76, 106]. Ainda assim, outras variáveis podem influenciar um projeto de PLA, como fatores econômicos, restrições de tempo, complexidade dos produtos da LPS, dentre outros.

Então, encontrar o melhor balanceamento de prioridades (*trade-offs*) entre as métricas utilizadas transforma o projeto de PLA em uma tarefa que demanda grande esforço por parte dos arquitetos. Além disso, algumas métricas podem estar em conflito e os arquitetos precisam escolher quais devem ser priorizadas. Por exemplo, muitas vezes, aumentar a modularidade das características da LPS implica em aumentar o acoplamento da PLA. Portanto, apesar da existência de métricas para apoiar a definição e avaliação de uma arquitetura de software, o projeto de PLA é, em geral, uma tarefa difícil, para a qual ainda não existe uma solução ótima e exata. Neste contexto, reconhecer um bom projeto pode ser fácil para arquitetos, mas por outro lado, é difícil obtê-lo.

A busca da solução para problemas difíceis tem sido tratada no campo de pesquisa denominado Engenharia de Software Baseada em Busca (*Search Based Software Engineering* - SBSE) [22, 42]. Técnicas aplicadas na área de SBSE têm provado sua efetividade em descobrir automaticamente soluções próximas das ótimas para este tipo de problemas da Engenharia de Software [2, 41, 77]. Em SBSE, os problemas de Engenharia de Software são formulados como problemas de otimização a serem resolvidos por meio de técnicas baseadas em busca, que são derivadas das áreas de pesquisa operacional, como programação linear, e meta-heurísticas, tais como algoritmos evolutivos, otimização por nuvem de partículas (*Particle Swarm Optimization* - PSO) e otimização por colônia de formigas (*Ant Colony Optimization* - ACO).

Abordagens baseadas em Algoritmos Evolutivos Multiobjetivos (*Multi-Objective Evolutionary Algorithms* - MOEAs) têm alcançado resultados promissores, pois permitem considerar diferentes fatores e medidas que afetam o problema [6, 10, 72]. Muitas vezes, esses fatores estão em conflito e, geralmente, não há uma única solução. Problemas de otimização com duas ou mais funções objetivo são chamados de multiobjetivos. A

idéia é encontrar um conjunto de soluções que melhor representem o possível compromisso entre os objetivos. Na literatura existem diversos trabalhos que utilizam algoritmos baseados em busca, incluindo multiobjetivos, para resolver problemas de Engenharia de Software [2, 22, 41, 77].

As técnicas aplicadas em SBSE podem fornecer apoio automatizado para avaliar possíveis alternativas para o projeto de uma PLA. Essas técnicas podem ser usadas para melhorar o projeto de uma PLA fornecida como entrada. A avaliação e a melhoria automatizada podem apoiar as decisões dos arquitetos de LPS. Apesar disso, não há trabalhos prévios que otimizam o projeto de PLA usando técnicas de SBSE. Há poucos estudos que empregam técnicas de SBSE no desenvolvimento de LPS e, geralmente, focam na automatização da configuração de produtos e na seleção de características para desenvolver e evoluir LPS [85, 49]. Então, o alvo de otimização desses estudos não é o projeto de PLA.

Trabalhos que focam em otimização de projeto de software usando técnicas de busca estão mais relacionados com otimização de projeto de PLA. Abordagens baseadas em MOEAs têm alcançado resultados promissores para otimizar o projeto de software ou otimizar a ordem de refatorações a serem aplicadas ao projeto [77]. Os trabalhos que focam a otimização do projeto de software tratam dois problemas: a aplicação de padrões de projeto ao projeto de software [82] e a associação de responsabilidades às classes [10, 89]. Neste último problema, por exemplo, operadores de busca são aplicados usando MOEAs para evoluir o projeto por meio da movimentação de atributos e métodos entre classes ou da adição de novas classes. As soluções alcançadas são avaliadas por meio de métricas de coesão e acoplamento.

Como o projeto de PLA faz parte do projeto de software, ambos são afetados por fatores conflitantes e alguns fatores são os mesmos para os dois casos, por exemplo, quando se aumenta a coesão dos componentes aumenta-se também o acoplamento entre eles. Entretanto, o projeto de PLA apresenta algumas especificidades que não são consideradas nos trabalhos de projeto de software baseado em busca como, por exemplo, a presença de variabilidades e de características associadas aos elementos arquiteturais, a modularização de características, a extensibilidade do projeto da PLA, e assim por diante.

Algumas questões pertinentes ao projeto de PLA que as abordagens de projeto de software baseado em busca não consideram são, por exemplo, (i) que a melhoria na modularização de características gera impactos na coesão e no acoplamento de componentes e, (ii) que nem sempre é possível alterar a estrutura de classes da PLA visando a melhorar sua extensibilidade devido a presença dos pontos de variação.

Diante do exposto, a hipótese deste trabalho é que a otimização multiobjetivo pode contribuir para melhorar o projeto de arquiteturas de LPS, uma vez que por meio da otimização multiobjetivo pode-se melhorar o espaço de soluções de PLA. Uma abordagem sistemática e automatizada contribui para diminuir o esforço dos arquitetos durante o projeto e avaliação de uma PLA porque eles podem fornecem um projeto de PLA inicial e depois interagir com a abordagem para selecionar uma das soluções geradas pela abordagem, conforme as necessidades organizacionais ou da LPS.

1.2 Justificativas

Dado o contexto exposto que reflete o estado da arte em projeto de PLA e do campo de pesquisa de SBSE, têm-se os seguintes pontos que justificam o presente trabalho:

- LPS é uma abordagem que vem sendo cada vez mais utilizada pelas empresas de software para facilitar o reúso e diminuir o custo de desenvolvimento de seus produtos [91].

- A definição de uma PLA é uma tarefa fundamental para a geração dos produtos da LPS e para facilitar seu gerenciamento. Para isso, são utilizadas técnicas tais como *frameworks*, padrões de projeto, estilos arquiteturais e desenvolvimento baseado em componentes, visando a melhorar a modularidade e reusabilidade da PLA.

- A avaliação de uma LPS é uma atividade não trivial, que envolve vários fatores. No contexto de LPS, a avaliação de modularização de características e extensibilidade de LPS requer um conjunto específico de métricas, tema ainda pouco estudado na literatura.

- Processos e ferramentas específicas para racionalizar a tomada de decisão quanto ao projeto de arquitetura de software são relativamente raros e imprescindíveis para apoiar o arquiteto nesta atividade [30].

- A utilização de algoritmos de busca multiobjetivos foi explorada com sucesso no projeto detalhado de classes.

- Entende-se que o projeto de PLA também pode ser tratado como um problema multiobjetivo. Existem trabalhos nos quais técnicas de SBSE são aplicadas no projeto de software, contudo, tais trabalhos não envolvem o projeto de PLA nem o uso de métricas específicas para LPS.

1.3 Objetivos

Considerando o contexto descrito anteriormente e a hipótese apresentada no final da Seção 1.1, investiga-se neste trabalho se as técnicas de otimização multiobjetivo podem contribuir para melhorar o projeto de PLAs e, por consequência, apoiar os arquitetos de LPS nessa atividade.

Baseando-se nesse tópico de pesquisa, o principal objetivo deste trabalho é propor uma abordagem de otimização multiobjetivo para identificar automaticamente as melhores alternativas de projeto para uma PLA. Em decorrência deste, os objetivos específicos do trabalho incluem:

- Identificar um conjunto de métricas arquiteturais que pode ser empregado na avaliação de PLAs no contexto da abordagem proposta;

- Identificar qual a melhor representação para o problema de projeto de PLA;

- Definir operadores de busca para otimizar projetos de PLA;

- Automatizar a abordagem proposta;

- Realizar experimentos de avaliação da abordagem proposta.

1.4 Metodologia de Pesquisa

A metodologia de pesquisa utilizada para cumprir os objetivos do presente trabalho envolve desde atividades de pesquisa bibliográfica até atividades experimentais. Essa metodologia consiste das atividades relacionadas a seguir:

- Revisão bibliográfica sobre métricas arquiteturais e métricas específicas para LPS.
- Estudo da viabilidade de aplicação de abordagens existentes de projeto de software baseado em busca no contexto de projeto de PLA.
- Definição de uma abordagem sistemática e automatizada para a otimização multiobjetivo de projeto de PLA. A definição dessa abordagem envolve todos os aspectos envolvidos com otimização multiobjetivo que incluem: a definição de uma representação para projeto de PLA; a descrição do processo de otimização; a adaptação e/ou proposta de operadores de busca para o problema sendo tratado; e, a proposta do modelo de avaliação que inclui as métricas arquiteturais.
- Estudo de algoritmos multiobjetivos e instanciação da abordagem proposta usando alguns dos algoritmos estudados. Essa atividade envolve a definição de quais algoritmos multiobjetivos utilizar na instanciação da abordagem proposta e de como adaptá-los para o contexto da abordagem. Na sequência, deve-se implementar esses algoritmos e todas as características da abordagem de forma a torná-la operacional.
- Avaliação experimental da abordagem proposta. No contexto deste trabalho, estudos empíricos permitem avaliar: (i) a viabilidade de uso dos operadores de busca propostos, (ii) a eficiência das métricas em medir os projetos de PLA alcançados e (iii) o tratamento multiobjetivo dado ao problema. Ademais, eles podem fornecer evidência empírica de que a abordagem proposta melhora os projetos de PLA de acordo com os objetivos traçados.

Na MOA4PLA, uma visão estreita de arquitetura de LPS foi adotada como ponto de partida, considerando apenas o aspecto estrutural estático, expressível em um diagrama

de classes UML estereotipado, o que corresponde à visão lógica (parcial) considerando o modelo de visões arquiteturais $4 + 1$ [54] ou à visão de desenvolvimento definida em [96]. Abordagens semelhantes poderiam ser aplicadas para gerar outros pontos de vista da PLA, porém há algumas limitações fundamentais na utilização de métodos heurísticos. Por exemplo, seria difícil para uma técnica de busca produzir o *rationale* para as decisões de concepção tomadas. Além disso, a MOA4PLA toma como alvo PLAs baseadas em componentes.

1.5 Organização do texto

Esta tese está organizada em seis scapítulos. No primeiro capítulo foram abordados o contexto, a justificativa e os objetivos do trabalho. No Capítulo 2 são apresentados os principais conceitos a respeito de LPS, PLA e métricas para LPS. O Capítulo 3 envolve o campo de SBSE, mais especificamente *search based design*, abordando algoritmos de otimização multiobjetivos e os trabalhos relacionados ao tema desta tese. Na sequência, no Capítulo 4 é apresentada a abordagem de otimização de projeto de PLA proposta e são discutidos alguns aspectos de implementação. O Capítulo 5 contém a descrição e os resultados dos estudos empíricos realizados para a avaliação da abordagem proposta. Esse capítulo contém ainda uma explanação acerca de como o arquiteto pode selecionar uma das soluções geradas pela MOA4PLA. O Capítulo 6 apresenta as conclusões do trabalho, suas contribuições e relaciona possíveis trabalhos futuros.

A tese também possui cinco apêndices. O Apêndice A contém o artigo publicado que trata das lições aprendidas quando da aplicação de abordagens de projeto de software baseado em busca ao projeto de PLA. O Apêndice B contém o artigo que relata o estudo realizado para determinar o tipo de representação a ser adotado na MOA4PLA. Os Apêndices C e D apresentam dois artigos submetidos referentes aos Estudos 1 e 2 (Capítulo 5). Os resultados desses dois estudos são apresentados no Capítulo 5 de forma mais sintetizada do que nos artigos inseridos em seus respectivos apêndices. Por fim, o Apêndice E apresenta alguns resultados do Estudo 4 que não foram incluídos no corpo da tese por serem complementares aos dados apresentados no Capítulo 5.

CAPÍTULO 2

LINHA DE PRODUTO DE SOFTWARE

Uma LPS representa um conjunto de sistemas, que compartilham características comuns que satisfazem as necessidades de um determinado segmento de mercado ou domínio [96]. Assim, o desenvolvimento de uma LPS visa à construção de componentes que podem ser reutilizados em sistemas do mesmo domínio, e à identificação de variabilidades que podem ser implementadas e combinadas para a formação de diferentes produtos. Para tal, é necessário que ambos aspectos sejam planejados para todo o ciclo de desenvolvimento de modo que possam ser centralizados em uma estrutura comum, denominada de núcleo de artefatos (*core assets*), facilitando a manutenção e reutilização de artefatos. Assim, por meio desta abordagem, as organizações podem explorar as semelhanças dos seus produtos para aumentar o reúso de artefatos. A definição explícita de variabilidades em LPS é a diferença chave entre o desenvolvimento de sistemas únicos e o desenvolvimento de LPS. Nos últimos anos, a abordagem de LPS tem sido adotada cada vez mais na indústria de software que mantém LPSs que focam em diferentes tipos de produtos, segmentos de mercado e/ou domínios. Na literatura encontram-se alguns exemplos de empresas que têm obtido sucesso ao adotar esta abordagem, tais como: Philips, Bosch, Boeing, HP, Nokia e Toshiba [91].

A base de uma LPS é seu núcleo de artefatos, o qual engloba a arquitetura da LPS, os componentes reusáveis, modelos de domínio, requisitos da LPS, modelos de características e de variabilidades e planos de teste. O modelo de características inclui todas as características de uma LPS e os seus inter-relacionamentos. Uma característica (do inglês *feature*) é uma capacidade do sistema que é relevante e visível para o usuário final [48]. Uma característica pode ser obrigatória, opcional ou alternativa. As características obrigatórias estão presentes em todos os produtos da LPS, as opcionais são utilizadas somente

por alguns produtos e as alternativas constituem um conjunto de características relacionadas, a partir do qual é possível escolher apenas uma, logo são mutuamente exclusivas.

O modelo de características representa as variabilidades de uma LPS. Variabilidades são descritas em termos de pontos de variação e variantes [38]. Um ponto de variação é um lugar específico em um artefato da LPS ao qual uma decisão de projeto está associada. Cada ponto de variação está associado a um conjunto de variantes que correspondem às alternativas de projeto para uma variabilidade [96]. As variabilidades podem estar associadas a diferentes níveis de abstração, tais como, a descrição da arquitetura, a documentação de projeto, o código fonte e o código compilado. A variabilidade é o ponto chave de uma LPS, uma vez que sua representação explícita torna possível a geração de produtos específicos de uma LPS.

A variabilidade surge do adiamento de certas decisões fundamentais ao projeto de produtos de software. Essas decisões, quando tomadas no início do projeto, restringem o domínio no qual o sistema pode ser aplicado [16]. Assim, quanto maior o número de decisões de projeto adiadas, maior será o número de variabilidades de um produto de software [67].

Dentre as motivações que levam a indústria a adotar o desenvolvimento de LPS estão [96]:

- redução de custos no desenvolvimento de software devido ao reúso;
- melhoria da qualidade dos produtos já que os artefatos reusados são revisados e testados em vários produtos;
- redução do tempo de produção (*time to market*) que inicialmente é alto devido ao desenvolvimento do núcleo da LPS, mas que diminui consideravelmente quando da produção de cada novo produto;
- redução do esforço de manutenção, pois sempre que um artefato do núcleo passar por uma manutenção, as mudanças podem ser propagadas para os produtos que utilizam tal artefato;

- facilidade para a evolução, pois quando um novo artefato é inserido no núcleo da LPS, os produtos da LPS podem se beneficiar dessa evolução;

- contribuição para a redução da complexidade, porque apesar de ter produtos cada vez mais complexos em virtude do crescente número de solicitações de clientes, a complexidade é mais facilmente gerenciada e, por isso, diminui já que na abordagem de LPS, o núcleo deve fornecer uma estrutura a fim de determinar os lugares em que os componentes podem ser reutilizados por meio da definição de variabilidades em diferentes pontos;

- melhoria na estimativa de custo porque não havendo riscos fica mais fácil estimar o custo de produtos que serão gerados a partir do reúso de artefatos do núcleo e produtos que exijam extensões podem ser vendidos por um valor mais elevado; e

- benefícios para os clientes que obtêm produtos adaptados às suas necessidades por um preço mais acessível devido à redução de custo, viabilizada pela abordagem de LPS.

Independentemente do processo de engenharia de LPS adotado, existem três diferentes abordagens para desenvolvimento de LPS: proativa, reativa e incremental [46, 55].

Na abordagem proativa o núcleo de artefatos é desenvolvido primeiro. Na sequência, os produtos são desenvolvidos exigindo mínimo esforço para escrever código e chegando rapidamente ao mercado. Esta abordagem requer investimento inicial e conhecimento prévio sobre o domínio e os produtos a serem gerados. No contexto de engenharia de LPS proativa, projetos modulares de PLAs são mais difíceis de se alcançar uma vez que os desenvolvedores muitas vezes não podem se basear em arquiteturas existentes [96].

Por outro lado a abordagem reativa inicia com um ou mais produtos. A partir dos quais é gerado o núcleo de artefatos e os futuros produtos. Em alguns casos, o custo de adoção de LPS é mais baixo, porém a evolução do escopo é mais difícil. A arquitetura e outros artefatos do núcleo devem ser robustos, extensíveis e preparados para futuras necessidades da LPS.

Tanto na abordagem proativa como na reativa é possível desenvolver o núcleo de artefatos em estágios planejando desde o início o desenvolvimento da LPS, isso constitui a abordagem incremental. Por exemplo, desenvolver parte do núcleo de artefatos incluindo a arquitetura e alguns componentes, desenvolver um ou mais produtos, desenvolver parte do resto do núcleo de artefatos, desenvolver mais produtos, evoluir o núcleo de artefatos e assim por diante.

2.1 Atividades Essenciais de Linha de Produto de Software

A organização de uma LPS engloba três atividades essenciais [46, 96]: desenvolvimento do núcleo de artefatos, no qual é desenvolvido o núcleo de artefatos; desenvolvimento do produto, em que os produtos da LPS são gerados e, gerenciamento da LPS (Figura 2.1). O núcleo de artefatos é constituído dos recursos comuns à LPS e um desses principais recursos é a PLA (abordada na Seção 2.3), a qual é utilizada para derivar a arquitetura de cada produto da LPS. A atividade de desenvolvimento do núcleo de artefatos pode ser chamada de engenharia de domínio, assim como a atividade de desenvolvimento do produto pode ser chamada de engenharia de aplicação.

Figura 2.1: Atividades essenciais de LPS (adaptada de [46])

O desenvolvimento do núcleo de artefatos corresponde a um ciclo iterativo e incremental, isto é, à medida que novos produtos são desenvolvidos, suas características são

incorporadas ao domínio da LPS. A atividade de desenvolvimento do produto corresponde à geração dos produtos específicos e tem como propósito explorar rapidamente os requisitos e construir a arquitetura de um determinado produto a partir da PLA para, então, produzir tal aplicação. O desenvolvimento do produto é realizado por meio da seleção das características desejadas e implementadas na atividade anterior, ligando-as às características comuns a todos os produtos. Finalmente, o gerenciamento da LPS corresponde à manutenção e evolução da LPS. Nesta atividade é importante o controle das atividades já desenvolvidas e o planejamento de novas funcionalidades ou melhorias dos artefatos já construídos.

O foco desta tese é o projeto da arquitetura de uma LPS (descrita na Seção 2.3) que é produzida na atividade de desenvolvimento do núcleo de artefatos. Mais detalhes a respeito de cada atividade podem ser obtidos em [46]. Como mencionado anteriormente as variabilidades são um ponto chave em uma LPS e, para permitir a geração de produtos específicos é preciso representá-las nos artefatos da LPS. Uma abordagem para representação de variabilidades é descrita na próxima seção.

2.2 Abordagem SMarty para representar variabilidades de LPS

SMarty [68] é uma abordagem de gerenciamento de variabilidades em LPS baseada em UML. Essa abordagem utiliza estereótipos específicos para representar cada tipo de variante, pontos de variação e variabilidades em diagramas UML. SMarty estabelece um perfil UML (*SMartyProfile*) e um processo sistemático de gerenciamento de variabilidades (*SMartyProcess*). O *SMartyProfile* [68] (apresentado na Figura 2.2) foi utilizado no desenvolvimento deste trabalho, mais especificamente na representação de variabilidades nos diagramas de classes UML que representam projetos de PLA. Para isso foram utilizados os seguintes estereótipos:

- <<variability>> – Representa o conceito de variabilidade de LPS e é uma extensão da meta-classe *Comment* da UML. Nesse comentário são definidos os atributos de caracterização da variabilidade como: nome da variabilidade; quantidade mínima e

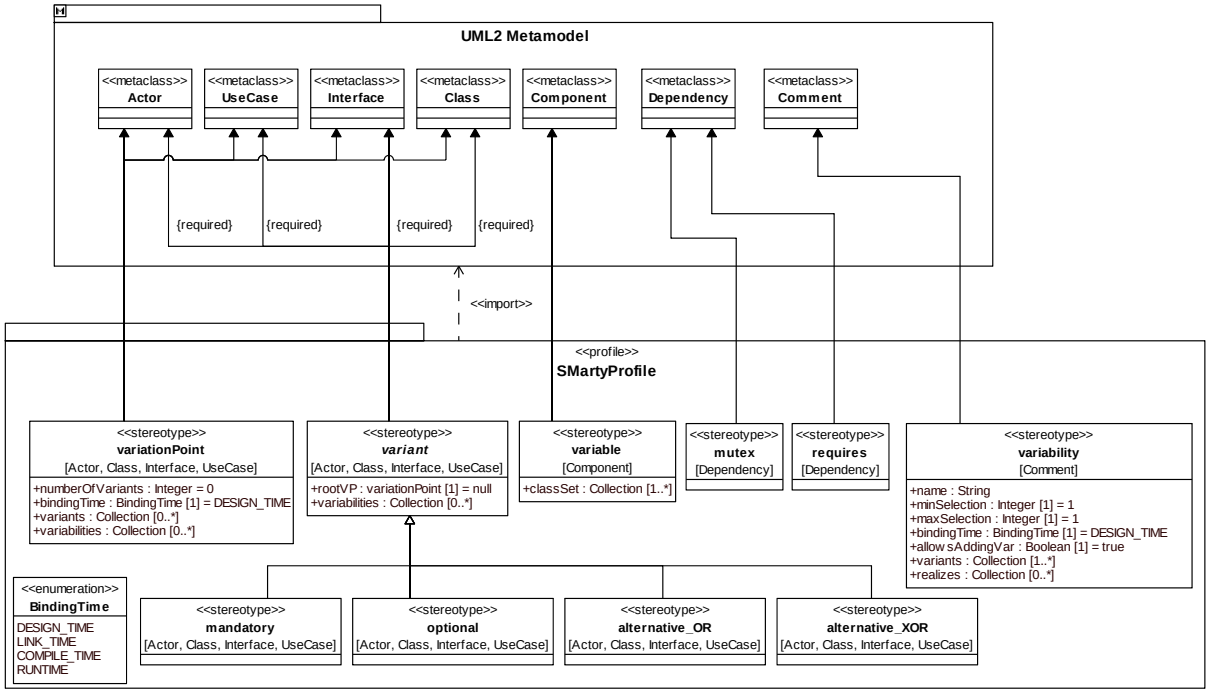


Figura 2.2: SMartyProfile [68]

máxima de variantes que devem ser selecionadas; momento em que a variabilidade deve ser solucionada; variantes disponíveis; se é possível incluir novas variantes após realizar uma variabilidade; e, as variabilidades de menor nível que realizam essa variabilidade;

- <<variationPoint>> – Representa o conceito de ponto de variação e é uma extensão das meta-classes *Actor*, *UseCase*, *Interface* e *Class*. Assim como o estereótipo de variabilidade, esse estereótipo possui atributos similares que o caracterizam;

- <<variant>> – Representa o conceito de variante e é uma extensão das meta-classes *Actor*, *UseCase*, *Interface* e *Class*. Além de possuir atributos que apontam sua variabilidade e ponto de variação, este estereótipo é especializado em 4 outros estereótipos não-abstratos: <<mandatory>>, <<optional>>, <<alternative_OR>> e <<alternative_XOR>>;

- <<mandatory>> – Representa o conceito de variante obrigatória (que faz parte de todos os produtos em uma LPS);

- <<optional>> – Representa o conceito de variante opcional;

- <<alternative_OR>> – Representa uma variante que faz parte de um grupo de variantes alternativas e inclusivas, ou seja, podem ser combinadas entre si para solucionar a variabilidade;

- <<alternative_XOR>> – Representa uma variante que faz parte de um grupo de variantes alternativas e exclusivas, ou seja, apenas uma das variantes do grupo pode ser selecionada para solucionar a variabilidade;

- <<mutex>> – Representa o conceito de restrição de exclusão entre duas variantes, ou seja, se uma variante é selecionada, a outra não pode ser selecionada;

- <<requires>> – Representa o conceito de restrição de inclusão entre duas variantes, ou seja, a inclusão de uma variante requer a inclusão da outra variante.

2.3 Arquitetura de LPS

A essência de todo sistema de software bem projetado é uma boa arquitetura de software, isto é, um projeto baseado em um conjunto de boas decisões de projeto [92]. Uma arquitetura de software captura decisões de projeto de alto nível, incluindo a organização de componentes e a interação entre eles, bem como os princípios e diretrizes para a implementação e evolução da arquitetura. A PLA é o resultado da engenharia de domínio e descreve uma arquitetura genérica que fornece uma solução para a gama de produtos da LPS. Assim, a PLA representa a abstração de todas as possíveis arquiteturas de produtos que a LPS pode gerar. A similaridade estrutural entre as variantes, resultante da arquitetura comum, habilita os desenvolvedores a reusar componentes nos diferentes produtos da LPS [86]. Desse modo, a PLA define as partes essenciais da infraestrutura de reúso e garante que tanto os componentes compartilhados como os específicos para alguns produtos se encaixem para criar cada produto da LPS. São as variabilidades que permitem maximizar o reúso na engenharia de LPS, sendo assim, quanto mais modularizadas estiverem as variabilidades, maior a capacidade de reúso, de derivação de produtos e de

evolução da LPS. Por esses motivos, a PLA engloba desafios interessantes além daqueles de uma arquitetura de software único.

Dessa forma, a PLA torna-se um dos artefatos mais importantes para gerar produtos específicos com sucesso durante todo o ciclo de vida da LPS. Portanto, é importante analisar a arquitetura para predizer sua qualidade antes da derivação dos produtos, de modo a identificar potenciais riscos e verificar se os requisitos de qualidade estão sendo tratados no projeto.

Geralmente, a estrutura arquitetural não é documentada como uma única entidade e existem diferentes visões sobre a arquitetura, que juntas, determinam a estrutura da LPS. Algumas dessas visões são: (i) visão lógica, que incorpora os modelos de requisitos e descreve as aplicações em termos de domínio do problema; (ii) visão de desenvolvimento, que mostra a decomposição hierárquica do sistema em componentes, objetos e suas interfaces; (iii) visão de processo, que mostra a decomposição do sistema durante a execução, relacionando atividades ordenadas e seus relacionamentos; e, (iv) visão de código, que mostra a decomposição do código executável em arquivos e a atribuição de cada um desses arquivos para unidades de processamento [96]. Uma visão é representada por um ou mais diagramas e, um mesmo diagrama pode participar em mais de uma visão. Além disso, visando a proporcionar o reúso, padrões arquiteturais, estilos arquiteturais e/ou arquiteturas de referência parametrizadas costumam estar presentes no projeto de PLA [92].

Como a PLA é um dos ativos mais importantes para gerar com sucesso os produtos de uma LPS, a avaliação da PLA deve ser considerada uma atividade fundamental ao longo de um ciclo de vida da LPS [69]. Neste contexto, Etxeberria e Sagardui [32] classificam os atributos de qualidade, que devem estar presentes e que afetam diretamente uma PLA, em duas categorias: os que são observáveis durante a operação do software, como desempenho, segurança, usabilidade; e, os que não são observáveis durante a execução do software ou que são relacionados a atributos de desenvolvimento, como modificabilidade, portabilidade, reusabilidade e testabilidade.

Considerando as peculiaridades da avaliação de PLA, há dois níveis de abstração [32]: (i) arquitetura da LPS, nível no qual avalia-se a flexibilidade da PLA para acomodar todos os produtos da LPS; e (ii) arquitetura dos produtos derivados, no qual o objetivo é ter certeza de que o comportamento específico e os requisitos de qualidade do produto são atendidos.

Em uma PLA os atributos são classificados como [32]:

- atributos de qualidade da LPS: permitem que a arquitetura seja a base para os produtos relacionados ou futuros produtos. Estão relacionados à variabilidade ou flexibilidade;
- atributos relevantes do domínio: como segurança para domínio de segurança crítica, como desempenho para domínio de tempo real e, assim por diante;
- requisitos funcionais/comportamento comum: devem ser avaliados para evitar propagação de problemas.

Etxeberria e Sagardui [32] destacam os momentos de avaliação arquitetural de software usuais como sendo: durante o projeto da arquitetura (para LPS na engenharia de domínio e na engenharia de aplicação) ou durante a evolução. O primeiro momento é importante para detectar problemas e riscos ou para comparar possíveis arquiteturas candidatas a fim de selecionar uma delas. O segundo momento, para adicionar novos requisitos ou adaptar a arquitetura de referência. Especificamente para LPS, surgem três novos momentos de avaliação: antes do desenvolvimento da PLA, durante a instanciação do produto e, durante a atualização da arquitetura (para analisar como as modificações afetam as arquiteturas de produtos já existentes e o contrário, como alterações no produto alteram a arquitetura). Cabe ao arquiteto selecionar os momentos mais apropriados para realizar a avaliação da arquitetura [32].

Há dois tipos de técnicas de avaliação de LPS: técnicas de questionamento (avaliação qualitativa) e técnicas de medição (avaliação quantitativa). Atributos de qualidade de LPS costumam ser avaliados por técnicas qualitativas e, a maioria delas é baseada em cenários. Atributos relevantes do domínio podem ser avaliados por qualquer um dos dois

tipos de técnicas. Para requisitos funcionais outras técnicas automatizadas são usadas como checagem de modelos, prova de teorema, checagem de equivalência, etc.

Oliveira Junior, Maldonado e Gimenes [70] caracterizam a avaliação de LPS segundo os aspectos definidos em [32] e apresentam o panorama sobre avaliação de PLA, classificando os trabalhos em três grupos: (a) avaliação de atributos de qualidade de PLA; (b) avaliação estrutural de PLA; e, (c) definição e avaliação de escopo de LPS.

No grupo *a* predomina a análise qualitativa da PLA [70] e há vários métodos de avaliação baseados em cenários, GQM (*Goal Question Metric*), redes bayesianas e nos métodos ATAM (*Architecture Tradeoff Analysis Method*) e SAAM (*Software Architecture Analysis Method*) [50, 53, 63, 95, 106].

Os trabalhos do grupo *b* focam na avaliação estrutural de PLA, com base na definição e uso de métricas para avaliação dos componentes arquiteturais de LPS e atributos de qualidade. Esse tipo de avaliação é vista como quantitativa, devido ao uso de métricas e, também qualitativa, por relacionar as métricas coletadas com atributos de qualidade de PLAs. Esse é o grupo que tem o menor número de trabalhos [27, 76, 94].

Portanto, a avaliação estrutural de uma PLA requer um conjunto de métricas arquiteturais, o que pode evidenciar a qualidade da LPS. Ainda assim, outras variáveis podem influenciar no desenvolvimento da PLA, como fatores econômicos, restrições de tempo e complexidade dos produtos da LPS. Dessa forma, encontrar o melhor equilíbrio de prioridades a serem estabelecidas para essas variáveis não é uma tarefa trivial.

O foco da abordagem proposta nesta tese é apoiar a avaliação estrutural e o desenvolvimento do projeto de uma PLA, isto é, antes da derivação dos produtos da LPS. Algumas métricas que apoiam essa avaliação estrutural são apresentadas na próxima seção.

2.4 Métricas Arquiteturais

As métricas arquiteturais são definidas com o objetivo de medir abstrações relacionadas a elementos arquiteturais, tais como, componentes e interfaces. Elas fornecem indicadores de modularidade, uma vez que podem medir propriedades como acoplamento entre componentes, coesão de um componente e complexidade de uma interface.

Neste trabalho, métricas para avaliar princípios básicos de projeto, tais como, baixo acoplamento e alta coesão, são denominadas como convencionais [11, 15, 62] e podem ser usadas para avaliar uma PLA. A Tabela 2.1 apresenta algumas métricas convencionais, extraídas de [100], que foram utilizadas nos experimentos realizados nesta tese. A métrica de coesão H avalia quão fortemente conectados são os elementos dos componentes de um projeto. Elementos fracamente conectados indicam baixa coesão. A métrica de tamanho NumOps revela aspectos de reusabilidade de uma PLA porque interfaces pequenas são, em geral, mais facilmente reutilizadas. As demais métricas referem-se ao acoplamento entre elementos arquiteturais.

Além desses princípios, a elegância de uma PLA também pode ser medida por meio de métricas de elegância de projeto OO [89]. Considerando um projeto OO, a elegância diz respeito à simetria e à uniformidade de distribuição de atributos e métodos pelas classes do projeto, embora, de alguma forma, sejam dependentes do contexto do projeto e do projetista. As métricas de elegância utilizadas na presente tese são apresentadas na Tabela 2.1. Simons *et al.* [89] concluíram que a elegância simétrica é significativa no projeto de software e que pode ser explorada no projeto de software baseado em busca para produzir projetos de software elegantes.

Tabela 2.1: Métricas Convencionais (CM)

Atributo	**Métrica**	**Definição**
Coesão	Coesão Relacional (H)	Número médio de relacionamentos internos entre as classes e interfaces de um pacote.
Acoplamento	Dependência de Pacotes (DepPack)	Número de pacotes dos quais classes e interfaces desse pacote dependem.
	Dependências de Entrada de uma Classe (CDepIn)	Número de elementos arquiteturais que dependem dessa classe.
	Dependências de Saída de uma Classe (CDepOut)	Número de elementos arquiteturais dos quais essa classe depende.
	Dependências de Entrada (DepIn)	Número de dependências UML nas quais o pacote é o fornecedor.
	Dependências de Saída (DepOut)	Número de dependências UML nas quais o pacote é o cliente.
Tamanho	Número de Operações por Interface (NumOps)	Número de operações de uma interface.
Elegância	Elegância de números entre classes (NAC)	Desvio padrão dos números de atributos e métodos entre as classes de um projeto.
	Elegância de acoplamentos externos (EC)	Desvio padrão dos acoplamentos externos entre as classes de um projeto.
	Elegância da razão entre atributos e métodos (ATMR)	Desvio padrão da razão entre atributos e métodos dentro das classes de um projeto.

Medidas específicas para LPS são mais apropriadas [4, 67, 94] para projeto de PLA e são descritas na Seção 2.4.1. Além disso, métricas sensíveis a interesses [83], descritas na Seção 2.4.2, podem ser usadas para analisar a modularização de características.

2.4.1 Métricas para LPS

Esta seção tem o objetivo de descrever trabalhos que visam à definição de métricas para a avaliação de PLA. Essas métricas são mais apropriadas do que as métricas convencionais, porque consideram aspectos inerentes à estrutura de arquiteturas de LPS como características variáveis e opcionais.

Oliveira Junior [67] definiu uma série de métricas básicas que visam a medir elementos UML essenciais aos modelos de LPS, como por exemplo, classes e componentes, com o objetivo de apoiar a definição de métricas compostas para atributos de qualidade. Dentre essas métricas básicas há 16 métricas para classes que: (a) indicam se uma classe é um ponto de variação, variante inclusiva, exclusiva, obrigatória ou opcional; (b) informam o número de subclasses que são variantes de uma classe que é ponto de variação; (c) informam o número de classes ou interfaces associadas a uma determinada classe que são ponto de variação, variante inclusiva, exclusiva, obrigatória ou opcional; e, (d) calculam o número de variabilidades de uma determinada classe.

Há ainda 16 métricas para interfaces que (a) indicam se uma interface é um ponto de variação, variante inclusiva, exclusiva, obrigatória ou opcional; (b) informam o número de interfaces associadas a uma determinada interface que são ponto de variação, variante inclusiva, exclusiva, obrigatória ou opcional; (c) informam o número de classes associadas a uma determinada interface que são ponto de variação, variante inclusiva, exclusiva, obrigatória ou opcional; e, (d) calculam o número de variabilidades de uma determinada interface.

Considerando um diagrama de classes, Oliveira Junior [67] define métricas para calcular: (a) o númeto total de classes que são pontos de variação, variantes inclusivas, exclusivas, obrigatórias e opcionais; (b) o número total de variabilidades em classes; (c) o número total de interfaces que são pontos de variação, variantes inclusivas, exclusivas, obrigatórias e opcionais; e, (d) o número total de variabilidades em interfaces. Para diagramas de componentes há uma métrica para indicar o número total de variabilidades em componentes.

Há ainda outras 7 métricas de componentes e modelos que podem medir: se um componente possui variação, o número total de pontos de variação em classes de uma arquitetura de LPS, o número total de pontos de variação em interfaces de uma PLA, o número total de pontos de variação em uma PLA, o número total de variabilidades em classes de uma arquitetura de LPS, o número de total de variabilidades em interfaces de uma PLA e o número total de variabilidades em uma PLA.

Chang, La e Kim [13] propuseram um método de avaliação de PLA que visa a medir o grau de aplicabilidade da PLA na engenharia de domínio. Para isso usam três métricas: (a) conformidade dos requisitos arquiteturais, (b) liberdade de conflitos entre elementos arquiteturais, e, (c) capacidade de adaptação (*tailorability*) da arquitetura projetada para gerar os produtos da LPS. Essas métricas foram propostas considerando: (i) que as variabilidades arquiteturais não existem somente nos componentes mas também nos relacionamentos e nos estilos arquiteturais empregados, e, (ii) que a validação de requisitos não-funcionais em LPS é um problema mais técnico e complicado do que o equivalente em sistemas únicos. Essas características levam a duas questões chave no projeto de PLAs: a cadeia de propagação de variabilidades e os conflitos entre elementos arquiteturais.

Apel e Beyer [4] propuseram um conjunto de métricas para medir a coesão de uma característica, e definiram a coesão de uma característica como o grau de dependência entre os elementos (métodos, atributos, classes) de uma mesma característica. As métricas não consideram somente o número de referências entre os elementos relacionados a uma característica, mas também a distância entre eles. Para analisar tal distância, são consideradas as distâncias baseadas nas dependências internas e externas à característica.

Há ainda métricas específicas para analisar a coesão de componentes da PLA (*Provide Service Utilization* (PSU) e *Required Service Utilization* (RSU) [94]) e para medir a extensibilidade de LPS [67]. Essas métricas são descritas nas próximas subseções.

2.4.1.1 RSU e PSU

O trabalho de van der Hoek, Dincel e Medvidovic [94] é voltado para uma qualidade específica de PLA: a sua solidez (*structural soundness*). Eles propõem métricas de utilização

de serviços que podem ser aplicadas tanto durante o projeto inicial como na manutenção da PLA. A métrica RSU (*Required Service Utilization*) mede a taxa de satisfação de um componente e PSU (*Provided Service Utilization*) mede a taxa de utilização de um componente. Para um componente individual x, RSU e PSU são calculadas de acordo com as Equações 2.1 e 2.2, onde: P_{actual} é o número de serviços fornecidos pelo componente x que são realmente usados por outros componentes, P_{total} é o número total de serviços fornecidos pelo componente x, R_{actual} é o número de serviços requeridos pelo componente x que são realmente fornecidos por outros componentes, R_{total} é o número total de serviços requeridos pelo componente x.

$$PSU_x = \frac{P_{actual}}{P_{total}} \tag{2.1}$$

$$RSU_x = \frac{R_{actual}}{R_{total}} \tag{2.2}$$

Os valores absolutos de PSU e RSU podem indicar problemas em potencial, por exemplo, um componente com RSU próximo de zero pode não funcionar bem na arquitetura porque isso significa que ele tem várias funcionalidades extras que não são usadas por outros componentes.

Os valores de PSU e RSU são úteis para avaliar a estrutura arquitetural escolhida, mas não são determinantes isoladamente. É importante calcular RSU e PSU de todos os componentes arquiteturais, independentemente do tamanho do componente (primitivo ou complexo). As médias de RSU e PSU dos componentes internos a um determinado componente complexo podem ser usadas para analisar a coesão desse componente. Quanto mais próximas de 1 forem os valores das médias de PSU e RSU, mais autocontido será o componente e, portanto, mais coeso [94].

Além disso, é salutar a avaliação da arquitetura como um todo. Para isso deve-se calcular PSU Composta (CPSU) e RSU Composta (CRSU) definidas nas Equações 2.3 e 2.4, onde P_{actual}, P_{total}, R_{actual}, R_{total} são definidos como nas métricas PSU e RSU e n é o número de componentes da arquitetura.

$$CPSU = \frac{\sum_{i=1}^{n} P_{actual}^{i}}{\sum_{i=1}^{n} P_{total}^{i}} \tag{2.3}$$

$$CRSU = \frac{\sum_{i=1}^{n} R_{actual}^{i}}{\sum_{i=1}^{n} R_{total}^{i}} \tag{2.4}$$

CPSU e CRSU servem para avaliar a coesão interna de uma PLA. Valores de CPSU e CRSU próximos de 1 indicam arquiteturas autocontidas e totalmente funcionais. Baixos CPSU e CRSU indicam arquiteturas desbalanceadas.

Pode-se ainda calcular a média de RSU e PSU de todas as possíveis configurações da arquitetura. Se a média de RSU é 1 isso indica que todas as configurações possíveis são válidas. Uma média de RSU muito baixa indica que algumas configurações da arquitetura não são válidas. Quanto à média de PSU, se o seu valor é 1, isso indica que todos os serviços que são fornecidos são realmente usados em cada instância de PLA. Na realidade, o valor deve ser mais baixo, já que devido à flexibilidade esperada de uma LPS, não se pode esperar que todos os serviços sejam usados em cada produto da LPS. Porém, se a média for muito baixa, isso indica um grau de separação na PLA e, talvez seja necessária a sua divisão em 2 ou mais arquiteturas de LPS [94].

van der Hoek, Dincel e Medvidovic [94] ainda discutem como analisar os efeitos da inclusão de componentes opcionais e variáveis em possíveis instâncias da PLA sobre os valores de PSU, RSU, CPSU e CRSU. No entanto, isso está fora do escopo do presente trabalho.

2.4.1.2 Extensibilidade de LPS

Oliveira Junior *et al.* [69] propuseram métricas para medir a extensibilidade de uma PLA. A métrica de extensibilidade de LPS pode fornecer um indicador do grau de reusabilidade de uma PLA. Quanto mais extensíveis forem os componentes da arquitetura, maior sua taxa de reusabilidade. A métrica de extensibilidade de uma PLA é composta por outras métricas para classes e componentes definidas a seguir:

- **ExtensClass**: é o nível de extensibilidade de uma classe. Fornece a porcentagem de métodos abstratos com relação ao total de métodos (abstratos mais os concretos) de uma classe. Esta métrica é representada pela Equação 2.5.

$$ExtensClass(Cls) = NumMetodosAbstratosCls/NumMetodosCls \tag{2.5}$$

- **ExtensVarPointClass**: é o valor da métrica *ExtensClass* (Equação 2.5), da classe que é um ponto de variação, mais a soma do valor da métrica *ExtensClass* (Equação 2.5) de cada variante associada à classe. Esta métrica é representada pela Equação 2.6.

$$ExtensVarPointClass(Cls) = ExtensClass(Cls) + \sum_{i=1}^{n} ExtensClass(ClsAss_i) \tag{2.6}$$

- **ExtensVariabilityClass**: é a soma da medida da métrica ExtensVarPointClass (Equação 2.6), de cada ponto de variação de uma determinada variabilidade. Esta métrica é representada pela Equação 2.7.

$$ExtensVariabilityClass(Vbt) = \sum_{i=1}^{nVP} ExtensVarPointClass(Cls_i) \tag{2.7}$$

- **ExtensVarComponent**: é a soma da medida da métrica ExtensVariabilityClass (Equação 2.7), de cada classe que forma um componente. Esta métrica é representada pela Equação 2.8.

$$ExtensVarComponent(Cpt) = \sum_{i=1}^{nCls} ExtensVariabilityClass(Cls_i) \tag{2.8}$$

- **ExtensPLA**: é o nível geral de extensibilidade da PLA, sendo a soma dos valores da métrica ExtensVarComponent (Equação 2.8) para cada componente de uma PLA. Esta métrica é representada pela Equação 2.9.

$$ExtensPLA(ALP) = \sum_{i=1}^{nCpt} ExtensVarComponent(Cpt_i) \quad (2.9)$$

2.4.2 Métricas Sensíveis a Interesses

Interesses (*concerns*) correspondem a requisitos, funcionais ou não, que devem estar presentes em um sistema de software. Dada esta definição, pode-se considerar que características de LPS são interesses do domínio. Sant'Anna [83] propôs um conjunto de métricas utilizando o conceito de interesse como forma de abstração para mensurar e garantir a modularidade de um projeto de software. Um interesse pode estar associado a mais de um elemento arquitetural (componentes, interfaces ou operações de interface). Além disso, um mesmo elemento arquitetural pode realizar mais de um interesse, ou parte de mais de um interesse. Isto ocorre pois alguns interesses não estão bem modularizados e, desse modo, não estão restritos a componentes cujo único propósito é realizá-los. Consequentemente, a interação entre interesses se dá não somente pelo relacionamento entre componentes, mas também pela presença desses interesses em um mesmo elemento arquitetural. Essa interação pode ocorrer de três formas: em nível de componente, em nível de interface ou em nível de operação.

Sant'Anna [83] afirma ainda que as métricas convencionais não são sensíveis a interesses arquiteturais e, portanto, não conseguem auxiliar o arquiteto de software a identificar anomalias de modularidade relacionadas aos interesses relevantes para o projeto [83]. Sendo assim, os objetivos específicos das Métricas Sensíveis a Interesses (MSI) são: a identificação de defeitos no projeto arquitetural que são causados pela falta de modularidade nos interesses relevantes para o projeto, e permitir a comparação das possíveis soluções para o projeto arquitetural, verificando a modularidade dos interesses relevantes para o projeto. Nesse sentido, considerando cada característica da LPS como um interesse, as MSI podem ser utilizadas para fornecer indicadores a respeito da modularização de características.

Dentre as MSI destacam-se as relacionadas à coesão baseada em interesses e difusão e sobreposição de interesses nos elementos arquiteturais, descritas a seguir:

Métricas para difusão de interesses

Esse grupo de métricas conta o número de elementos arquiteturais associados a cada interesse. Essa categoria assume que um interesse espalhado em um grande número de elementos é prejudicial para a modularidade.

As métricas pertencentes a este conjunto são:

- *Concern Diffusion over Architectural Components (CDAC)*: conta o número de componentes que contribuem para a realização de um dado interesse. Ela leva em consideração, além dos componentes que contribuem inteiramente para o interesse, os componentes que possuem pelo menos uma interface ou uma operação associada ao interesse;

- *Concern Diffusion over Architectural Interfaces (CDAI)*: conta o número de interfaces que contribuem para a realização de um dado interesse. Ela leva em consideração as interfaces que contribuem inteiramente para o interesse e aquelas que possuem pelo menos uma operação associada ao interesse;

- *Concern Diffusion over Architectural Operations (CDAO)*: conta o número de operações que contribuem para a realização de um dado interesse.

Métricas para interação entre interesses

Esse grupo de métricas objetiva avaliar as dependências entre interesses causadas por algum deles que não esteja bem modularizado e, dessa forma, não tenha limites bem definidos. Tais métricas assumem que muitas dependências entre interesses não ocorrem somente devido à dependência entre componentes.

As métricas pertencentes a esse conjunto são:

- *Component-level Interlacing Between Concerns (CIBC)*: conta o número de interesses com os quais um dado interesse compartilha ao menos um componente;

- *Interface-level Interlacing Between Concerns (IIBC)*: conta o número de interesses com os quais um dado interesse compartilha ao menos uma interface;

- *Operation-level Overlapping Between Concerns (OOBC)*: conta o número de interesses com os quais um dado interesse compartilha ao menos uma operação.

Métrica para coesão baseada em interesses

A métrica definida nesse grupo resulta do mapeamento dos interesses do sistema para os elementos arquiteturais. Ela consiste em contar, para cada componente, o número de interesses tratados por ele. A razão para esta métrica é que um componente associado a muitos interesses é pouco estável e uma modificação em qualquer interesse associado pode impactar nos outros interesses. A métrica pertencente a este grupo é: *Lack of Concern-based Cohesion (LCC)*. LCC conta o número de interesses com os quais um dado componente está associado, mais o número de interesses distintos com os quais as interfaces do componente estão associadas, mais o número de interesses distintos com os quais as operações dessas interfaces estão associadas.

2.5 Considerações Finais

Como mencionado anteriormente, de todos os trabalhos encontrados na literatura que envolvem a avaliação de PLA, a maioria realiza a avaliação qualitativa por meio de cenários e, somente uma pequena minoria inclui métricas para LPS. Muitos dos trabalhos que abordam a avaliação quantitativa da PLA incluem várias métricas a fim de analisar diferentes propriedades arquiteturais ao mesmo tempo. Somando-se a isso a afirmação de Savolainen [84] de que idealmente uma LPS deve ser constituída por componentes que fornecem baixo acoplamento, boa separação entre interesses não relacionados e melhor compreensibilidade da estrutura do sistema, tem-se que a definição e avaliação de PLA são atividades que envolvem vários fatores. Além disso, alguns desses fatores são naturalmente conflitantes, haja vista os princípios básicos de projeto de alta coesão e baixo acoplamento.

A avaliação de atributos de qualidade de arquiteturas de software é crítica na ponderação de alternativas de projeto e determinação se um sistema de software chegará aos alvos operacionais do usuário final. Além disso, quase todas as decisões arquiteturais não

triviais se transformam em *trade-offs* entre várias propriedades desejáveis, e, é necessário um arquiteto de software para projetar o *trade-off* entre objetivos conflitantes [30]. Portanto, a avaliação dos atributos de qualidade com base em métricas pode proporcionar ao arquiteto um *rationale* concreto e objetivo para as decisões fundamentais do projeto, além de reduzir o risco associado a um desenvolvimento em grande escala, como é o caso de LPS.

Dentre as métricas abordadas, somente as métricas de coesão de característica [4] e a métrica de extensibilidade [67] possuem ferramenta que automatiza o processo de sua aplicação. Sendo assim, outro ponto deficitário é a automatização do processo de medição com o objetivo de avaliar PLAs.

A abordagem de otimização de projeto de PLAs baseada em algoritmos de busca, proposta neste trabalho, automatiza a aplicação de métricas arquiteturais e, consequentemente, a avaliação de arquitetura. Com o objetivo de fornecer subsídios para o entendimento da referida abordagem, o próximo capítulo aborda os principais conceitos de otimização multiobjetivo e de algoritmos de busca, além dos trabalhos relacionados ao tema deste trabalho.

CAPÍTULO 3

OTIMIZAÇÃO DE PROJETO ARQUITETURAL

A Engenharia de Software está associada a uma diversidade de problemas que possuem grande quantidade de soluções possíveis, de forma que, encontrar a solução ideal é, em muitas situações, teoricamente impossível ou intratável na prática [42]. Essa característica torna as técnicas de otimização baseadas em busca fortes candidatas para encontrar soluções para os problemas da Engenharia de Software. Nas mais diferentes fases do processo de Engenharia de Software, são encontradas muitas situações que podem ser tratadas como problemas de otimização a serem resolvidos por algoritmos de busca. Esse fato culminou na criação de um novo e promissor campo de pesquisa na computação, denominado de *Search Based Software Engineering* (SBSE) [22, 39, 42, 44].

Algoritmos de busca são técnicas de otimização que permitem encontrar uma melhor solução dentro de um conjunto de possíveis soluções. Esses algoritmos são divididos em dois grupos principais [22]. O primeiro inclui técnicas clássicas do campo da pesquisa operacional e da programação linear. As técnicas clássicas costumam ser precisas e determinísticas. O segundo grupo inclui as meta-heurísticas, utilizadas principalmente para resolver problemas que não podem ser representados por equações matemáticas. Este grupo inclui Otimização por Nuvem de Partículas (PSO), Otimização por Colônia de Formigas (ACO) e algoritmos evolutivos.

Independentemente da técnica empregada, os conceitos fundamentais são comuns e são explicados a seguir. Em SBSE, o objetivo é otimizar (maximizar ou minimizar) uma função ou grupo de fatores que afetam o problema da Engenharia de Software sendo otimizado. Segundo Harman, Mansouri e Zhang [41], somente dois ingredientes chave são necessários para resolver um problema usando técnicas de busca: (i) uma representação do problema que permita a manipulação simbólica; e, (ii) uma função objetivo (também

chamada de função de avaliação ou de *fitness*) definida em termos da representação para avaliar a qualidade das soluções.

Dois aspectos básicos estão associados às técnicas de busca: um espaço de busca, que contém as possíveis soluções (ou estados) do problema; e uma função objetivo (o segundo ingrediente mencionado), que avalia as soluções encontradas, associando-as a valores. As variáveis que definem a(s) função(ões) são definidas de acordo com cada instância do problema. Pode ser considerado ainda um terceiro aspecto, que são os operadores de busca (ou operadores de movimento). De forma geral, um operador de busca tem a finalidade de realizar pequenas alterações em uma solução, de forma a gerar uma nova solução, não muito diferente da solução original.

A representação do problema (primeiro ingrediente) é adotada tanto para as soluções fornecidas como entrada para o algoritmo de busca, como para as soluções geradas. As técnicas de busca usualmente têm representações padrão, tais como *strings* binárias e vetores de inteiros. Para aplicá-las a um espaço de busca complexo, geralmente é necessário um mapeamento genótipo-fenótipo, i.e., um mapeamento entre a representação padrão e a representação original da solução [31]. Em alguns casos, é possível aplicar os operadores de busca diretamente sobre a representação natural da solução, por exemplo, um diagrama de classes. Na Engenharia de Software, a função objetivo (segundo ingrediente) é medida em termos de métricas específicas para o problema que está sendo tratado. Dessa forma, cada solução é avaliada com base nas métricas utilizadas na função objetivo.

Nesse contexto, a busca é iniciada a partir de uma ou mais soluções do espaço de busca. Os operadores de busca geram novas soluções iterativamente a partir das soluções atuais, até que alguma condição de parada seja atingida, possibilitando assim o funcionamento da busca. Cada nova solução gerada é avaliada segundo a função objetivo e as soluções pior avaliadas são substituídas pelas melhores durante o processo de busca. Nesse processo, a melhor solução encontrada na busca é retornada ao usuário. Há diversas técnicas de otimização baseadas em busca bastante difundidas na literatura e utilizadas em SBSE, tais como, Subida na Encosta (*Hill Climbing*), Têmpera Simulada (*Simulated Annealing*), Algoritmos Evolutivos e ACO [12].

O uso dos algoritmos de busca mostra-se oportuno em problemas com mais de uma função objetivo (problemas multiobjetivos) e ainda em problemas para os quais não é conhecido algum algoritmo exato que finalize a execução em tempo prático. Além disso, os algoritmos de busca possuem um grau de generalidade que permite que eles sejam adaptados para diversos problemas específicos.

De fato, vários trabalhos apresentam resultados da aplicação desses tipos de algoritmos em diversos problemas, incluindo diferentes áreas da Engenharia de Software, tais como: testes, planejamento, gerenciamento de projetos, refatoração e projeto, dentre outras [22, 39, 41, 44]. Ainda assim, o uso de técnicas de otimização multiobjetivo na Engenharia de Software é recente [22, 39, 41, 44].

Considerando que o projeto de PLA é uma atividade que demanda grande esforço humano já que envolve uma série de fatores que impactam na definição da arquitetura, como atributos de qualidade e interesses de negócio, o uso de técnicas de otimização multiobjetivo pode alcançar bons resultados para resolver este problema. O mesmo tem ocorrido com os trabalhos que abordam o projeto de software baseado em busca, que se trata do problema de Engenharia de Software mais relacionado à otimização de projeto de PLA. Porém, como mencionado anteriormente, há outros fatores envolvidos, diretamente ligados às particularidades de LPS, que demandam o uso de métricas específicas para LPS, operadores de busca que privilegiem as necessidades de LPS e, assim por diante.

Na próxima seção são abordados os principais conceitos relacionados à otimização multiobjetivo, algoritmos evolutivos e algoritmos evolutivos multiobjetivos, e, na Seção 3.2 são descritos os trabalhos relacionados à abordagem proposta na presente tese, os quais tratam da aplicação de técnicas de SBSE para melhorar a arquitetura de software.

3.1 Otimização Multiobjetivo

Muitos problemas do mundo real, para os quais a solução ótima é desconhecida ou custosa para ser alcançada, podem ser representados por meio de um modelo que simplifica seu tratamento, reduzindo o espaço de busca e a influência de fatores externos ao objetivo principal do problema. O modelo tem um único objetivo e é bem definido, permitindo

tratar um vetor n-dimensional $X = [x_1, x_2, ..., x_n]$ de variáveis de decisão, pertencentes ao universo de busca Ω, que são avaliadas mediante uma função objetivo $f(X)$, possibilitando avaliar o grau de aceitação da solução [18]. Estes problemas são chamados mono-objetivo.

Problemas de otimização com duas ou mais funções objetivo são chamados de multiobjetivos. Este tipo de problema exige a otimização simultânea de vários interesses interdependentes e muitas vezes conflitantes e, por isso, não há uma solução única. Desta forma, o objetivo é encontrar um bom conjunto de soluções representando um possível compromisso entre os diversos interesses.

O problema de minimização multiobjetivo pode ser declarado como minimizar a Equação 3.1 [18].

$$F(x) = (f_1(x), ..., f_B(x)) \tag{3.1}$$

sujeito a $g_i(x) \leq 0, i = \{1, ..., m\}$ e $h_j(x) = 0, j = \{1, ..., p\}$ $x \in \Omega$, onde: x é um vetor de variáveis de decisão e Ω é um conjunto finito de possíveis soluções. Observa-se que $g_i(x) \leq 0$ e $h_j(x) = 0$ são restrições que devem ser satisfeitas durante a minimização de $F(x)$ e Ω contém todos os possíveis x que podem ser usados para satisfazer a avaliação de $F(x)$.

Considerando que $x \in \Omega$ e $y \in \Omega$ sejam duas soluções, para um problema de minimização, a solução x domina y se as Equações 3.2 e 3.3 são satisfeitas.

$$\forall f_i \in F, i = 1...B, f_i(x) \leq f_i(y) \tag{3.2}$$

$$\exists f_i \in F, f_i(x) < f_i(y) \tag{3.3}$$

x é uma solução não dominada se não existe qualquer solução y que domine x.

Para encontrar o conjunto de soluções candidatas a solucionar um problema multiobjetivo utiliza-se o conceito de dominância de Pareto [71], que permite comparar soluções considerando todos os objetivos do problema. O conceito de dominância de Pareto originou-se da área de economia com inúmeras aplicações a problemas do mundo real e em diversas

áreas, envolvendo a teoria dos jogos e engenharia [51]. Então, considerando um problema de minimização, uma solução $\overrightarrow{x}$ é melhor que $\overrightarrow{y}$ quando as Equações 3.2 e 3.3 são satisfeitas, ou seja, quando todos os valores de objetivos de $\overrightarrow{x}$ forem menores ou iguais aos valores de objetivos de $\overrightarrow{y}$ e existir pelo menos um valor de objetivo em $\overrightarrow{x}$ que seja menor que seu correspondente valor em $\overrightarrow{y}$.

Assim, para solucionar problemas multiobjetivos, procura-se encontrar soluções que melhor representem o compromisso entre os objetivos, que formam o conjunto com todas as soluções não dominadas no espaço de busca, conforme definido no problema de minimização acima. Estas soluções são chamadas de não-dominadas e formam a fronteira de Pareto [71]. Em muitas aplicações a busca pela fronteira de Pareto ótima (PF_{true}) é NP-difícil [51], então, o problema de otimização busca encontrar a aproximação mais próxima possível da fronteira de Pareto (PF_{approx}).

Pode-se ilustrar o tratamento de um problema multiobjetivo por meio do exemplo da Figura 3.1, cujo objetivo é minimizar dois fatores para escolher um meio de transporte: o tempo e o custo de deslocamento. Neste caso, as soluções que representam avião, carro, trem, ônibus e moto (soluções a, c, d, f e g) formam a PF_{approx}. As soluções b e e são dominadas por outras soluções com valores menores e, por isso, devem ser descartadas.

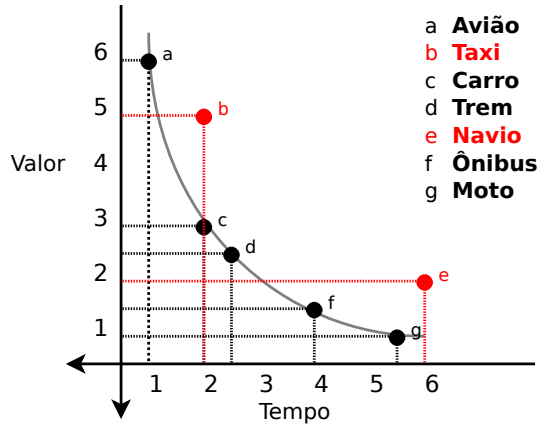


Figura 3.1: Exemplo de problema multiobjetivo com dois fatores

Existem três técnicas para solucionar problemas multiobjetivos [18]: (a) priorizar somente aquele objetivo que é considerado prioritário; (b) utilizar agregação ponderada de objetivos definindo um peso para cada objetivo; e, (c) aplicar algoritmos de busca multiobjetivos a fim de encontrar conjuntos de soluções não dominadas.

A primeira técnica não garante que as soluções encontradas se aproximem do conjunto PF_{approx}, já que a solução é avaliada somente segundo um dos objetivos e os outros objetivos podem estar distantes de possíveis pontos com melhores valores. Na segunda técnica cada solução é comparada com outras soluções segundo o resultado de uma função que agrega os objetivos ponderados. Esta técnica é melhor do que a primeira, entretanto, devem-se ajustar os parâmetros para permitir que o espaço de busca seja explorado de forma eficiente ao mesmo tempo que o objetivo desejado deve ser priorizado, o que não é uma atividade trivial. As duas técnicas tratam dois ou mais objetivos mas utilizam um único valor final para comparar as soluções, e assim é possível que somente uma solução seja considerada a melhor, o que não é condizente com problemas multiobjetivos.

Dessa maneira, a terceira técnica que consiste em utilizar algoritmos de busca multiobjetivos parece ser mais condizente, pois considera os objetivos independentemente, por meio do conceito de dominância de Pareto, a fim de encontrar um conjunto de possíveis soluções com diferentes compromissos entre os objetivos. Nesta técnica não são atribuídos pesos para os objetivos, de forma que todos têm a mesma importância durante a avaliação das soluções. Assim, de posse do conjunto de soluções alcançadas, o usuário deve escolher uma das soluções de acordo com suas prioridades, como, por exemplo, a solução que contém o melhor resultado para um determinado objetivo ou a solução de melhor *trade-off* entre os objetivos utilizados. Dessa forma, o uso de algoritmos de busca multiobjetivos tem a vantagem de oferecer maior diversidade de soluções para o usuário, o que não é possível utilizando qualquer uma das outras duas técnicas.

Além da técnica de otimização multiobjetivo, existem três diferentes formas de interação com o usuário: antes, durante ou depois do processo de otimização. A primeira forma ocorre quando o usuário fornece informações preliminares que serão utilizadas no processo de otimização. A segunda forma ocorre quando o usuário interage durante o processo de otimização, seja para avaliar e/ou editar soluções geradas, as quais são utilizadas na próxima iteração do processo de otimização. Na última forma, o usuário interage após o processo de otimização quando, ao receber as soluções retornadas do algoritmo, seleciona uma das soluções para ser utilizada.

Vários tipos diferentes de algoritmos meta-heurísticos foram adaptados para lidar com problemas multiobjetivos: ACO, PSO e algoritmos evolutivos [29]. Dentre os trabalhos de SBSE que aplicam otimização multiobjetivo, os MOEAs têm sido os mais usados e têm alcançado resultados mais promissores que as outras meta-heurísticas [7, 22, 35, 98]. Além disso, a maioria dos trabalhos sobre otimização de projeto arquitetural utiliza algoritmos evolutivos, adotando a otimização multiobjetivo em muitos casos, como pode ser visto na Seção 3.2. Por esses motivos, os MOEAs são descritos na Seção 3.1.2.

Diante da existência de diferentes tipos de algoritmos multiobjetivos é preciso dispor de meios para comparar os resultados alcançados. Existem alguns indicadores de qualidade que têm como intuito a avaliação dos resultados desses algoritmos. Alguns exemplos de indicadores de qualidade são *Generational Distance* (GD) [97], *Inverted Generational Distance* (IGD) [75], *Coverage* [104], *Hypervolume* [105], etc. Os dois indicadores de qualidade que foram utilizados nos estudos empíricos realizados nesta tese são descritos na próxima seção.

3.1.1 Indicadores de Qualidade

Tendo em vista que os indicadores de qualidade permitem avaliar o desempenho de algoritmos de busca multiobjetivos, é possível utilizá-los para comparar os resultados alcançados por diferentes algoritmos. Esses indicadores baseiam-se geralmente nos conjuntos de soluções alcançados pelos algoritmos. Três conjuntos costumam ser utilizados:

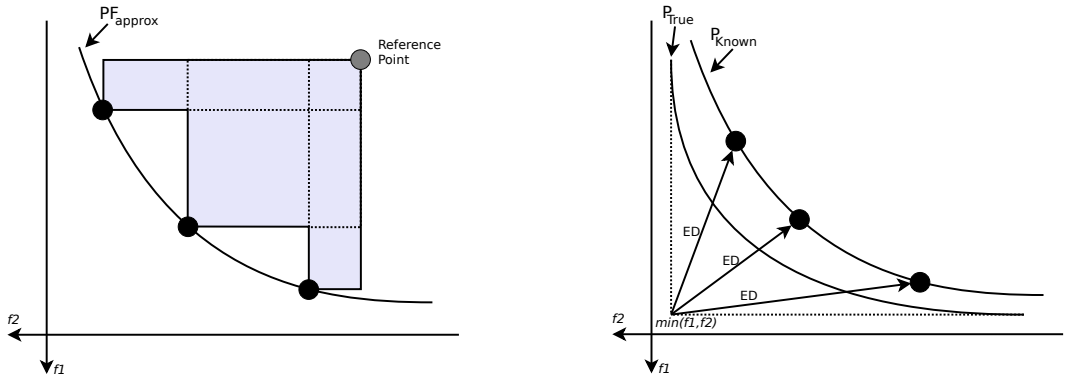
- PF_{approx}: é formado pelas soluções não dominadas retornadas por um algoritmo em uma execução. Costumeiramente, dada a característica não determinística dos algoritmos de busca, várias rodadas do mesmo algoritmo são executadas para um mesmo problema. Logo, se forem executadas 30 rodadas, 30 conjuntos PF_{approx} serão obtidos por esse algoritmo.

- PF_{known}: é obtido por meio da união de todos os PF_{approx} alcançados por um determinado algoritmo, removendo as soluções dominadas e repetidas. Dessa forma,

PF_{known} representa as melhores soluções encontradas por um algoritmo para um determinado problema.

- PF_{true}: representa a fronteira de Pareto ótima para o problema. No entanto, quando a fronteira não é conhecida, recomenda-se que o conjunto PF_{true} seja obtido pela união de todas as soluções encontradas em todas as rodadas de todos os algoritmos que tiverem sido executados, removendo as soluções dominadas e repetidas [105]. Esse procedimento foi adotado nos experimentos realizados neste trabalho.

O indicador de qualidade mais aceito é o Hypervolume (HV) [105]. Comparando dois conjuntos PF_{approx}: quando um PF_{approx} domina completamente outro PF_{approx}, o HV do primeiro é maior que o HV do segundo. O indicador HV calcula o volume da região delimitada entre um PF_{approx} e um ponto de referência. O ponto de referência é um ponto dominado pelas soluções de todos os conjuntos PF_{approx} encontrados pelo(s) algoritmo(s). A Figura 3.2(a) ilustra o cálculo realizado para obter o valor de HV.



(a) HV (b) ED

Figura 3.2: Indicadores de Qualidade

Além do HV, neste trabalho um outro indicador de qualidade foi utilizado, denominado Distância Euclidiana à Solução Ideal (ED). ED não é de fato um indicador de qualidade, mas é costumeiramente utilizado como uma medida que ajuda tomadores de decisão a selecionar, a partir de todas as soluções encontradas, uma solução a ser utilizada. O propósito deste indicador é encontrar a solução mais próxima da solução ideal. Uma solução ideal tem o menor valor de cada objetivo, considerando um problema de minimização [17]. Esse valores mínimos são obtidos a partir das soluções que formam o

conjunto PF_{true}. Nesse sentido, a solução com menor ED tem o melhor *trade-off* entre os objetivos. A Figura 3.2(b) mostra um exemplo do cálculo da ED para um problema de minimização com dois objetivos.

Como mencionado anteriormente, os indicadores de qualidade permitem comparar a qualidade das soluções obtidas por diferentes algoritmos de busca multiobjetivos. No caso do HV, o algoritmo cuja fronteira PF_{true} ocupa a maior área tem a melhor qualidade. No caso da ED, o algoritmo que alcançou a solução com menor ED supera os resultados do outro algoritmo. Além da qualidade, o HV também fornece indicadores a respeito da diversidade de soluções alcançadas por um determinado algoritmo, já que quanto maior a diversidade de soluções menor a distância entre os pontos que representam as soluções e a curva do PF_{true}, implicando numa maior área (HV).

3.1.2 Algoritmos Evolutivos Multiobjetivos

Algoritmos evolutivos são inspirados na teoria da seleção natural e evolução biológica dos seres vivos para otimização de processos classificados como complexos [12]. Os operadores de busca que realizam os movimentos básicos dos algoritmos evolutivos são os operadores de cruzamento de indivíduos e operadores de mutação dos novos indivíduos gerados. Então, considerando as taxas de probabilidade de ocorrer o cruzamento e/ou a mutação, a população inicial é evoluída, geração por geração, por meio da aplicação dos operadores de cruzamento e mutação.

Há diferentes categorias de algoritmos evolutivos, tais como [8]: (a) algoritmos genéticos (*Genetic Algorithms* - GA); (b) estratégias evolutivas; (c) programação genética; e, (d) programação evolutiva. Os GAs foram propostos visando o desenvolvimento de sistemas artificiais (simulados em computador) que retenham os mecanismos originais encontrados em sistemas naturais. As estratégias evolutivas têm o objetivo de solucionar problemas de otimização de parâmetros, tanto discretos como contínuos, e empregam apenas o operador de mutação. A programação genética é uma extensão dos GAs e tem por objetivo evoluir programas de computador usando os princípios da evolução natural. A programação evolutiva foi originalmente proposta como uma técnica para criar inteligência artificial

por meio da evolução de máquinas de estado finito. A programação evolutiva também emprega apenas mutação.

Os GAs são muito utilizados em SBSE e seu processo básico de funcionamento é descrito a seguir [12]:

1. **Inicialização**: A população inicial com soluções candidatas costuma ser gerada aleatoriamente dentro do espaço de estados, apesar de que conhecimento específico do domínio pode ser facilmente incorporado.

2. **Avaliação**: Uma vez que a população é inicializada ou que uma nova população descendente é criada, avalia-se o valor de *fitness* (ou de aptidão) das soluções candidatas.

3. **Seleção**: A seleção aloca probabilisticamente mais cópias daquelas soluções com os melhores valores de *fitness* (ou de aptidão), dessa forma, as melhores soluções candidatas sobreviverão na próxima população. A principal idéia da seleção é preferir as melhores soluções em detrimento das piores, como ocorre na seleção genética humana, em que os indivíduos mais aptos sobrevivem. Os principais métodos de seleção usam esta idéia, como a seleção por roleta, por torneio e por *ranking*.

4. **Cruzamento**: O cruzamento combina partes de duas soluções pais para criar uma nova solução, possivelmente com melhor *fitness*. Há várias formas de fazer o cruzamento e o mecanismo projetado pode ser específico para o tipo de representação das soluções candidatas.

5. **Mutação**: Enquanto o cruzamento atua sobre dois ou mais cromossomos pais, a mutação modifica aleatoriamente a solução criada pelo cruzamento (filho). Há várias formas de se realizar a mutação, mas geralmente ela executa uma varredura aleatória na vizinhança da solução candidata.

6. **Substituição**: A população descendente criada a partir da seleção, cruzamento e mutação substitui a população pai. Existem várias técnicas de substituição, por

exemplo, elitismo, em que somente as soluções com os melhores *fitness* permanecem na população descendente.

7. Repetir os passos 2-6 até que a condição de parada seja alcançada. A condição de parada pode ser dada, por exemplo, por um número máximo de gerações ou por número de avaliações de *fitness*.

Na literatura foram propostos algoritmos variantes de GA adaptados para problemas multiobjetivos. Três MOEAs representativos, que são variantes dos GA tradicionais, são: NSGAII(*Non-dominated Sorting Genetic Algorithm*) [25], SPEA2(*Strenght Pareto Evolutionary Algorithm*) [103] e PAES(*Pareto Archived Evolution Strategy*) [52]. NSGA-II e SPEA2 têm sido utilizados em vários estudos envolvendo problemas multiobjetivos na área de Engenharia de Software. No entanto, NSGAII é o mais utilizado [42]. PAES é um MOEA que não utiliza o operador de cruzamento e, por isso, tem comportamento diferente do NSGAII. A seguir são descritos os dois MOEAs utilizados nos experimentos desta tese (NSGAII e PAES), visando a destacar as diferentes estratégias de evolução e diversificação utilizadas para lidar com os problemas de otimização multiobjetivo.

3.1.2.1 *Non-dominated Sorting Genetic Algorithm*

O *Non-dominated Sorting Genetic Algorithm* (NSGAII) [25] é um MOEA baseado em GA que possui forte estratégia de elitismo. Seu pseudocódigo é apresentado no Algoritmo 1. Como pode ser observado no pseudocódigo, as entradas do algoritmo são o tamanho da população N', o número de gerações g e as funções a serem otimizadas $f_k(X)$.

O NSGAII ordena a população em várias fronteiras não-dominadas. Em cada geração ele ordena os indivíduos das populações de pais e filhos, de acordo com a dominância entre as soluções, formando diversas fronteiras (linhas 10 e 11 do Algoritmo 1). A primeira fronteira é composta por todas as soluções não dominadas. Após a exclusão das soluções da primeira fronteira, a segunda fronteira é formada pelas soluções que se tornaram não dominadas. Da mesma forma, a terceira fronteira é formada pelas soluções que se tornaram não dominadas após a exclusão das soluções da primeira e da segunda fronteiras, e, assim por diante, até que todas as soluções estejam classificadas em alguma fronteira.

Algorithm 1: Pseudocódigo do NSGAII (adaptado de [18])

```
1  Entrada: N', g, f_k(X)
2  Inicializar população ℙ'
3  Gerar população aleatória - Tamanho N'
4  Avaliar valores dos objetivos
5  Atribuir rank baseado na dominância de Pareto - Ordenação
6  Gerar população filho
7      Seleção por torneio binário
8      Recombinação e Mutação
9  para i=1 até g hacer
10     para cada Pai e Filho na População hacer
11         Atribuir rank baseado na dominância de Pareto - Ordenação
12         Gerar fronteiras não dominadas
13         Determinar a distância de multidão para cada solução das fronteiras e percorrer todas as
           fronteiras adicionando para a próxima geração do primeiro ao N' indivíduo
14     fin
15     Selecionar indivíduos das melhores fronteiras e com maior distância de multidão
16     Gerar população filho
17         Seleção por torneio binário
18         Cruzamento e Mutação
19 fin
```

Para cada fronteira outra ordenação é feita usando uma medida que visa a manter a diversidade das soluções. Essa medida é chamada de distância de multidão (*crowding distance*). A distância de multidão calcula o quão distante uma solução está em relação aos seus vizinhos da mesma fronteira. Assim, é possível estabelecer uma ordem decrescente que privilegia as soluções mais espalhadas no espaço de busca. As soluções que estão no limite do espaço de busca possuem somente um vizinho, porém, são as mais diversificadas da fronteira, por isso, recebem altos valores de forma a ficarem no topo da ordenação.

Ambas ordenações, de fronteiras e de distância de multidão, são usadas pelo operador de seleção (linhas 6 e 16 do Algoritmo 1) e para determinar os indivíduos que sobrevivem para a próxima geração. O NSGAII utiliza a seleção por torneio, selecionando soluções de fronteiras com maior dominância e, em caso de empate na dominação, é utilizado como critério de desempate a distância de multidão.

O processo de ordenação em fronteiras, ordenação pela distância de multidão e o elitismo são ilustrados na Figura 3.3, em que P_t é a população dos pais; Q_t é a população dos filhos; F_1, F_2 e F_3 são fronteiras de soluções já ordenadas da união de P_t e Q_t; e, P_{t+1} representa o conjunto de soluções que serão usadas na próxima geração.

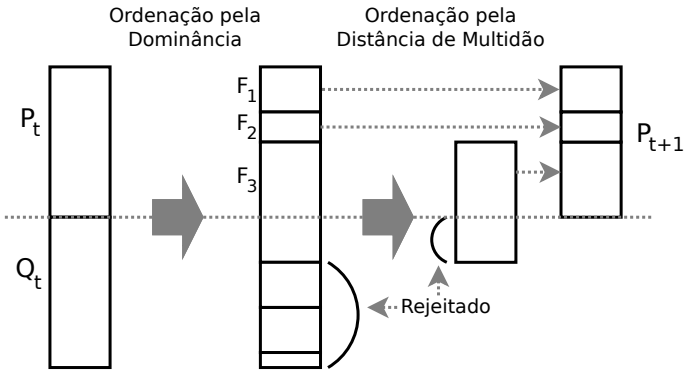


Figura 3.3: Diagrama de funcionamento do elitismo no NSGAII (adaptado de [25])

O algoritmo NSGAII é um dos MOEAs mais tradicionais e é largamente utilizado em comparações com outros algoritmos [6, 18, 22].

3.1.2.2 *Pareto Archived Evolution Strategy*

No processo evolutivo do MOEA *Pareto Archived Evolution Strategy* (PAES) [52] o conceito de população é diferente das estratégias tradicionais de GAs, pois apenas uma solução é mantida em cada geração e os novos indivíduos são obtidos somente por meio do operador de mutação, como pode ser observado na linha 3 do Algoritmo 2. Não há possibilidade de se utilizar o operador de cruzamento já que o PAES trabalha apenas com uma solução por geração. Um arquivo externo de soluções é populado com as soluções não dominadas encontradas durante o processo.

Algorithm 2: Pseudocódigo do PAES (adaptado de [18])

```
1  Entrada: f_k(X)
2  repita
3  |  Inicializar população com um único pai C e adicionar para o arquivo 𝔸
4  |  Mutar C para produzir um filho C' e avaliar seu fitness
5  |  se C ≺ C' então
6  |  |  Descartar C'
7  |  senão se C ≻ C' então
8  |  |  Substituir C por C', e adicionar C' para 𝔸
9  |  senão se ∃_{c''∈𝔸}(C'' ≺ C') então
10 |  |  Descartar C'
11 |  senão
12 |  |  Testar (C, C', 𝔸) para determinar qual será a solução que continuará no processo
   |  |  evolutivo, possibilitando adicionar C' para 𝔸
13 |  fim
14 até atingir critério de parada;
```

A cada geração o PAES cria uma nova solução filho que é comparada com a solução pai: se a solução filho é dominada pela solução pai, a solução filho é descartada (linhas 4 e 5 do Algoritmo 2); se a solução filho domina a solução pai, o filho toma o lugar do pai e o filho é acrescentado ao arquivo externo (linhas 6 e 7); se a solução filho for dominada por alguma solução do arquivo, o filho é descartado (linhas 8 e 9); e, caso nenhuma das soluções pai, filho e do arquivo seja dominante, a escolha da solução que vai permanecer no processo evolutivo é feita considerando a diversidade entre o pai, o filho e as soluções do arquivo externo (linhas 10 e 11).

Caso o tamanho do arquivo externo exceda, uma estratégia de diversificação é aplicada sobre o conjunto de soluções a fim de eliminar soluções similares e manter um espaço de busca a ser explorado maior. O pseudocódigo do algoritmo PAES apresentado no Algoritmo 2 representa uma otimização de minimização.

3.2 Trabalhos Relacionados

Alguns trabalhos utilizam técnicas de busca multiobjetivo para otimizar a arquitetura da linha de produto de domínios diferentes do de desenvolvimento de software, por exemplo, motores elétricos, turbinas a gás e ventiladores. Alguns se preocupam com o melhor particionamento da arquitetura física de um produto de uma linha de produto em um conjunto ótimo de módulos ou subsistemas [102]. Enquanto outros focam na maximização do número de diferentes produtos que podem ser gerados a partir da arquitetura da linha de produto [93] ou da participação de mercado dos produtos de uma empresa [56]. Apesar de otimizarem a arquitetura de linha de produto em geral, esses trabalhos não tratam de LPS e não têm uma aplicação direta dentro do contexto desta tese.

Há poucos trabalhos que empregam técnicas de SBSE para apoiar a Engenharia de LPS e, geralmente, focam a configuração automática de produtos e a seleção de características, tanto para o desenvolvimento de LPS como para a sua evolução [49, 85, 99]. Então, seu foco não é otimizar o projeto de PLA.

Nesse sentido, o foco dos trabalhos que usam técnicas baseadas em busca para otimizar arquiteturas de software [2, 77] é mais próximo do projeto de PLA. Enquanto o *survey* de Aleti *et al.* [2] é focado em trabalhos que visam a otimizar a confiabilidade, desempenho e custo da arquitetura de software em nível de implantação, o *survey* de Raiha [77] é mais abrangente, incluindo a otimização de projeto OO, projeto orientado a serviços, modularização e refatoração de software.

Há trabalhos [65, 66, 87] que abordam a refatoração de código visando a melhorar a estrutura de classes de um software OO. Esses trabalhos modelam o projeto OO como um problema de otimização combinatorial, onde a função de *fitness* define a qualidade do projeto, geralmente, com base em uma soma ponderada de métricas convencionais OO. Apesar de otimizar o projeto de software, eles não estão diretamente relacionados ao presente trabalho porque a informação de entrada é o código fonte, o alvo da otimização é o código e o processo evolutivo busca encontrar a melhor sequência de refatorações a ser aplicada.

Nesse contexto, os trabalhos que envolvem a refatoração de projeto e projeto de software baseado em busca são mais relacionados a esta tese. A Tabela 3.1 contém uma síntese dos principais trabalhos relacionados, destacando qual algoritmo de busca, métricas e representação do problema foram utilizados em cada trabalho. Na terceira coluna da tabela, além de indicar se o trabalho é mono ou multiobjetivo, o número de objetivos é apresentado entre parênteses quando se trata de um trabalho multiobjetivo. Somente um dos trabalhos não menciona o número de objetivos utilizados.

Alguns trabalhos tratam da refatoração arquitetural em nível de implantação [1, 31, 37, 57, 61]. A maioria deles usa MOEAs para otimizar arquiteturas baseadas em componentes com respeito a requisitos não funcionais, tais como, desempenho, confiabilidade e custo [37, 57, 61]. A abordagem multiobjetivo de Etemaadi *et al.* [31] aplica padrões arquiteturais e padrões de projeto para otimizar o projeto arquitetural do software, com relação a requisitos como desempenho, segurança e confiabilidade. Os padrões são implementados como operadores de busca. Essa abordagem propõe que a arquitetura a ser otimizada seja representada na própria linguagem de especificação, uma ADL (*Architecture Description*

Tabela 3.1: Síntese dos trabalhos relacionados

Problema	**Trabalho**	**Multi/Mono-objetivo**	**Algoritmo utilizado**	**Métrica(s)**	**Representação do Problema**	**Aplicado a LPS?**
Refatoração em nível de implantação	Grunske [37]	Multi (2)	Algoritmo evolutivo não especificado	custo e confiabilidade	cadeia de caracteres	Não
	Martens et al. [61]	Multi (3)	NSGAII	desempenho, confiabilidade e custo	arquitetura (visão de implantação)	Não
	Etemaadi *et al.* [31]	Multi	Algoritmo evolutivo não especificado	requisitos não funcionais	modelo em ADL	Não
Refatoração em nível de projeto	Jensen e Cheng [47]	Mono	Programação genética	coesão e acoplamento	grafo de transformações	Não
	Qayum e Heckel [74]	Mono	ACO	métricas OO	grafo de transformações	Não
	Amoui et al. [3]	Mono	GA	distância da sequência principal	grafo de transformações	Não
	Harman e Tratt [43]	Multi (2)	Hill Climbing	métricas OO	lista de refatorações	Não
Projeto de software	Räihä [79]	Multi (2)	GA	modificabilidade e eficiência	coleção de vetores de inteiros (supergenes)	Não
	Bowman, Briand e Labiche [10]	Multi (5)	SPEA2	coesão e acoplamento	vetor de inteiros	Não
	Simons [90]	Multi (2)	NSGAII	coesão e acoplamento	arquitetura baseada em objetos	Não

Language). Então, uma solução é representada em um modelo do sistema, em vez de aplicar um mapeamento genótipo-fenótipo. Os operadores evolutivos foram implementados de forma a permitir a transformação de modelo para modelo.

Alguns outros trabalhos focam a refatoração de arquiteturas já existentes visando a melhorar a sua qualidade em nível estrutural [43, 47, 74]. Eles representam a arquitetura em diagramas de classes e usam um grafo de transformações para determinar as operações de refatoração que acrescentam padrões de projeto na arquitetura [47, 74]. O trabalho de Qayum e Heckel [74] é mono-objetivo e utiliza ACO. Cada solução é avaliada considerando o custo da refatoração e a qualidade do resultado final. O trabalho de Jensen e Cheng [47] usa programação genética e métricas convencionais. No trabalho de Amoui *et al.* [3], cada solução é avaliada de acordo com uma métrica convencional chamada Distância da Sequência Principal [62] utilizando GA. Dentre esses trabalhos, somente o trabalho de Harman e Tratt [43] utiliza otimização multiobjetivo para tratar a refatoração, aplicando o algoritmo Hill Climbing com dois objetivos (métricas OO).

3.2.1 Projeto de Software Baseado em Busca

Esta seção tem como objetivo descrever os trabalhos mais diretamente relacionados ao tema desta tese. Tratam-se de trabalhos mais recentes que os anteriormente citados e que focam na definição da arquitetura de software durante a fase de projeto. Tais trabalhos utilizam técnicas de SBSE para otimizar uma arquitetura ainda em desenvolvimento, em vez de arquiteturas já existentes como é o caso dos trabalhos anteriores. Eles têm focado em dois problemas relacionados ao projeto de software: o problema de atribuição de responsabilidades a classes (*Class Responsibility Assignment* (CRA)) [10, 90] e o problema de aplicação de padrões de projeto à arquitetura [79].

Simons *et al.* [90] usam MOEA e agentes de software para auxiliar o arquiteto de software a encontrar o melhor projeto OO para um dado software. Os casos de uso são usados como ponto de partida. Ações (verbos) nos casos de uso são transformadas em métodos, e dados (substantivos) são transformados em atributos. Os métodos e atributos

são agrupados em classes. Cada classe deve ter pelo menos um método e um atributo, e nenhum método ou atributo pode estar em mais de uma classe.

A representação do problema é baseada em objetos, a qual não exige um mapeamento genótipo-fenótipo para a manipulação da solução pelo algoritmo de busca. Nessa representação, cada indivíduo (solução) é um projeto que contém todos os métodos e os atributos (e sua distribuição em classes). Para mutação, um conjunto de métodos e/ou atributos é selecionado e realocado para uma classe diferente. Para o cruzamento, dois indivíduos pais são escolhidos aleatoriamente na população. Em seguida, os atributos e os métodos são escolhidos aleatoriamente e trocados entre os dois, com base em sua posição de classe nos indivíduos. Uma restrição é que cada classe do projeto deve conter, pelo menos, um atributo e um método. Assim, a troca de posição só pode ocorrer onde a troca de um atributo ou método de outra classe não deixará a classe sem atributos ou métodos. O *fitness* é calculado com base em métricas de acoplamento e coesão.

A busca evolutiva é realizada por um agente de software. Simons [90] sugere que uma busca multiobjetivo global é desnecessária e, a busca deveria ser reduzida para os "projetos candidatos mais úteis e interessantes". Então, primeiro é realizada uma busca global usando o NSGAII a fim de alcançar esses projetos isolando zonas distintas do espaço de busca e, em seguida, realiza-se uma busca local dentro destas zonas. A busca local é realizada utilizando um GA mono-objetivo que só considera o acoplamento nos cálculos de *fitness*. O arquiteto recebe os resultados dessa busca local e pode especificar, manualmente, limiares (*thresholds*) de diversidade para controlar quando uma zona é isolada, ou um agente pode fazê-lo automaticamente.

Bowman *et al.* [10] aplicaram um GA multiobjetivo, denominado MOGA, para ajudar no problema CRA. O problema CRA é semelhante ao problema estudado por Simons e seus colaboradores, já que tenta resolver a atribuição de responsabilidades (métodos) em classes e a interação entre as classes. O *fitness* de um indivíduo também é medido por meio de métricas de acoplamento e coesão. O algoritmo fornece um *feedback* interativo para um arquiteto em vez de produzir um projeto por completo. MOGA recebe como entrada um diagrama de classes e as restrições definidas pelo usuário sobre o que pode

ou não mudar no diagrama de classes. O diagrama de classes é então avaliado e possíveis melhorias são sugeridas. MOGA finalmente fornece soluções alternativas para o usuário.

Informações sobre acoplamento são extraídas de modelos de interação e de contrato de operações expressos em OCL. A função de *fitness* baseia-se em cinco métricas, sendo uma de coesão e quatro de acoplamento. O algoritmo multiobjetivo utilizado é o SPEA2 [103]. Cada indivíduo é representado por um vetor de inteiros, no qual cada gene representa um membro de classe e seus valores denotam sua associação com uma classe. O principal mecanismo da busca é obter modelos de domínio melhores e, portanto, move métodos, atributos e associações de uma classe para outra, o que afeta as métricas de acoplamento e coesão. Além disso, o algoritmo inclui ou exclui classes, mas não altera dependências entre métodos e atributos, não inclui ou remove métodos, atributos ou relacionamentos e não move métodos abstratos. Existem ainda duas restrições: classes não podem ser vazias e devem estar envolvidas em pelo menos uma dependência, como cliente ou servidor.

O trabalho de Räihä *et al.* [79] está num nível de abstração um pouco mais alto do que os dois trabalhos anteriores. O trabalho propõe uma técnica baseada em GA que otimiza a arquitetura adicionando padrões arquiteturais. O problema da síntese de arquitetura de software é modelado como um problema combinatorial: "Dado um conjunto de padrões arquiteturais, como encontrar a configuração ótima de tais padrões para um sistema não trivial?". As soluções geradas no processo evolutivo são avaliadas segundo dois atributos de qualidade: modificabilidade e eficiência.

A arquitetura inicial, fornecida como entrada para o GA, é codificada em uma coleção de supergenes que representam todas as operações do sistema. Cada supergene é um vetor de inteiros que codifica informações sobre uma operação tais como: a qual classe a operação pertence, qual o seu tipo, qual interface a operação implementa, a qual padrão de projeto está associada e operações que ela chama. Logo, a representação não é dirigida à classes, mas sim à operações, já que os padrões de projeto utilizados são centrados em operações.

A mutação consiste em adicionar ou remover padrões de projeto e/ou estilos arquiteturais. Dentre os padrões de projeto utilizados estão *server*, *façade*, *mediator*, *strategy*,

adapter, *template method* e *interface*. Além disso, dois estilos arquiteturais podem ser adicionados por meio do operador de mutação: os estilos *message dispatcher* e cliente-servidor. O cruzamento entre os indivíduos consiste em mesclar duas arquiteturas sem quebrar as instâncias de padrões existentes no projeto. O *fitness* de cada indivíduo é medido com base em métricas de qualidade, como coesão, acoplamento, etc.

Em um estudo empírico [80], Räihä *et al.* observaram que o operador de cruzamento implementado de uma maneira aleatória não oferece uma vantagem significativa na otimização de arquitetura de software. Por isso, propuseram o operador de cruzamento complementar que seleciona pais complementares, a fim de produzir descendentes com as melhores partes de cada indivíduo pai para cada um dos dois atributos de qualidade considerados. Por exemplo, o operador seleciona um pai com boa modificabilidade e outro com boa eficiência, na esperança de que a descendência possa herdar, pelo menos em certa medida, ambos atributos de qualidade desejáveis. Em comparação com o cruzamento tradicional no qual os pais são escolhidos aleatoriamente, os resultados experimentais sugerem que o cruzamento complementar oferece mais versatilidade nas arquiteturas produzidas e permite soluções mais complexas, levando à valores de *fitness* significativamente melhores [81]. Em outro estudo [82], os resultados demonstraram que MOEAs têm potencial para produzir um bom conjunto de arquitetura, considerando eficiência e modificabilidade.

3.3 Considerações Finais

Sumarizando os trabalhos listados na Tabela 3.1, pode-se afirmar que a maioria (8 dentre 10) utiliza algoritmos evolutivos, seja GA, programação genética ou algum MOEA. Dentre todos os trabalhos, é possível perceber também que 70% deles (7 dentre 10) utiliza otimização multiobjetivo (de 2 a 5 objetivos). Nesse caso, os MOEAs utilizados são o NSGAII e o SPEA2.

Como se pode observar, alguns trabalhos relacionados adotam representações diretas para a arquitetura de software e outros realizam um mapeamento genótipo-fenótipo. Na representação direta, a representação natural do projeto é o alvo dos operadores de busca.

Isso indica que não há um tipo de representação padrão a ser adotada para o problema em questão.

Apesar de todos os trabalhos estarem relacionados a arquitetura, nenhum está ligado a LPS. Além disso, todas as funções de *fitness* baseiam-se em métricas convencionais de projeto OO, especialmente as de coesão e de acoplamento, e em alguns requisitos não funcionais, como confiabilidade e custo.

Portanto, apesar de haver alguns trabalhos da literatura que abordam a aplicação de técnicas de otimização multiobjetivo no projeto de arquitetura de software, eles não levam em conta o projeto de PLA, não usam métricas específicas para as características de LPS, tais como variabilidades e características associadas aos elementos arquiteturais. Nesse sentido, é necessário definir uma nova abordagem de otimização específica para o contexto de LPS e que use métricas mais apropriadas para esse contexto, de forma a apoiar os arquitetos na atividade de projeto de PLA. Essa abordagem é introduzida no próximo capítulo. É preciso também definir qual o melhor tipo de representação a ser adotada para resolver o problema e propor operadores de busca específicos para o problema com base nessa representação.

CAPÍTULO 4

ABORDAGEM DE OTIMIZAÇÃO MULTIOBJETIVO PARA PROJETO DE PLA

O foco deste trabalho está em utilizar técnicas baseadas em busca para a definição e a avaliação de PLA. Entretanto, como mencionado no Capítulo 3, até o presente momento, não existem trabalhos que aplicam algoritmos de busca visando a avaliação e definição do projeto de PLA.

Há diversos fatores envolvidos no projeto de PLA, por isso esta é uma atividade que demanda grande esforço humano. Logo, dispor de uma abordagem sistemática e automatizada para avaliar e melhorar o projeto de PLA pode apoiar o arquiteto na realização dessa atividade. Tal abordagem deve encontrar um balanço de prioridades a serem estabelecidas para os fatores envolvidos no projeto. A abordagem deve tratar o projeto de PLA como um problema de otimização com vários objetivos para os quais pode não existir uma única solução possível, mas sim um conjunto de soluções que apresentam o melhor compromisso entre os objetivos envolvidos. Nesse contexto, o uso de tal abordagem permite melhorar o espaço de soluções de PLA, as quais o arquiteto pode avaliar antes de decidir qual delas será adotada para a LPS.

Dessa forma, este capítulo tem como objetivo apresentar uma abordagem para otimização de projeto de PLA baseando-se em algoritmos de busca. Inicialmente, foi realizado um estudo para verificar a possibilidade de aplicar abordagens existentes de projeto de software baseado em busca no contexto de projeto de PLA. Os principais detalhes e resultados do estudo são apresentados na Seção 4.1. Considerando as lições aprendidas nesse estudo, a abordagem para otimização de projeto de PLA proposta nesta tese é apresentada na Seção 4.2. Na sequência, na Seção 4.3 é descrita uma instanciação da abordagem proposta utilizando MOEAs. Finalmente, na Seção 4.4, é apresentada uma ferramenta para automatizar a aplicação da abordagem proposta.

4.1 Aplicação de abordagens existentes no projeto de PLA

Como ponto de partida, foi desenvolvido um estudo exploratório [19] para investigar se é adequado o uso de abordagens de projeto de software baseado em busca existentes para otimizar PLAs. Para fazer isso, as abordagens foram estendidas para incluir as métricas de LPS CRSU e CPSU (Seção 2.4.1) com o intuito de avaliar a coesão das PLAs. O estudo foi desenvolvido utilizando a LPS Arcade Game Maker (AGM) [45]. Somente operadores de mutação presentes nas abordagens existentes foram aplicados. Um algoritmo baseado no NSGAII utilizando cinco objetivos que incluem métricas convencionais e as métricas CPSU e CRSU. A configuração do estudo e os resultados detalhados estão incluídos no Apêndice A.

Os resultados mostraram que as soluções obtidas são mais estáveis e mais reusáveis que a PLA original. Além disso, as seguintes lições foram aprendidas:

- As métricas usadas não são sensíveis a modularização de características. As mutações aplicadas geraram soluções de melhor *fitness*, mas em alguns casos as mudanças realizadas no projeto de PLA não foram boas porque deixaram algumas características mais espalhadas e entrelaçadas.

- As métricas CRSU e CPSU não são sensíveis às mudanças em PLAs. Em alguns casos há mudanças significativas nas PLAs que não foram capturadas por essas métricas. Mudanças internas aos componentes que melhoram a sua coesão não são detectadas porque não afetam diretamente as taxas de satisfação e utilização dos serviços que constituem a essência da definição dessas métricas.

- Para maximizar os benefícios alcançados, as métricas e os operadores de busca utilizados devem ser adequados ao estágio do projeto de PLA fornecido como entrada.

- Os operadores de busca devem ser sensíveis ao estilo arquitetural utilizado no projeto de PLA a ser otimizado, a fim de evitar que as soluções geradas infrinjam regras do estilo empregado.

Apesar de os resultados apontarem que abordagens existentes podem melhorar o projeto de PLA, os resultados seriam ainda melhores se pontos relacionados às lições aprendidas fossem considerados. As duas métricas de LPS não atendem a esses pontos. No entanto, o estudo mostrou que o uso de algoritmos evolutivos multiobjetivos é adequado para o contexto de projeto de PLA baseado em busca.

Assim, este estudo serviu para apontar que havia necessidade de inclusão de métricas adicionais, específicas para LPS, e de operadores de busca específicos para o problema. Nesse contexto, foi proposta a abordagem de otimização concebida neste trabalho, a qual é descrita a seguir.

4.2 MOA4PLA

MOA4PLA (*Multi-Objective Approach for Product-Line Architecture Design*) é uma abordagem sistemática e automatizada que usa algoritmos de busca multiobjetivos para avaliar e melhorar o projeto de PLA em termos de princípios básicos de projeto, modularização de características e extensibilidade de LPS. Esta abordagem produz um conjunto com as soluções que têm o melhor *trade-off* entre os vários objetivos otimizados.

MOA4PLA tem quatro principais aspectos: (a) engloba um processo para conduzir a otimização do projeto de PLA por meio de algoritmos multiobjetivos (Seção 4.2.1); (b) inclui um metamodelo que representa as PLAs a fim de viabilizar a manipulação da PLA por parte dos algoritmos (Seção 4.2.2); (c) trata a modularização de características por meio de um novo operador de busca (Seção 4.2.3); e, (d) propõe um tratamento multiobjetivo para o problema de projeto de PLA, incluindo métricas dirigidas a características (Seção 4.2.4).

MOA4PLA foi concebida com o intuito de aplicar diferentes algoritmos multiobjetivos no processo de otimização. Tal decisão se justifica porque, em geral, um mesmo problema pode ser eficientemente resolvido por vários algoritmos que seguem diferentes mecanismos para efetuar a busca pela solução. O principal foco da MOA4PLA é a avaliação e a melhoria do projeto de PLA, considerando múltiplos objetivos (métricas), independentemente

do algoritmo de busca adotado. Sendo assim, trata-se de uma abordagem genérica no que diz respeito ao algoritmo multiobjetivo a ser empregado.

MOA4PLA é independente tanto do método adotado para mapear as características nos elementos arquiteturais, como do método usado para representar as variabilidades na PLA. O foco desta abordagem é a análise estática automatizada de PLA em nível estrutural representada por diagramas de classes UML estereotipados.

4.2.1 Processo da MOA4PLA

O processo da MOA4PLA está representado na Figura 4.1. Lembrando dos dois ingredientes chave necessários para atacar um problema usando algoritmos de busca [41], a representação do problema é gerada na atividade Construction of the PLA Representation e a função objetivo é definida na atividade Definition of the Evaluation Model. Então, na terceira atividade um algoritmo de busca é aplicado para resolver o problema e, na quarta e última atividade, a saída da MOA4PLA é apresentada para o arquiteto. Essas atividades são explicadas a seguir.

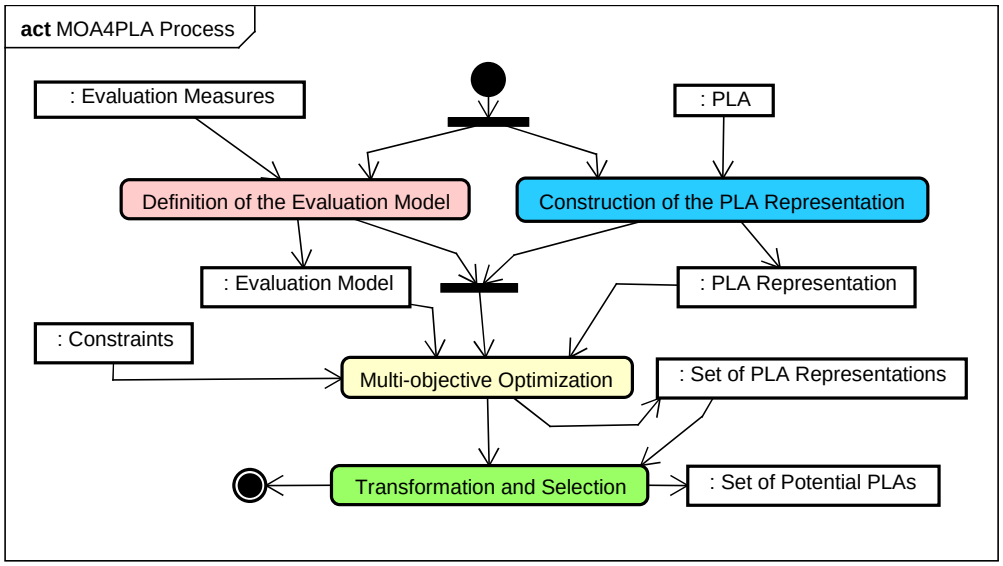


Figura 4.1: Processo da MOA4PLA

4.2.1.1 Construção da Representação da PLA

A entrada para a atividade Construction of the PLA Representation é o projeto completo da PLA modelado em um diagrama de classes. O diagrama representa a visão estrutural estática da PLA e deve conter, pelo menos, classes com seus respectivos atributos e métodos, interfaces e componentes. Além dos elementos arquiteturais, o diagrama de entrada deve conter informações sobre as variabilidades da LPS incluindo os pontos de variação e seus variantes. Além disso, nesse diagrama, as características devem estar associadas aos elementos arquiteturais utilizando estereótipos. Esta última informação permitirá que os operadores de busca (Seção 4.2.3) sejam aplicados adequadamente.

O objetivo dessa atividade é, a partir do projeto de PLA fornecido como entrada (PLA), instanciar a solução inicial a ser utilizada pelo algoritmo de busca como ponto de partida para o processo evolutivo (atividade Multi-Objective Optimization). Por isso, o projeto precisa estar num formato que seja manipulável pelo algoritmo de busca. Esse formato é, na verdade, a representação do problema que é abordada na Seção 4.2.2. Sendo assim, a saída desta atividade é uma representação da PLA (PLA Representation) contendo todos os elementos arquiteturais, seus inter-relacionamentos, todos os pontos de variação, variantes e características associadas aos elementos arquiteturais.

4.2.1.2 Definição do Modelo de Avaliação

Na atividade Definition of the Evaluation Model, o arquiteto deve definir quais medidas devem ser utilizadas no modelo de avaliação (Evaluation Model) que será utilizado no processo de otimização. Então, de acordo com as necessidades do projeto da LPS, o arquiteto pode escolher métricas que atendam tais necessidades. Assim, as métricas constituem os objetivos a serem otimizados pela abordagem. Vale ressaltar que o uso de diferentes métricas apoia o arquiteto na análise de *trade-off* entre os diferentes atributos de qualidade e que algumas dessas métricas podem ser conflitantes.

Em problemas da Engenharia de Software cujas soluções usando técnicas de SBSE têm sido amplamente estudadas nos últimos anos, como, por exemplo, teste de software,

a definição de quais métricas utilizar como função objetivo é uma tarefa já bastante explorada e de fácil resolução. No entanto, para projeto de PLA baseado em busca, a definição do modelo de avaliação ainda é um campo em aberto e pode ser difícil para o arquiteto selecionar as métricas para compor o referido modelo.

Relembrando, o principal objetivo da MOA4PLA é alcançar projetos de PLAs que sejam extensíveis, modulares e reusáveis, o que contribui para diminuir o esforço de manutenção e evolução da LPS. Por isso, além de otimizar princípios básicos de projeto tais como coesão e acoplamento, MOA4PLA tem como objetivo otimizar a modularização de características e a extensibilidade da LPS. Quanto maior a modularização de uma arquitetura, maior a sua capacidade de reúso. Além disso, se o conjunto das características priorizadas em módulos não variar com o tempo, a modularização também propiciará a estabilidade da arquitetura e, consequentemente, favorecerá a manutenibilidade.

Após uma investigação a respeito de métricas que atendem a esses objetivos, foi proposto um conjunto mínimo de métricas para serem disponibilizadas na MOA4PLA visando a facilitar o trabalho do arquiteto no momento da definição do modelo de avaliação.

Métricas convencionais [11, 15, 62, 89] (Tabela 2.1) podem fornecer indícios a respeito de diferentes atributos de qualidade da PLA, tais como coesão, acoplamento, tamanho da arquitetura e elegância do projeto. Entretanto, métricas específicas para LPS (Seção 2.4.1) são mais apropriadas. A métrica de extensibilidade de LPS [69] permite analisar o quão extensível é uma PLA. As MSI [83] podem ser utilizadas para analisar a modularização das características de uma PLA [33]. Assim, essas métricas (LCC, CDAC, CDAI, CDAO, CIBC, IIBC e OOBC) permitem medir coesão e acoplamento baseados em características. Dado que as MSI são aplicadas na MOA4PLA com o intuito de medir a modularização de características, elas são chamadas, daqui em diante, de métricas dirigidas à características.

O conjunto de métricas arquiteturais disponíveis na MOA4PLA é inicialmente composto pelas métricas convencionais apresentadas na Tabela 2.1, pela métrica de extensibilidade de LPS e pelas métricas dirigidas a características. Futuramente, outras métricas convencionais podem ser adicionadas à MOA4PLA, como, por exemplo, uma adaptação da métrica de qualidade da modularização de *clusters* (*Modularization Quality*) [72] para

o contexto de projeto de PLA, a qual agrega dois objetivos a fim de atingir alta coesão e baixo acoplamento entre *clusters*, atribuindo pesos para cada um desses objetivos.

MOA4PLA também pode ser estendida para incluir outras métricas de LPS que forneçam indicadores de modularidade, tais como: *Internal-ratio Feature Dependency (IFD)* e *External-ratio Feature Dependency (EFD)* [4] para medir a coesão de uma característica; e/ou PSU e RSU [94] para medir a coesão dos componentes de uma PLA com base em seus serviços fornecidos e requeridos.

A saída dessa atividade é o modelo de avaliação composto pelos objetivos definidos pelo arquiteto para serem usados na otimização. Cada objetivo é formado por uma função de *fitness* que estabelece como as métricas, escolhidas pelo arquiteto, serão coletadas em cada solução de projeto de PLA, considerando a representação adotada para o problema.

4.2.1.3 Otimização Multiobjetivo

A atividade Multi-Objective Optimization, tem três entradas: (i) o modelo de avaliação (Evaluation Model), que contém os valores das métricas selecionadas pelo arquiteto na atividade Definition of the Evaluation Model; (ii) a representação da arquitetura gerada na primeira atividade (PLA Representation); e (iii) as restrições que serão utilizadas no processo de otimização. Essas restrições estão relacionadas à particularidades que uma PLA deve satisfazer e são definidas na Seção 4.2.5.

O projeto de PLA original é otimizado durante o processo de otimização multiobjetivo, respeitando as restrições estabelecidas. Cada alternativa de projeto de PLA obtida é avaliada com base no modelo de avaliação. Como saída, é gerado um conjunto com as alternativas de projeto de PLA que têm o melhor *trade-off* entre os objetivos presentes no modelo de avaliação (Set of PLA Representations).

Diferentes algoritmos podem ser usados para automatizar essa atividade, desde um algoritmo exato (que retornará sempre a mesma solução dada uma mesma entrada), passando por algoritmos de busca tradicionais, como Simulated Annealing e Hill Climbing, até algoritmos meta-heurísticos, tais como ACO, PSO, MOEAs (Seção 3.1.2).

Räihä [78] relata em seu trabalho que foi possível perceber rapidamente que a tentativa de resolver simultaneamente tanto o problema CRA como o problema de otimizar a arquitetura em um nível mais alto como a aplicação de padrões de projeto era audaciosa e de difícil resolução. Os estudos de Simons [88] e de Bowman *et al.* [10] também indicam claramente que o problema CRA é difícil de resolver, mesmo sem a carga de problemas adicionais de projeto. Assim, o algoritmo de busca deve trabalhar em duas etapas - primeiro resolver o problema CRA, e, em seguida, começar a introduzir os padrões, ou apenas concentrar-se na arquitetura de alto nível e padrões [78]. Desse modo, entende-se que a MOA4PLA deve ser aplicada seguindo tais ideias a fim de obter resultados satisfatórios no processo de otimização.

Para isso, num primeiro momento, a MOA4PLA utiliza somente operadores de busca para resolver o problema CRA, além do operador de busca para modularizar características que é definido na Seção 4.2.3. Operadores de busca para aplicar padrões de projeto, estilos arquiteturais ou ainda para trabalhar especificamente com *frameworks* podem ser incluídos na abordagem posteriormente.

4.2.1.4 Transformação e Seleção

A entrada da atividade Transformation and Selection é o conjunto de soluções (Set of PLA Representations) gerado como saída da atividade anterior. Essas soluções consistem das alternativas de projeto de PLA encontradas pela MOA4PLA. Nessa atividade, o conjunto de soluções (Set of Potential PLAs) é convertido para uma forma legível para o arquiteto.

O arquiteto atua diretamente na seleção de qual das alternativas geradas pela abordagem será adotada para a LPS. Ele deve, então, escolher uma das alternativas geradas desse conjunto para ser adotada como a PLA da LPS. Para isso, pode-se priorizar alguma(s) métrica(s) utilizada(s) no modelo de avaliação, ou ainda, escolher a solução que tem o melhor *trade-off* entre os objetivos usados. Essa decisão pode ser influenciada por diferentes fatores, tais como, restrições econômicas ou do negócio. A Seção 5.7 apresenta exemplos de como o arquiteto pode selecionar uma solução.

4.2.2 Representação do Problema

Independentemente do algoritmo de busca utilizado, é necessário definir uma representação apropriada para um projeto de PLA, pois essa representação impacta na implementação de todos os estágios dos algoritmos de busca. Nenhum trabalho relacionado usa a representação da PLA em diagrama de classes. Então, foi preciso analisar as possibilidades de adaptação das representações adotadas nos trabalhos relacionados [10, 79, 88] para o contexto de projeto de PLA. A referida análise foi publicada em [20] e está incorporada a esta tese no Apêndice B.

As representações utilizadas nos trabalhos relacionados incluem árvores, grafos direcionados, vetores de inteiros, especificações usando ADLs e representações baseadas em objetos. A maioria delas não é natural e não é facilmente adaptável para representar variabilidades e características associadas aos elementos arquiteturais no nível de classes. Além disso, essas representações podem tornar complexa a implementação dos operadores de busca.

A representação baseada em objetos [88] é a mais facilmente extensível para o contexto de PLA, além de ser mais adequada, intuitiva e natural para o problema. Ademais, uma representação baseada em objetos permite utilizar a abordagem direta, na qual a própria arquitetura é o alvo de otimização do algoritmo de busca, facilitando o uso da abordagem já que um mapeamento genótipo-fenótipo não é necessário. No entanto, ao adotar a abordagem direta e o uso de GA fica difícil assegurar a exatidão da solução quando o operador de cruzamento é aplicado [43]. Desse modo, é necessário garantir a consistência da arquitetura após a sua manipulação pelo operador de cruzamento.

Ainda assim, a referida representação baseada em objetos não inclui características peculiares de PLAs, tais como, pontos de variação, variantes e características associadas aos elementos arquiteturais. Dessa forma, propõe-se uma nova representação para resolver o problema de otimização de projeto de PLA. Essa representação consiste do metamodelo apresentado na Figura 4.2, o qual é uma extensão do metamodelo proposto em [13] e foi adequado às necessidades da MOA4PLA.

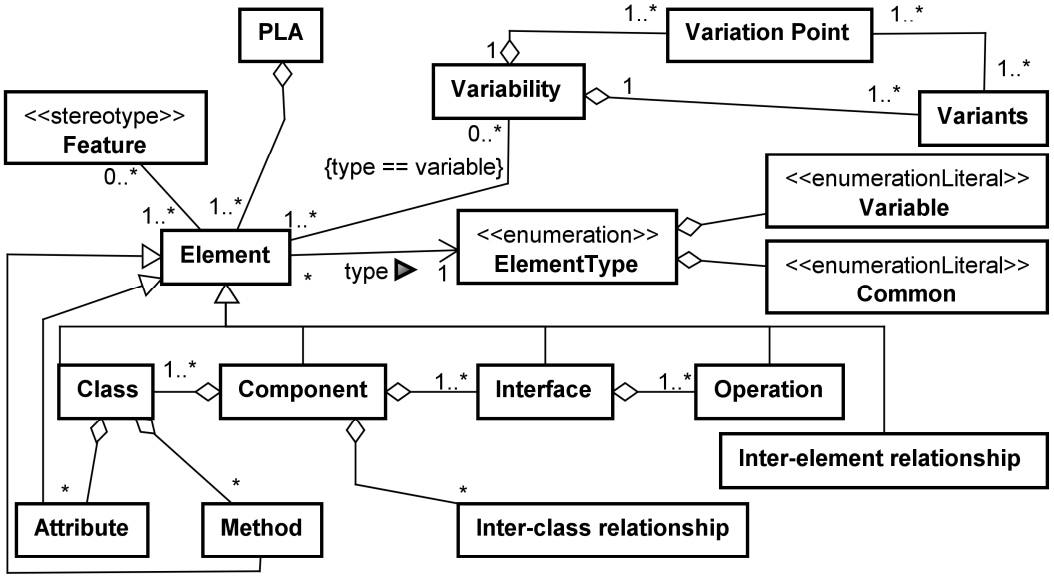


Figura 4.2: Metamodelo da PLA

Uma PLA contém elementos arquiteturais que podem ser componentes, interfaces, operações de interfaces e seus inter-relacionamentos. Os componentes são compostos por classes e relacionamentos interclasses. Cada elemento arquitetural está associado à característica(s) por meio de estereótipos. Um elemento pode ser comum a todos os produtos da LPS ou variável, estando presente só em algum(ns) produto(s) da LPS. Os elementos variáveis estão associados a pontos de variação e suas respectivas variantes.

A Figura 4.3 mostra um exemplo do projeto de PLA da LPS Mobile Media, o qual pode ser representado pelo metamodelo proposto. Nesse exemplo, os componentes estão representados por pacotes. É possível ver que os elementos arquiteturais estão associados à características usando estereótipos tais como, *Label Photo*, *Favourites*, *Copy*, *Sorting*, e *SMSTransfer*. Essa PLA foi concebida utilizando o SMarty Components [23], um método que permite o projeto de PLAs baseadas em componentes que seguem o estilo em camadas e, no qual, as variabilidades são modeladas em notas UML usando a abordagem SMarty (Seção 2.2). Na Figura 4.3, pode-se perceber que as variabilidades da LPS estão associadas às classes *Media* e *Entry*. Os estereótipos <<business interface>>, <<system interface>> e <<core>> são utilizados pelo método para indicar interfaces de negócio, interfaces de sistema e elementos arquiteturais que fazem parte do domínio da LPS, respectivamente.

Adotar uma representação direta implica em: (i) propor operadores de busca específicos para a representação utilizada para o projeto de PLA e, (ii) estabelecer restrições para manter a consistência do projeto de PLA durante o processo de otimização. Esses dois assuntos são abordados nas duas próximas subseções.

4.2.3 Operadores de busca

Esta seção tem como objetivo apresentar operadores de busca aplicáveis à representação adotada na MOA4PLA. O operador de busca que visa a melhorar a modularização das características da PLA proposto para a MOA4PLA é definido na Seção 4.2.3.2. No entanto, primeiramente, são apresentados os operadores utilizados nos trabalhos relacionados que foram adaptados para o contexto da MOA4PLA.

4.2.3.1 Operadores convencionais

MOA4PLA inclui cinco operadores convencionais: Move Method, Move Attribute, Add Class, Move Operation e Add Component, assim denominados porque foram concebidos para o contexto de projeto de software baseado em busca. Os três primeiros foram implementados como operadores de mutação em abordagens de projeto de software baseado em busca [10, 88]. A adição de classes é necessária, uma vez que alguma classe que permitiria uma melhor atribuição de responsabilidades no projeto como um todo pode não ter sido identificada no projeto inicial. Considerando que as PLAs alvo da MOA4PLA são baseadas em componentes, os quais possuem interfaces que oferecem operações, dois operadores adicionais foram derivados de Move Method e Add Class: Move Operation and Add Manager Class, respectivamente. Esses operadores foram aplicados pela primeira vez no contexto de PLA em [19].

Durante a adaptação para o contexto de projeto de PLA, os operadores foram preparados para serem aplicados em PLAs seguindo o estilo arquitetural em camadas. Assim, eles evitam que elementos arquiteturais de uma camada sejam movimentados para outra camada. A identificação da camada é feita analisando o sufixo do nome do componente, por exemplo, Mgr, Ctrl ou GUI. A seguir, cada operador é descrito.

Cada um dos operadores convencionais é descrito a seguir:

- Move Method: Esse operador move um método selecionado aleatoriamente entre duas classes, também selecionadas aleatoriamente. Além de tirar o método da classe original e inserí-lo na classe destino, o operador adiciona uma associação bidirecional entre essas duas classes. O operador só é aplicado se a classe de origem não fizer parte de uma hierarquia de herança, se tiver mais de um atributo e mais de um método e, se o método não estiver associado com a característica da variabilidade vinculada à classe, caso ela seja um ponto de variação ou uma variante.

- Move Attribute: Esse operador move um atributo escolhido aleatoriamente entre duas classes aleatoriamente selecionadas. Ele também adiciona uma associação bidirecional entre essas duas classes. O operador só é aplicado se a classe de origem não fizer parte de uma hierarquia de herança, se tiver mais de um atributo e mais de um método e, se o atributo não estiver associado com a característica da variabilidade vinculada à classe, caso ela seja um ponto de variação ou uma variante.

- Add Class: Esse operador seleciona uma classe qualquer de um componente e move um atributo ou um método (selecionado aleatoriamente) para uma nova classe que é criada em um componente também selecionado aleatoriamente. Além disso, caso as classes sejam do mesmo componente, o operador adiciona uma associação bidirecional entre essas duas classes. Quando os componentes são diferentes, uma interface do componente destino é adicionada como interface requerida para o componente de origem da classe. Esse operador só é aplicado se: (i) a classe não fizer parte de uma hierarquia de herança; (ii) a classe tiver mais de um atributo e mais de um método; e (iii) os componentes das interfaces forem da mesma camada arquitetural.

- Move Operation: Esse operador move uma operação selecionada aleatoriamente entre duas interfaces, também selecionadas aleatoriamente. Além disso, se os componentes que realizam as interfaces forem diferentes, o operador adiciona a interface destino

como interface realizada pelo componente da interface original da operação movida. A operação só é movida se os componentes das interfaces forem da mesma camada arquitetural.

- Add Component: Esse operador seleciona aleatoriamente uma operação de uma interface de um componente e move essa operação para uma nova interface de um novo componente que é criado na arquitetura. Além de tirar a operação da interface original e inserí-la na nova interface, o operador adiciona a nova interface como interface realizada pelo componente da interface de origem. O novo componente é criado na mesma camada (isto é com o mesmo sufixo) do componente de origem.

A PLA da Mobile Media, apresentada na Figura 4.3, é utilizada para ilustrar a aplicação dos operadores de busca. Tomando como exemplo o operador Move Operation, e considerando que a operação *hasMemory* da interface *IMediaMgt* tenha sido selecionada para ser movida para a interface *ILabelFiles*. O resultado da aplicação desse operador pode ser visto na Figura 4.4, a qual apresenta a parte do projeto que foi afetada pelo operador.

Os operadores de busca convencionais podem provocar modificações no projeto de uma PLA de forma a alterar a coesão e o acoplamento, alterar tamanho de classes e interfaces, interferir na elegância do projeto como um todo, impactar benéfica ou maleficamente a extensibilidade da PLA. No exemplo dado, a extensibilidade da PLA não foi alterada, no entanto, poderia ter sido caso o operador Move Method tivesse movido um método abstrato de uma classe que é ponto de variação para uma outra que não é ponto de variação. Os operadores convencionais podem ainda impactar o valor das métricas dirigidas a características quando a movimentação de elementos arquiteturais altera, por exemplo, a difusão de características ou a sobreposição entre elas.

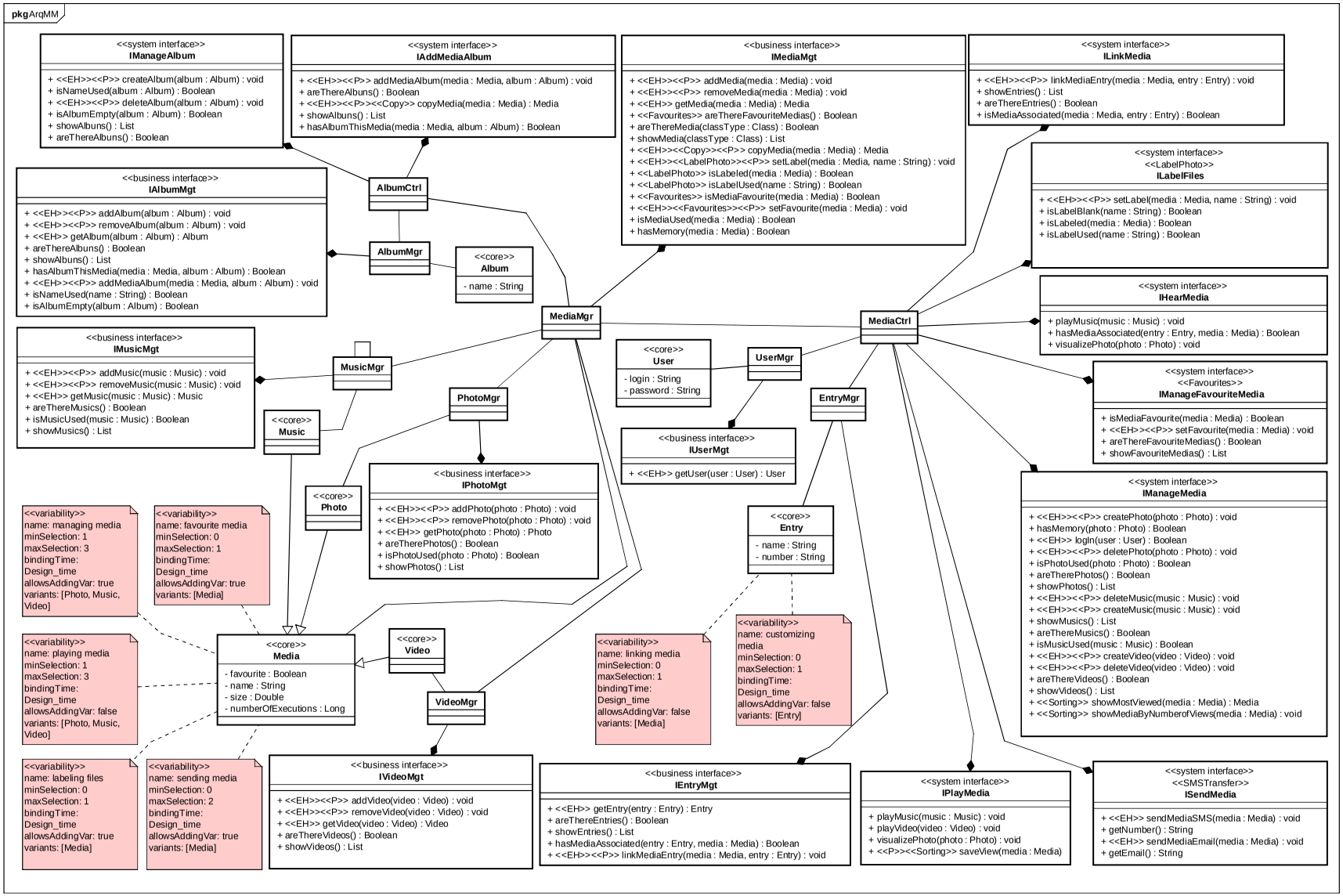


Figura 4.3: Arquitetura original da LPS Mobile Media

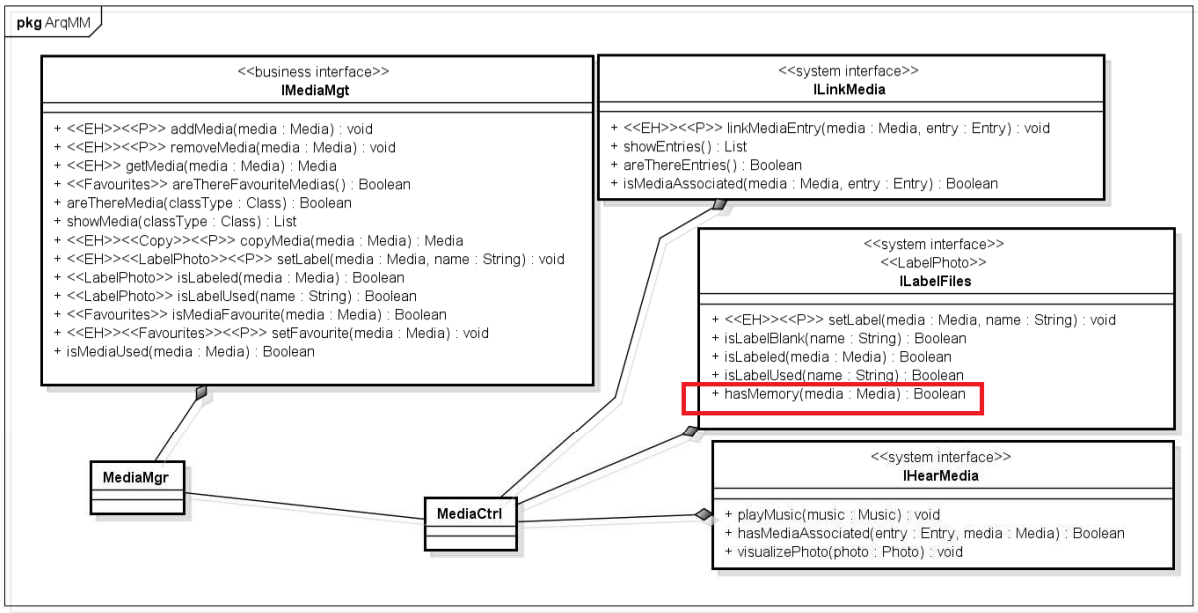


Figura 4.4: PLA da Mobile Media (parcial) após aplicação do operador Move Operation

4.2.3.2 Operador dirigido a características

O operador de busca denominado Feature-driven Operator faz parte da MOA4PLA e tem como objetivo modularizar uma característica que esteja entrelaçada com outra(s) em um componente selecionado aleatoriamente. Partindo do pressuposto de que uma característica de LPS costuma ser resolvida por um grupo de elementos arquiteturais, Feature-driven Operator seleciona aleatoriamente um componente, e, se esse componente tiver interface(s) associada(s) à diferentes características, selecionar uma dessas características para ser modularizada em um novo componente. Para isso, um novo componente é criado. Todos os elementos arquiteturais associados à característica sendo modularizada são movidos para o novo componente.

O pseudocódigo do operador é apresentado no Algoritmo 3, onde A é a arquitetura fornecida como entrada para o operador de busca. Na linha 4, um componente é selecionado aleatoriamente (c_x), e, se c_x tem elementos arquiteturais (interfaces, classes, operações, atributos e métodos) associados à diferentes características (linha 6), uma característica f_x é selecionada aleatoriamente para ser modularizada no componente c_z (linha 7). c_z é escolhido de acordo com o tamanho do conjunto c_f que contém todos os componentes da

arquitetura A associados à característica f_x. Quando o tamanho é igual a 0 (linha 9), um novo componente c_z é criado (linha 10). Quando o tamanho é igual a 1 (linha 13), c_z recebe o único componente de c_f (linha 14). Ou, se o tamanho for maior que um, c_z recebe um componente de c_f selecionado aleatoriamente (linha 16). Os elementos arquiteturais de c_x associados exclusivamente com f_x são movidos para c_z (linha 20). Então, na linha 21, um novo relacionamento é estabelecido entre c_z e o componente original de cada elemento arquitetural movido (linha 21). O novo componente c_z é criado na mesma camada arquitetural (isto é, com o mesmo sufixo) do componente de origem c_x e, somente são movidos os elementos arquiteturais provenientes de componentes da camada em questão.

Algorithm 3: Pseudocódigo do Feature-driven Operator

```
1  begin
2      C ← A.getAllComponents()
3      E ← set of architectural elements of A
4      c_x ← a randomly selected component from C
5      F_c ← c_x.getAllFeatures()
6      if F_c.size()>1 then
7          f_x ← a randomly selected feature from F
8          c_f ← A.getAllComponentsAssociatedWithFeature(f_x)
9          if c_f.size==0 then
10             c_z ← A.createComponent()
11             c_z.addFeature(f_x)
12         else
13             if c_f.size==1 then
14                 c_z ← c_f.get(0)
15             else
16                 c_z ← a randomly selected component from c_f
17         for each e ∈ E | (e ≠ c_z or e ∉ c_z) do
18             if e.getFeatures.size()==1 and e.isAssociatedWithFeature(f_x) then
19                 Move e to c_z
20                 A.addRelationship (c_z, original component of e)
21         end
22     return A;
23 end
```

A aplicação do Feature-driven Operator contribui para a obtenção de projetos de PLA com características menos espalhadas e menos entrelaçadas, o que também pode melhorar a coesão baseada em características dos componentes arquiteturais. Dessa forma, o operador também contribui para evitar a presença do *architectural bad smell Scattered Functionality* [36], no qual vários componentes de um sistema são responsáveis por realizar um interesse em alto nível - por exemplo, uma característica de LPS - violando o princípio de separação de interesses.

Tomando como base a PLA da Mobile Media (Figura 4.3), pode-se observar que o componente *MediaCtrl* tem interfaces associadas a diferentes características: *Favourites*, *SMSTransfer* e *LabelPhoto*. Para aplicar o Feature-driven Operator ao componente *MediaCtrl* visando a modularizar a característica *SMSTransfer*, o primeiro passo foi criar o componente *SMSTransferCtrl* (em destaque na Figura 4.5). Depois a interface *ISendMedia* foi movida para *SMSTransferCtrl* (também destacada na figura). O próximo passo foi estabelecer os relacionamentos entre *SMSTransferCtrl* e *EntryMgr, UserMgr* e *MediaMgr* e associá-lo à característica *SMSTransfer*. A Figura 4.5 mostra as partes da PLA que foram alteradas após a aplicação desse operador. Caso houvesse outros elementos arquiteturais associados à característica *SMSTransfer*, eles também seriam movidos para *SMSTransferCtrl* durante a aplicação desse operador.

Após a aplicação do Feature-driven Operator, os valores de LCC, CDAC, CDAI, OOBC e CIBC tendem a diminuir. No caso específico da característica *SMSTransfer*, houve uma alteração benéfica nos valores de LCC e CIBC dessa característica, indicando que a coesão dessa característica aumentou e a sua sobreposição com outras características diminuiu. Além disso, esse operador pode contribuir para desinchar componentes que eventualmente estejam sobrecarregados, como pode ser visto no exemplo em relação ao componente *MediaCtrl*. Desse modo, o operador gera impacto também sobre o valor das métricas de tamanho, elegância, coesão e acoplamento.

4.2.4 Modelo de avaliação

Durante o processo de busca, cada solução gerada pelo algoritmo utilizado, seja ele qual for, é avaliada de acordo com as funções objetivo definidas no modelo de avaliação e usando os conceitos de dominância de Pareto, sem a necessidade de estipular um peso para cada um dos objetivos. Isto permite alcançar, ao final do processo de busca, um conjunto de soluções com os melhores *trade-offs* entre os objetivos adotados. Como mencionado anteriormente, cada objetivo a ser incluído no modelo de avaliação deve ser escolhido pelo arquiteto a partir do conjunto de opções disponíveis, que envolvem métricas arquiteturais convencionais e específicas para LPS.

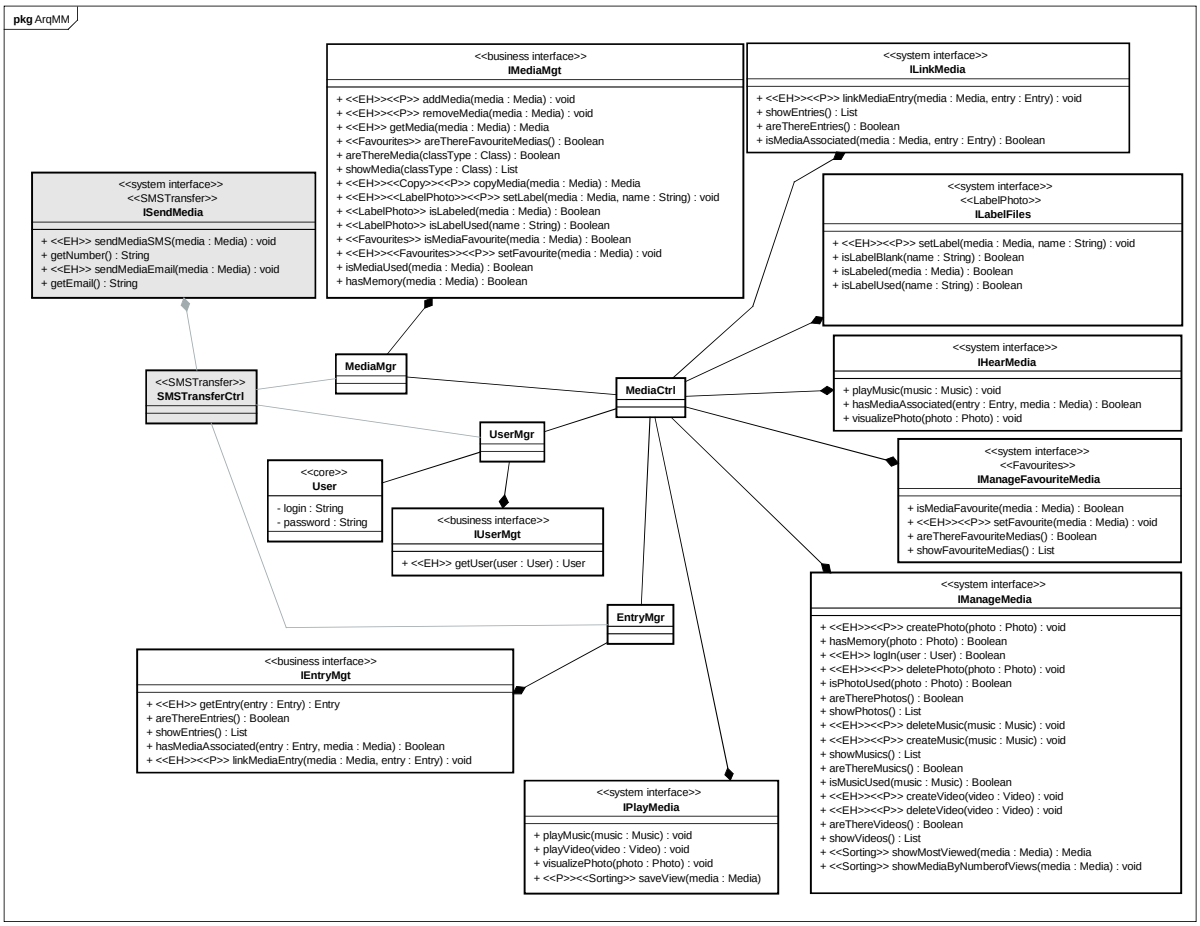


Figura 4.5: PLA da Mobile Media (parcial) após aplicação do Feature-driven Operator

Quatro funções objetivo estão disponíveis na MOA4PLA: $CM(pla)$, $FM(pla)$, $Ext(pla)$ e $Eleg(pla)$ (Equações 4.1 a 4.4). Todas são funções a serem minimizadas. A primeira é definida por meio de uma agregação de várias métricas convencionais (Tabela 2.1) de acordo com a Equação 4.1, onde c é o número de componentes, itf é o número de interfaces e cl é o número de classes de uma dada pla. $CM(pla)$ é composta pela soma de várias métricas convencionais. Nessa soma são usadas a média de DepPack de todos os componentes de pla e a média de NumOps de todas as interfaces do projeto. A métrica H foi contada inversamente porque pretende-se maximizar a coesão e minimizar as outras métricas da função. O objetivo dessa função é contribuir para alcançar soluções propensas a conter projetos adequados, dos pontos de vista de alta coesão, baixo acoplamento e maior reusabilidade.

$$CM(pla) = \sum_{i=1}^{c} DepIn + \sum_{i=1}^{c} DepOut + \sum_{i=1}^{cl} CDepIn + \sum_{i=1}^{cl} CDepOut + \frac{\sum_{i=1}^{c} DepPack}{c} + \frac{\sum_{i=1}^{itf} NumOps}{itf} + \frac{1}{\sum_{i=1}^{c} H} \tag{4.1}$$

A segunda função objetivo é uma agregação das várias métricas dirigidas a características (Seção 2.4.2). A função é definida na Equação 4.2, onde c é o número de componentes, itf é o número de interfaces, cl é o número de classes e f é o número de características de uma dada pla. $FM(pla)$ consiste do somatório da soma de cada métrica dirigida à características. Como as métricas são correlacionadas, a diminuição do valor de uma métrica implica na diminuição dos valores das demais. Essa função é usada para avaliar a modularização de características ocasionada pelo operador de busca Feature-driven Operator e tem como objetivo permitir que projetos com alta modularização de características sejam alcançados pela MOA4PLA.

$$FM(pla) = \sum_{i=1}^{c} LCC + \sum_{i=1}^{f} CDAC + \sum_{i=1}^{f} CDAI + \sum_{i=1}^{f} CDAO + \sum_{i=1}^{f} CIBC + \sum_{i=1}^{f} IIBC + \sum_{i=1}^{f} OOBC \tag{4.2}$$

$FM(pla)$ avalia o grau de modularização de uma PLA em termos das características sendo realizadas. Além do impacto negativo sobre a reusabilidade e manutenibilidade da PLA, uma modularização imprecisa de características pode levar a uma grande variedade de falhas de projeto, que vão desde emaranhamento e espalhamento de características até específicos *bad smells*, tais como, *feature envy* e *god class* [34].

A Equação 4.3 apresenta a terceira função objetivo $Ext(pla)$, a qual consiste do valor invertido da métrica ExtensPLA (Seção 2.4.1.2) para uma dada pla. Esse cálculo é feito para que seja possível maximizar a extensibilidade da PLA, lembrando que os objetivos são minimizados na MOA4PLA.

$$Ext(pla) = \frac{1}{ExtensPLA(pla)} \tag{4.3}$$

A última função objetivo é apresentada na Equação 4.4, cujo objetivo é melhorar a elegância do projeto de PLA alcançado em cada solução gerada pela MOA4PLA. Para isso deve-se minimizar os valores das métricas NAC, EC e ATMR (Tabela 2.1). Essa função é uma agregação dessas métricas de elegância.

$$Eleg(pla) = NAC(pla) + EC(pla) + ATMR(pla) \tag{4.4}$$

De posse dessas funções objetivo, o arquiteto pode selecionar de 1 até 4 objetivos para compor o modelo de avaliação a ser empregado durante o processo de busca por melhores alternativas de projeto para uma determinada PLA.

Essas funções objetivo constituem uma das contribuições do presente trabalho, as quais foram obtidas a partir de um estudo das métricas que seriam mais adequadas para o contexto de projeto de PLA baseado em busca. Entretanto, outros objetivos podem ser utilizados e incluídos na MOA4PLA.

4.2.5 Restrições

Durante o processo de otimização algumas restrições (Constraints) devem ser respeitadas. As restrições são importantes uma vez que as alterações provocadas pelos operadores de busca podem ocasionar projetos de PLA inconsistentes. As restrições incluídas na MOA4PLA são as seguintes:

- componentes, interfaces e classes não podem ser vazios;
- classes e interfaces devem estar envolvidas em pelo menos um relacionamento;
- um método abstrato não deve ser movido para outra classe se ele estiver em uma hierarquia de herança, pois pode haver métodos concretos que o implementem nas subclasses;
- os relacionamentos de generalização entre classes e suas subclasses devem ser mantidos em todas as soluções geradas;

- os elementos arquiteturais que sofreram ação dos operadores de busca devem permanecer associados às características originais, ainda que tenham sido movidos para um elemento arquitetural associado à uma característica diferente; e,
- os atributos e os métodos de classes que são pontos de variação ou variantes que estão associados com a característica da variabilidade associada não devem ser movidos.

Durante o processo de otimização, os operadores de busca são aplicados sobre a PLA original de forma aleatória e iterativa durante um certo número de avaliações de *fitness*. Cada nova solução alcançada é avaliada pelo modelo de avaliação e deve atender às restrições listadas anteriormente. Dessa forma, pretende-se obter como saída do processo evolutivo somente soluções consistentes, ou em outras palavras, soluções cuja semântica continue correta em relação à PLA original. Os aspectos de implementação dessas restrições são descritos na Seção 4.4.2.

4.2.6 Pressupostos

A concepção da MOA4PLA leva em consideração os seguintes pressupostos:

- o arquiteto é competente;
- os projetos de PLA fornecidos como entrada são baseados em componentes,
- todos os componentes têm interfaces as quais oferecem operações;
- todas as interfaces são fornecidas por, ao menos, um componente;
- o mapeamento de características aos elementos arquiteturais está correto e foi realizado no projeto de PLA utilizando estereótipos;
- todas as características da LPS têm a mesma importância e, portanto, têm o mesmo peso no momento da modularização por parte dos operadores de busca; e,
- uma solução que não atende a alguma das restrições apresentadas na Seção 4.2.5 é inválida. O tratamento dado a cada solução inválida está descrito na Seção 4.4.2.

4.3 Instanciação da Abordagem

Considerando que é preciso escolher um algoritmo de busca multiobjetivo para que a MOA4PLA possa otimizar o projeto de PLA, nesta seção são apresentadas as principais decisões tomadas a respeito de como a abordagem foi instanciada a fim de executar experimentos que apontem a sua efetividade para solucionar o problema proposto.

Dentre as possíveis formas de se resolver problemas de Engenharia de Software com vários objetivos, os algoritmos evolutivos multiobjetivos (MOEA) têm alcançado melhor desempenho que as outras alternativas [7, 35, 98]. No trabalho de Vergilio *et al.* [98] os autores trataram do problema de ordem de integração e teste de classes aplicando três tipos diferentes de algoritmos de busca multiobjetivos: o NSGAII, o algoritmo PACO (*Pareto Ant Colony Optimization*) que é um ACO multiobjetivo, e o *Multi-objective Tabu Search* que é baseado em busca Tabu. Dentre eles, o MOEA NSGAII apresentou os melhores resultados.

Especificamente no contexto de projeto de software baseado em busca, os MOEAs têm sido empregados com sucesso [10, 37, 61, 78, 88] (Seção 3.2). Portanto, há evidências de que MOEAs podem permitir que a MOA4PLA alcance resultados satisfatórios em relação aos objetivos traçados. Por isso, MOEAs baseados em GA foram escolhidos para a primeira instanciação da abordagem, dentre eles o NSGAII. Nada impede que no futuro, outro tipo de algoritmo de busca seja aplicado como, por exemplo, PACO ou *Multi-objective Tabu Search*.

4.3.1 O processo evolutivo

A população de um GA é composta por vários indivíduos. Cada indivíduo representa uma possível solução arquitetural para uma LPS. Ao longo do processo de otimização, vários novos indivíduos são criados por meio dos operadores evolutivos e durante o processo de seleção somente os indivíduos mais aptos permanecem na população para a próxima iteração. A aptidão (*fitness*) de cada indivíduo provém dos resultados de métricas que representam os objetivos. O processo de otimização é realizado até que um determinado número de avaliações de *fitness* seja atingido. Sendo assim, o uso de GAs inclui alguns

pontos tais como, a representação do problema, o cálculo do *fitness* e a definição dos operadores evolutivos. A representação do problema e as funções objetivos adotadas na MOA4PLA foram definidas nas Seções 4.2.2 e 4.2.4, respectivamente. Como mencionado na Seção 3.1.2, em algoritmos evolutivos, especialmente os derivados de GA, os dois principais operadores evolutivos são cruzamento e mutação de indivíduos.

Um operador de mutação realiza alguma modificação aleatória em uma única solução. Os operadores de busca definidos na Seção 4.2.3 fazem exatamente esse tipo de alteração, por isso, são aplicados como operadores de mutação, e assim são chamados daqui em diante. Esses operadores são utilizados também para criar a população inicial. Para criar a população inicial um operador de mutação, selecionado aleatoriamente, é aplicado à PLA original, até que a população esteja completa. Uma cópia original intacta da PLA fornecida pelo arquiteto também faz parte dessa população. Esse mesmo procedimento para a criação da população inicial foi usado em [81].

O operador de cruzamento é aplicado antes do operador de mutação e combina duas soluções a fim de gerar dois novos indivíduos (filhos). Räihä *et al.* comparam o cruzamento com uma situação na qual dois arquitetos fornecem projetos alternativos para uma PLA e decidem mesclar as duas alternativas pegando partes de cada projeto.

A otimização aplicando somente mutação alcança melhores resultados que a otimização que combina cruzamento tradicional com mutação [80]. No entanto, o operador de cruzamento complementar proposto em [81] alcança melhores resultados do que a otimização realizada usando somente mutação. Esse e outros operadores de cruzamento específicos para projeto de software [40, 90] não são diretamente aplicáveis ao projeto de PLA, mas podem ser adaptados para esse contexto. Ainda assim, eles não se preocupam com uma importante propriedade para PLAs: a modularização de características, que impacta diretamente na reusabilidade e na manutenibilidade da PLA [34].

Diante disso, considera-se que um operador de cruzamento específico para a modularização de características é uma contribuição mais valiosa para o projeto de PLA baseado em busca do que a adaptação de operadores existentes. Sendo assim, na próxima seção, propõe-se um novo operador de cruzamento dirigido à características para projeto de PLA.

4.3.2 Cruzamento dirigido a características

O operador de cruzamento que tem como objetivo melhorar a modularização de características de projetos de PLA é chamado de Feature-driven Crossover. Esse operador assume a premissa de que, em uma das duas soluções pais, uma determinada característica pode estar melhor modularizada. Por isso, o operador seleciona uma característica e então cria filhos a partir dos pais por meio da troca dos elementos arquiteturais (por exemplo, classes, interfaces e operações) que estão associados à característica selecionada.

O funcionamento desse operador é ilustrado genericamente na Figura 4.6 usando somente classes. As características aparecem associadas às classes por meio de estereótipos. As classes brancas de $Parent1$ e as classes azuis de $Parent2$ estão associados à característica A. As classes amarelas de $Parent1$ e as classes rosadas de $Parent2$ estão associadas à característica B. Supondo que a característica B tenha sido selecionada para a aplicação do operador, durante o cruzamento, as classes rosadas são movidas de $Parent2$ para $Child1$ e as classes amarelas são movidas de $Parent1$ para $Child2$. O projeto de $Child1$ é complementado com todas as outras classes de $Parent1$ e vice-versa para $Child2$.

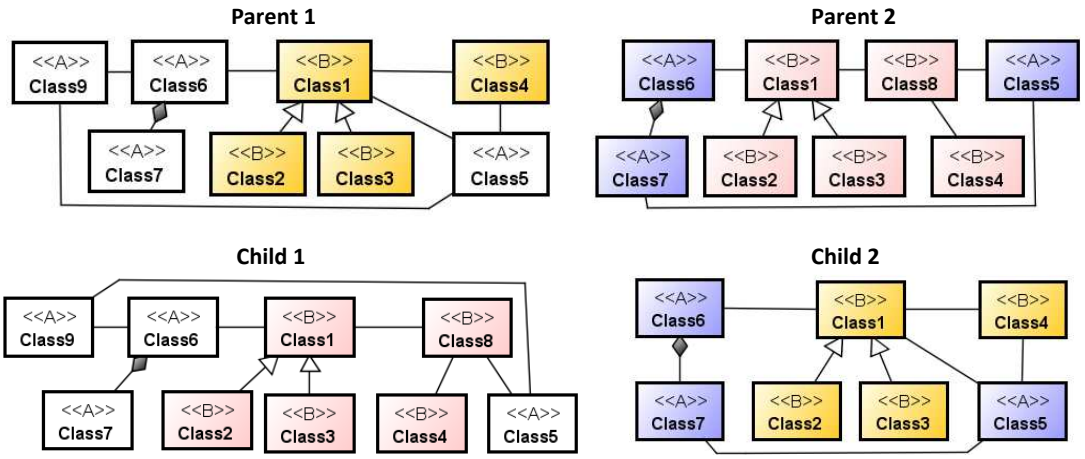


Figura 4.6: Operador Feature-driven Crossover

O pseudocódigo desse operador é apresentado no Algoritmo 4. Uma característica f_x é selecionada aleatoriamente na linha 4. A partir de duas soluções ($Parent1$ and $Parent2$), dois filhos são gerados nas linhas 6 e 10 ($Child1$ e $Child2$). Então, os elementos arquiteturais associados a f_x são removidos de $Child1$ e de $Child2$ (linhas 7 e 11). O próximo passo é adicionar os elementos (e seus relacionamentos) de $Parent2$ associados com f_x em

Algorithm 4: Pseudocódigo do Operador Feature-driven Crossover

```
1  begin
2      Input: Parent1, Parent2
3      F ← Parent1.getAllFeatures()
4      f_x ← a randomly selected feature from F
5      c_1 ← Parent1.getAllElementsAssociatedWithFeature(f_x)
6      c_2 ← Parent2.getAllElementsAssociatedWithFeature(f_x)
7      Child1 ← new Solution(Parent1)
8      Child1.removeElementsRealizingFeature(c_1, f_x)
9      Child1.addElementsRealizingFeature(c_2, f_x)
10     Child1.updateVariabilities()
11     Child2 ← new Solution(Parent2)
12     Child2.removeElementsRealizingFeature(c_2, f_x)
13     Child2.addElementsRealizingFeature(c_1, f_x)
14     Child1.updateVariabilities()
15     Output: Child1, Child2
16 end
```

$Child1$ (linha 8) e os elementos de $Parent1$ em $Child2$ (linha 12). Depois de completar cada filho, os relacionamentos relativos às variabilidades das LPS são atualizados (linhas 9 e 13). Isto precisa ser feito porque existe a possibilidade das classes que representam pontos de variação terem sido trocadas nas soluções geradas ($Child1$ e $Child2$). Ao final do cruzamento, é possível que f_x esteja melhor modularizada em algum filho gerado.

A Figura 4.7 ilustra a aplicação do Feature-driven Crossover na LPS Mobile Media. Apenas os excertos das soluções pais alterados pelo cruzamento são mostrados na figura por questão de espaço e para facilitar o entendimento. Nesse exemplo, a característica $linkMedia$ está modularizada na camada de sistema no componente $LinkMediaCtrl$ de $Parent1$. Em $Parent2$, a característica $SMSTransfer$ está modularizada na camada de sistema dentro do componente $SMSTransferCtrl$.

Supondo que a característica selecionada tenha sido $SMSTransfer$, após a aplicação do Feature-driven Crossover obtém-se as duas soluções filhas, $Child1$ e $Child2$ (Figura 4.7). Em termos de *fitness*, a primeira solução é melhor do que $Parent1$, $Parent2$ e $Child2$ porque o componente $MediaCtrl$ tem mais coesão baseada em características (métrica LCC) e a característica $SMSTransfer$ está menos entrelaçada com as demais características da LPS. Por outro lado, a solução $Child2$ é a pior das quatro, considerando a modularização de $SMSTransfer$ e a coesão de $MediaCtrl$ e, provavelmente, não sobreviverá na próxima geração.

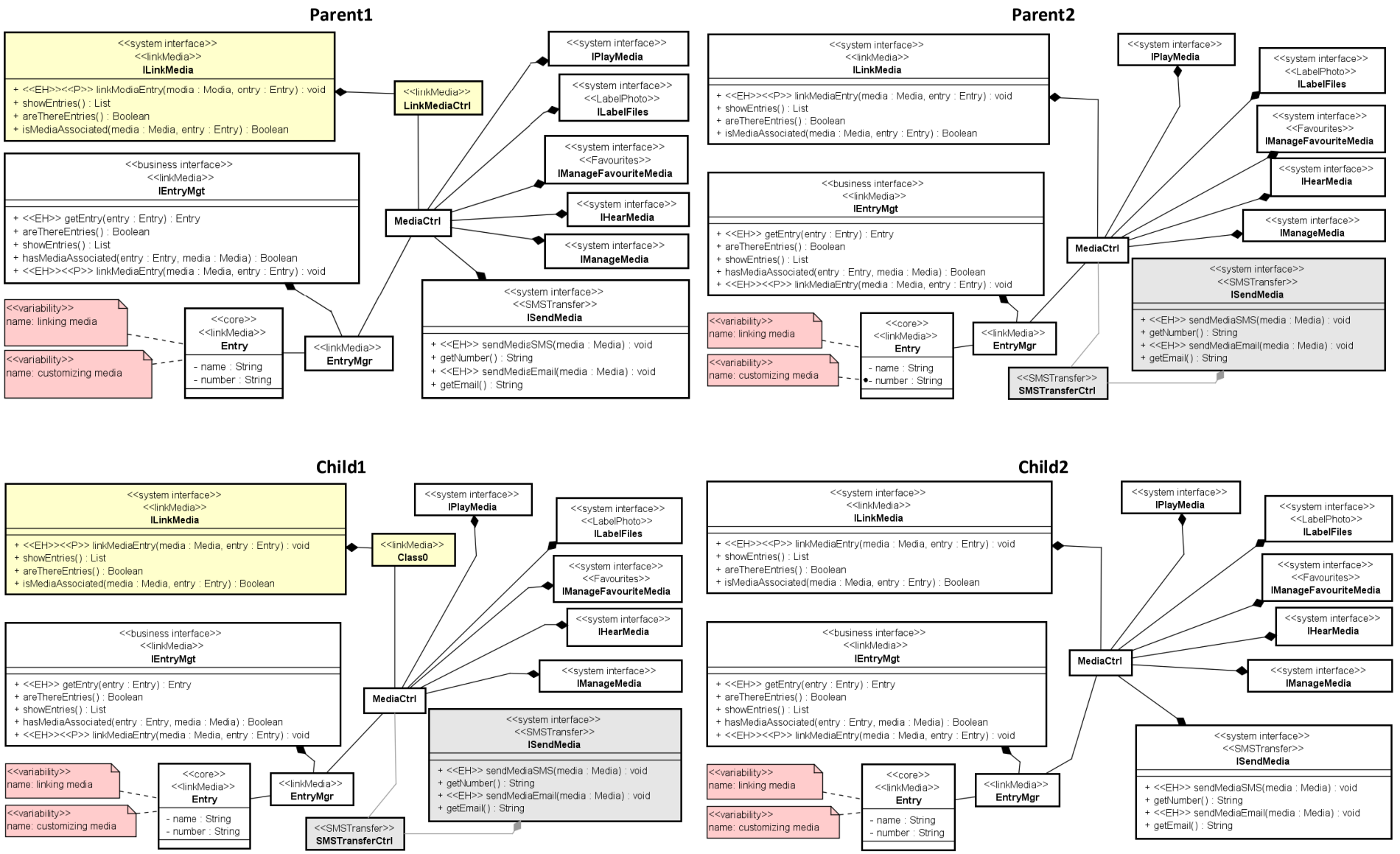


Figura 4.7: Exemplo de aplicação do Feature-driven Crossover

O operador Feature-driven Crossover busca manter e melhorar a modularização das características de um projeto de PLA. Por isso, a aplicação do operador cria novos indivíduos trocando elementos associados com uma característica selecionada aleatoriamente. A hipótese por trás desse operador é que as soluções resultantes ($Child1$ e $Child2$) podem combinar grupos de elementos arquiteturais que melhor modularizem algumas características da LPS que foram herdadas de seus pais. Como o operador de cruzamento pode ser aplicado em cada nova geração, a tendência é que, conforme o número de gerações cresce, cada vez mais filhos com características mais modulares sejam alcançados. Os pais, por sua vez, podem ter sofrido mutações aplicadas pelo Feature-driven Operator em gerações anteriores e, com isso, conter características que estejam razoavelmente bem modularizadas. Assim, o operador de cruzamento dirigido à características Feature-driven Crossover complementa o operador de mutação Feature-driven Operator.

4.4 OPLA-Tool

A ferramenta OPLA-Tool foi criada para possibilitar a aplicação da abordagem MOA4PLA. Os módulos que compõem a OPLA-Tool e suas interdependências são representados na Figura 4.8. Nessa figura os módulos foram coloridos de forma a relacioná-los com as atividades da MOA4PLA (Figura 4.1) que cada um deles apoia.

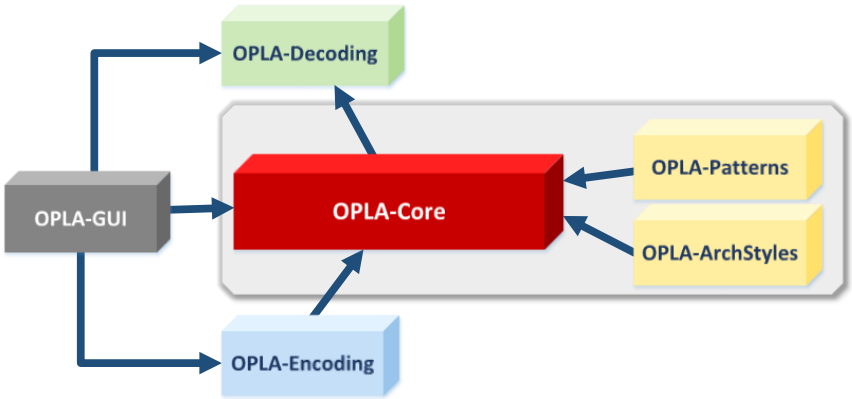


Figura 4.8: Módulos da OPLA-Tool

O módulo OPLA-GUI deve apoiar o arquiteto permitindo que ele escolha qual MOEA deverá ser utilizado no processo de busca, bem como quais funções objetivo e quais ope-

radores evolutivos devem ser aplicados na busca. Além disso, esse módulo deve permitir a edição de uma nova PLA usando os recursos do Papyrus[1] e a utilização dessa PLA como entrada para o processo de busca. Da mesma maneira, uma PLA gerada como saída da MOA4PLA poderá ser utilizada como entrada para uma nova busca, tendo sido editada ou não antes do início dessa nova busca. Visando a facilitar o uso prático da abordagem e ao intercâmbio de modelos, todos os artefatos da MOA4PLA tanto de entrada como de saída são arquivos XMI contendo o projeto da PLA.

Outra funcionalidade do módulo OPLA-GUI é permitir que o arquiteto visualize o conjunto de PLAs geradas como saída, de forma a apoiar a atividade de seleção de uma das alternativas de PLA para empregar na LPS. Tanto os valores das funções objetivo como o diagrama de classes que representa o projeto da PLA podem ser visualizados. O arquiteto tem a opção de selecionar uma alternativa de projeto a partir do seu *fitness* ou por meio da visualização do diagrama. Esse módulo também identifica quais são as soluções que têm o menor valor para cada uma das funções objetivo utilizadas e qual é a solução que tem o melhor *trade-off* entre os objetivos (indicador ED).

Para fazer isso, OPLA-GUI deve utilizar serviços disponibilizados pelos módulos OPLA-Encoding, OPLA-Decoding e OPLA-Core. OPLA-Encoding converte uma PLA modelada usando o Papyrus em uma representação equivalente àquela definida no metamodelo da Figura 4.2. Essa representação é então manipulada pelo OPLA-Core que realiza todo o processo evolutivo utilizando algum MOEA e que retorna o conjunto de soluções não dominadas, obtido no processo de otimização. Esse conjunto é decodificado pelo OPLA-Decoding que converte cada representação de PLA em uma versão legível no Papyrus.

Os módulos OPLA-Patterns e OPLA-Arch-Styles devem conter operadores de busca para aplicar respectivamente, padrões de projeto e estilos arquiteturais durante o processo de otimização. Esses dois módulos acoplam-se ao OPLA-Core para que seus operadores de busca sejam utilizados em conjunto com os operadores já existentes no OPLA-Core ou substituir os operadores previamente definidos, de acordo com a escolha do arquiteto.

[1] $< http://www.eclipse.org/modeling/mdt/papyrus/ >$

Neste trabalho foi implementado o módulo OPLA-Core, cujos aspectos de implementação são descritos na próxima seção. Os demais módulos da OPLA-Tool encontram-se em desenvolvimento em três trabalhos de mestrado. Os módulos OPLA-Encoding e OPLA-Decoding já estão desenvolvidos e estão em fase de integração com o OPLA-Core. A previsão é de que os módulos OPLA-GUI e OPLA-Patterns estejam concluídos até setembro de 2014 e que o módulo OPLA-Arch-Styles esteja disponível a partir de fevereiro de 2015.

Na primeira versão da OPLA-Tool, o arquiteto de LPS interage somente antes e depois do processo de otimização. Na interação inicial o arquiteto fornece o projeto de PLA a ser utilizado e seleciona o MOEA, os operadores evolutivos e as funções de *fitness* a serem utilizados no experimento. Após o processo de otimização, a interação envolve a seleção de uma das soluções geradas (Seção 5.7).

4.4.1 Aspectos de Implementação do OPLA-Core

O módulo OPLA-Core estende a implementação do *framework* jMetal [29], o qual é voltado para otimização multiobjetivo e implementa uma grande variedade de metaheurísticas, incluindo algoritmos evolutivos, tais como o GA canônico (gGA) e os MOEAs NSGAII e PAES. OPLA-Tool adiciona ao jMetal um novo problema de otimização (alvo da MOA4PLA), novos operadores evolutivos propostos por essa abordagem, além de novas métricas, tanto convencionais como específicas para avaliação de PLAs. Nesse sentido ele apoia mais de uma atividade da MOA4PLA. Um detalhamento dos pacotes contidos na OPLA-Tool pode ser visto na Figura 4.9. O pacote *Metrics* contém tanto as métricas convencionais como as métricas específicas para LPS que são utilizadas nas funções objetivo utilizadas no modelo de avaliação da MOA4PLA. O pacote *MultiobjectiveOptimization* contém o problema de otimização denominado $OPLAProblem$, a adaptação de alguns algoritmos para o problema OPLA (sub-pacote MOEAs) bem como os operadores evolutivos (pacote *EvolutionaryOperators*) que incluem tanto operadores de mutação convencionais como operadores evolutivos específicos para PLA propostos nesta tese (Feature-driven Operator e Feature-driven Crossover).

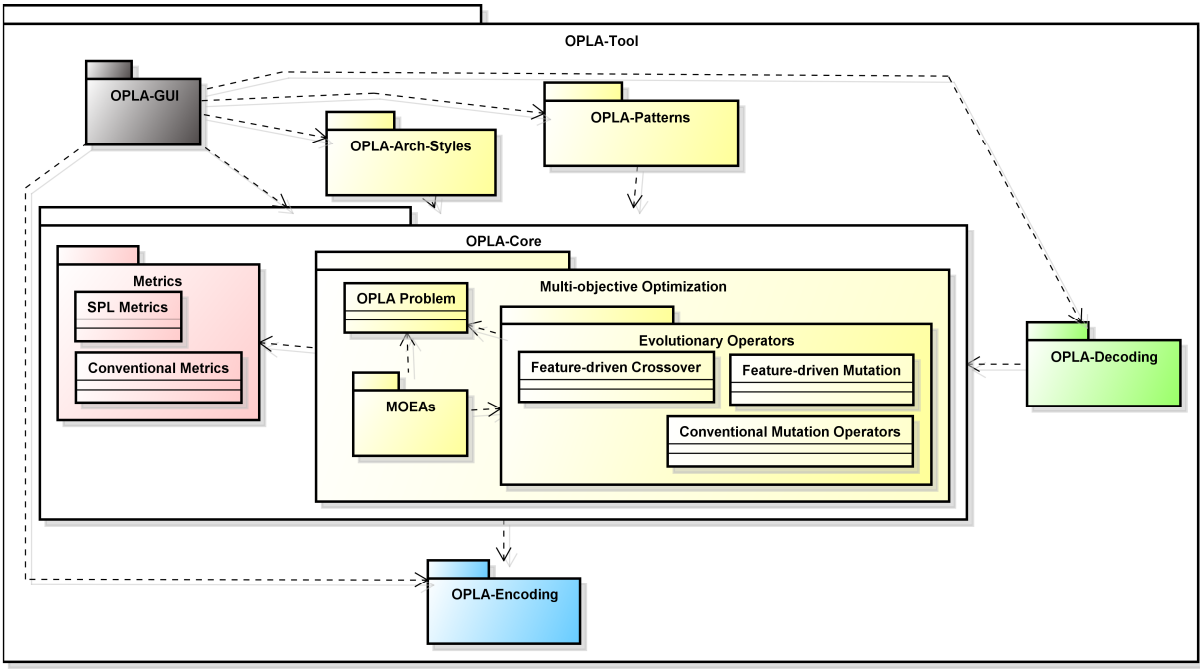


Figura 4.9: Pacotes que compõem a OPLA-Tool

A Figura 4.10 mostra as principais classes criadas na OPLA-Tool (destacadas em cinza). OPLA-Tool adiciona um novo problema, chamado $OPLA$, o qual é uma especialização da classe $Problem$ do jMetal. O algoritmo genético gGA e os MOEAs NSGAII e PAES foram implementados para o problema $OPLA$. $OPLA$ usa $featuredrivenMetrics$ e $conventionalMetrics$ para otimizar variáveis do tipo $Architecture$. A classe $ArchitectureSolutionType$ foi adicionada para facilitar a manipulação de arquiteturas. O arquivo XMI, contendo a PLA, fornecido como entrada para a OPLA-Tool é instanciado como um objeto da classe $Architecture$.

$PLAMutation$ contém os operadores de busca convencionais. $PLAFeatureMutation$ implementa o Feature-driven Operator (descrito na Seção 4.2.3). O operador de cruzamento Feature-driven Crossover foi implementado na classe $PLACrossover$. Os indivíduos de cada geração são selecionados usando o operador de seleção por torneio chamado binary tournament implementado pelo jMetal [29].

A versão atual da OPLA-Tool, não oferece apoio de interface gráfica ao usuário já que o módulo OPLA-GUI ainda não foi desenvolvido. Uma outra limitação dessa versão está relacionada ao formato dos artefatos de entrada e saída da OPLA-Tool. Atualmente, os recursos do SDMetrics OpenCore [100] são utilizados pelo módulo (desenvolvido em [64])

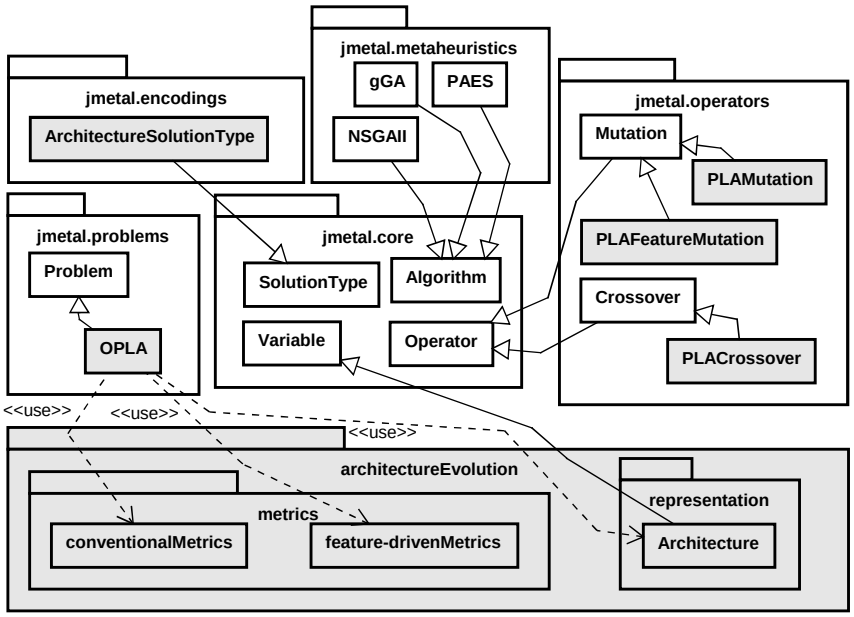


Figura 4.10: Classes da OPLA-Tool e do jMetal

que lê o arquivo XMI contendo o projeto original da PLA, para instanciar a representação do problema a ser usada no processo evolutivo. Para garantir o funcionamento adequado desse módulo, a PLA deve ser modelada usando a ferramenta Poseidon[2] e exportada no formato XMI[3]. As variabilidades da LPS devem ser representadas usando a abordagem SMarty (Seção 2.2). Então, a representação da PLA é gerada a partir do arquivo XMI e seguindo o metamodelo da MOA4PLA. Durante o processamento do arquivo de entrada, para cada elemento do arquivo XMI identificado, instancia-se um objeto referente à alguma metaclasse do metamodelo.

Durante o processo de otimização, os operadores evolutivos são aplicados sobre as alternativas de projeto de PLA geradas e, ao final, obtém-se o conjunto de soluções não-dominadas como saída. Cada solução é uma alternativa de projeto para a PLA, que deve ser convertida em um formato legível para o arquiteto. Para isso, cada solução é salva em um arquivo XMI. O SDMetrics OpenCore não apoia a criação de arquivos XMI, oferecendo apoio somente à atividade de leitura de arquivos XMI. Por esse motivo, o *framework* UML2[4] da Eclipse Foundation tem sido usado para criar os arquivos XMI referentes a cada solução gerada como saída do algoritmo. Entretanto, a Poseidon não consegue ler

[2] $< http://www.gentleware.com/ >$
[3] $< http://www.omg.org/spec/XMI/2.1.1/ >$
[4] $< www.eclipse.org/uml2/ >$

os arquivos XMI gerados pelo UML2 devido a alguns detalhes específicos que a Poseidon inclui nos seus arquivos XMI. Dessa forma, existe um problema de compatibilidade entre os formatos dos arquivos de entrada e saída da OPLA-Tool. Esse problema impede a retroalimentação do processo. Assim, uma alternativa de projeto gerada pela OPLA-Tool não consegue ser utilizada como entrada para um novo processo evolutivo. Essa situação dificulta a operação da ferramenta afetando diretamente a sua usabilidade. Essas limitações técnicas justificaram o desenvolvimento dos módulos OPLA-Encoding e OPLA-Decoding, eliminando o uso do SDMetrics OpenCore e da Poseidon e baseando-se na ferramenta de modelagem Papyrus que é integrada com XML2.

Apesar das limitações técnicas, o módulo que gera as representações do projeto da PLA permitiu usar o OPLA-Core para realizar experimentos a fim de mostrar a efetividade da MOA4PLA. Tais experimentos são relatados no próximo capítulo. Mais detalhes a respeito da arquitetura do módulo usado na versão atual podem ser obtidos em [64]. Informações mais detalhadas sobre a implementação da OPLA-Tool estão relatadas em [21].

4.4.2 Implementação das restrições

Após cada mutação ou cruzamento realizado pelos algoritmos evolutivos, um método é executado para avaliar as restrições impostas para o problema que está sendo otimizado. A restrição imposta pela MOA4PLA de que "componentes, interfaces e/ou classes não podem ser vazios"é verificada pelo método *evaluateConstraints* da classe *OPLA* (pacote *jmetal.problems*). Nesse método cada um dos componentes (pacotes) é analisado para verificar se ele está vazio. Um componente é considerado vazio quando não tem classes ou interfaces. As classes são consideradas vazias se não contêm atributos nem métodos. As interfaces são consideradas vazias se não contém operações.

Assim, essa restrição é verificada sempre que uma nova solução é gerada. Caso a solução encontrada infrinja essa restrição, a solução é reparada eliminando-se os elementos vazios. A opção por reparar a solução ao invés de descartá-la visa a preservar a diversidade da população e a manter projetos de PLA que têm potencial para serem bem avaliados em termos de fitness.

Todas as demais restrições (Seção 4.2.5) foram implementadas nos operadores de busca e são garantidas por eles, uma vez que foram adicionadas verificações para cada uma das restrições. Essas verificações impedem a aplicação do operador de busca quando algum dos elementos selecionados para sofrer o efeito do operador de busca entrar em uma condição que infrinja uma restrição. Por exemplo, o operador MoveMethod não será aplicado caso tenha selecionado um método abstrato de uma superclasse para ser movido para uma outra classe, pois a sua aplicação gerará uma solução que infringe a restrição referente a não movimentar métodos abstratos de classes que participam de hierarquias de herança. Assim, os operadores de busca somente são aplicados quando não gerarão soluções inconsistentes com relação às referidas restrições.

Duas novas restrições foram adicionadas no último estudo empírico (Capítulo 5) para evitar que soluções inválidas fossem geradas nos experimentos usando o operador de cruzamento Feature-driven Crossover. Dessa forma, no último estudo uma solução foi considerada inválida quando:

1. não atende a alguma das restrições definidas na Seção 4.2.5;
2. possui alguma interface não vazia desconectada no projeto, isto é, sem clientes ou *suppliers*; e,
3. algum ponto de variação não está presente no projeto.

As soluções inválidas que não atendam às restrições 2 e 3 são contadas e descartadas porque recuperá-las seria mais custoso. Essas duas restrições são verificadas no método $crossoverFeatures$ da classe $PLACrossover$, logo após a obtenção de cada filho pelo método *obtainChild*. A restrição 2 é verificada pelo método $isValidSolution$ da classe $PLACrossover$. A restrição 3 é verificada pelo método $updateVariabilitiesOffspring$ da classe $PLACrossover$, cujo retorno é armazenado na variável $variabilitiesOk$.

4.5 Considerações Finais

Neste capítulo foi introduzida a MOA4PLA, uma abordagem de otimização multiobjetivo para projeto de PLA. A abordagem se justifica dado que abordagens existentes para pro-

jeto de software baseado em busca não se mostraram apropriadas para avaliar e melhorar o projeto de PLA.

MOA4PLA inclui um processo para realizar a otimização multiobjetivo de um projeto de PLA. Ela inclui também um metamodelo para representar os projetos de PLA, operadores de busca convencionais e um operador de busca para modularização de características. Além disso, estão disponíveis algumas funções objetivo que podem ser combinadas diferentemente pelo arquiteto. Essas funções incluem as métricas sugeridas para atender ao propósito da abordagem proposta.

MOA4PLA pode ser instanciada utilizando diferentes algoritmos multiobjetivos e diferentes métricas. Inclusive, a abordagem permite tratar o problema de otimização de projeto de PLA em diferentes níveis de abstração, seja sob a ótica do problema de CRA ou do problema de aplicação de padrões de projeto à arquitetura. A instanciação apresentada neste capítulo sugere o uso de MOEAs por terem se mostrado eficientes para resolver problemas similares. Para essa instanciação, um operador de cruzamento específico para modularizar características de LPS (Feature-driven Crossover) também foi proposto. Nos algoritmos evolutivos, Feature-driven Crossover e Feature-driven Operator, este último implementado como operador de mutação, podem ser usados a fim de alcançar projetos de PLA com melhor modularização de características do que os projetos originais.

A ferramenta OPLA-Tool foi projetada para permitir a aplicação da MOA4PLA. O módulo OPLA-Core possibilitou a realização de vários estudos empíricos com o intuito de avaliar a efetividade da abordagem proposta. Ainda assim, a conclusão dos demais módulos da OPLA-Tool permitirá, por exemplo, que o projeto de PLA seja otimizado em relação à aplicação de padrões de projeto e estilos arquiteturais, e que o arquiteto possa utilizar, como entrada de um processo de busca, as soluções geradas como saída em processo de busca anteriores.

Apesar da definição teórica da MOA4PLA e da ilustração de algumas de suas características, é importante aplicá-la em estudos empíricos para aumentar a confiabilidade de sua utilização. Nesse sentido, o próximo capítulo é dedicado à avaliação empírica da abordagem proposta.

CAPÍTULO 5

AVALIANDO A ABORDAGEM PROPOSTA

Em geral, a melhor forma para conhecer o desempenho de um algoritmo de busca multiobjetivo na resolução de um determinado problema é experimentalmente. Por esse motivo, a avaliação da MOA4PLA envolve a realização de estudos empíricos, os quais têm o intuito de avaliar as características da abordagem proposta. Tais características envolvem os operadores de busca, o tratamento multiobjetivo dado para o problema, as métricas utilizadas e, os algoritmos de busca empregados. Como há vários fatores envolvidos, decidiu-se por conduzir quatro estudos empíricos, que foram planejados de acordo com a Tabela 5.1. Nesta tabela são apresentados os objetivos de cada estudo, os nomes dos experimentos executados, os algoritmos, operadores de busca e funções objetivo utilizados. Na última coluna são apresentados quais dados foram coletados para serem utilizados na análise quantitativa dos resultados obtidos.

O alvo do Estudo 1 foi investigar as três principais características da MOA4PLA. O primeiro objetivo foi avaliar o tratamento multiobjetivo proposto pela MOA4PLA e, para isso, são utilizados dois algoritmos GA e NSGAII. Como o primeiro é mono-objetivo diferentes pesos foram atribuídos às funções objetivo de forma a agregá-las numa mesma função de *fitness*. Os pesos são apresentados na coluna Observações da Tabela 5.1. O segundo objetivo desse estudo foi analisar a efetividade do Feature-driven Operator. Para atender a esse objetivo, foram realizados experimentos utilizando somente os operadores de busca convencionais e outros adicionando o Feature-driven Operator. O terceiro e último objetivo era analisar a adequação das métricas utilizadas para medir os projetos de PLA alcançados. Nos experimentos executados foram utilizadas as funções objetivo FM e CM, referentes às métricas dirigidas à características e às métricas convencionais.

O intuito do Estudo 2 foi avaliar a viabilidade de uso do operador de cruzamento Feature-driven Crossover. Nesse estudo, dois experimentos foram executados utilizando

Tabela 5.1: Estudos de Avaliação da MOA4PLA

Estudo	Objetivo	Nome do Experimento	Algoritmo	Operadores de Busca Utilizados	Funções Objetivo	Observações	Dados coletados
1	Investigar as principais características da MOA4PLA: o tratamento multiobjetivo, a efetividade do Feature-driven Operator e a adequação das métricas utilizadas	GA	GA	Convencionais	FM e CM	CM:0.5, FM:0.5	Tempo de execução *Fitness* das soluções % de Melhoria em FM Solução de menor ED
		GAC	GA	Convencionais		CM:1.0, FM:0	
		GAF	GA	Convencionais		CM:0, FM:1.0	
		GA-FM	GA	Convencionais+Feature-driven Operator		CM:0.5, FM:0.5	
		GAC-FM	GA	Convencionais+Feature-driven Operator		CM:1.0, FM:0	
		GAF-FM	GA	Convencionais+Feature-driven Operator		CM:0, FM:1.0	
		NSGAII	NSGAII	Convencionais			
		NSGAII-FM	NSGAII	Convencionais+Feature-driven Operator			
2	Avaliar a viabilidade de uso do operador de cruzamento proposto (Feature-driven Crossover)	NSGAII-M	NSGAII	Convencionais + Feature-driven Operator	FM e CM		Tempo de execução *Fitness* das soluções Hipervolume Solução de menor ED
		NSGAII-MC	NSGAII	Convencionais+Feature-driven Operator+ Feature-driven Crossover			
3	Comparar o desempenho do algoritmo PAES em relação ao NSGAII	NSGAII-M	NSGAII	Convencionais + Feature-driven Operator	FM e CM	Prob. de Mutação: 90%	Tempo de execução *Fitness* das soluções Solução de menor ED
		NSGAII-MC	NSGAII	Convencionais+Feature-driven Operator+ Feature-driven Crossover		Prob. de Mutação: 90%	
		PAES-90	PAES	Convencionais + Feature-driven Operator		Prob. de Mutação: 90%	
		PAES-100	PAES	Convencionais + Feature-driven Operator		Prob. de Mutação: 100%	
4	Caracterizar as soluções obtidas a partir de diferentes combinações de funções de *fitness* disponíveis na MOA4PLA	NSGAII-M-FE	NSGAII	Convencionais+Feature-driven Operator	FM e Ext		Tempo de execução *Fitness* das soluções Elegância das Soluções
		NSGAII-M-FC			FM e CM		
		NSGAII-M-FCE			FM, CM e Ext		
		NSGAII-MC-FE		Convencionais+Feature-driven Operator+ Feature-driven Crossover	FM e Ext		
		NSGAII-MC-FC			FM e CM		
		NSGAII-MC-FCE			FM, CM e Ext		

o NSGAII e as funções objetivo FM e CM. O fator que variou foi a combinação de operadores de busca aplicados nos experimentos. No primeiro experimento somente operadores de mutação foram utilizados. No segundo experimento, o operador de cruzamento proposto foi aplicado conjuntamente ao operadores de mutação.

O objetivo do Estudo 3 foi comparar o desempenho do NSGAII com o desempenho do PAES, o qual foi concebido para aplicar somente operadores de mutação. Quatro experimentos foram executados variando dois fatores: a probabilidade de mutação e os operadores de busca aplicados.

O último estudo empírico teve o intuito de caracterizar as soluções geradas pela MOA4PLA diante de três diferentes combinações de funções objetivo para compor o modelo de avaliação. Um estudo deste tipo se justifica dado que no contexto do projeto de PLA baseado em busca ainda não há consenso a respeito das métricas que devem ser empregadas visando a atender os objetivos traçados pela abordagem proposta. O fator variável foi, então, a configuração do modelo de avaliação que foi composto de três diferentes formas como apresentado na Tabela 5.1.

Os quatros estudos empíricos foram realizados utilizando nove projetos de PLAs, apresentados na próxima seção. Seguindo os procedimentos sugeridos em [5], cada experimento envolvido nos estudos foi executado 30 vezes visando a analisar o seu comportamento para resolver o problema. O número de avaliações de *fitness* foi utilizado como critério de parada em todos os experimentos. Os indicadores de qualidade HV e ED foram utilizados para comparar a qualidade das fronteiras de Pareto encontradas pelos experimentos envolvidos nos estudos empíricos realizados. Nenhum indicador específico de diversidade foi utilizado porque o HV já considera esse quesito. Nas Seções 5.2 a 5.5 são apresentados a configuração e os resultados dos estudos empíricos conduzidos. Ameaças à validade dos estudos são abordadas na Seção 5.6 e, na Seção 5.7, discutem-se alguns aspectos relacionados à seleção de um projeto de PLA a ser adotado pelo arquiteto.

5.1 PLAs utilizadas

A Tabela 5.2 contém informações sobre as PLAs utilizadas nos estudos empíricos realizados, tais como, números de componentes, de classes, de características e de variabilidades. Esta tabela também apresenta o valor original de *fitness* de cada PLA, considerando as quatro funções objetivo disponíveis na MOA4PLA.

Tabela 5.2: Informações sobre as PLAs

PLA	*Fitness* Original (FM, CM, Ext, Eleg)	# Componentes	# Interfaces	# Classes	# Variabilidades	# Características
AGM-1	(391.0, 31.82, 0.524, 0.278)	9	14	20	4	9
AGM-2	(500.0, 35.18, 0.617, 0.335)	9	14	21	5	11
AGM-3	(360.0, 31.82, 0.524, 0.278)	9	14	20	4	9
AGM-4	(462.0, 35.18, 0.617, 0.335)	9	14	21	5	11
MM-1	(608.0, 39.31, 0.777, 0.182)	11	16	13	7	13
MM-2	(534.0, 33.48, 0.777, 0.195)	8	13	10	7	13
MM-3	(434.0, 39.31, 0.777, 0.182)	11	16	13	7	12
MM-4	(430.0, 33.48, 0.777, 0.195)	8	13	10	7	12
LPS-BET	(678.0, 178.70, 1000.0, 0.567)	56	33	115	10	16

A Arcade Game Maker (AGM) [46] é uma LPS criada pelo SEI que inclui três jogos de arcade: Brickles, Bowling e Pong. AGM-1 é apresentada em [23] e AGM-2 foi criada em [101] a partir de AGM-1, por meio da adição de duas novas características: *ranking* e *logging*. Essas características têm como objetivo manter informações tanto sobre as melhores pontuações obtidas em cada jogo disponível no produto como quais usuários alcançaram as respectivas pontuações. AGM-3 e AGM-4 contêm os mesmos elementos arquiteturais de AGM-1 e AGM-2, respectivamente, porém, o mapeamento de características para os elementos arquiteturais é diferente das PLAs originais. Esse fato pode levar a diferentes resultados durante o processo de otimização, tendo em vista a aplicação dos operadores de mutação e de cruzamento dirigidos à modularização de características.

Mobile Media (MM) [23] é uma LPS que apoia o gerenciamento de diferentes tipos de mídias para dispositivos móveis, incluindo música, vídeos e fotos. MM-1 foi desenvolvida em [23] e MM-2 [101] é equivalente à *release* R8 apresentada em [33]. MM-1 e MM-2 são equivalentes em termos de características apesar de MM-2 ter menor número de elementos arquiteturais. A principal diferença está no componente MediaMgr que é menos

acoplado e mais coeso na versão MM-2. Além disso, três características estão menos espalhadas, aumentando a coesão baseada em características. MM-3 e MM-4 foram obtidas a partir MM-1 e MM-2, respectivamente, apesar de terem diferentes mapeamentos de características.

Nas PLAs AGM-3, AGM-4, MM-3 e MM-4 o mapeamento de características aos elementos arquiteturais foi realizado por um projetista diferente dos que realizaram o mapeamento de características nas PLAs AGM-1, AGM-2, MM-1 e MM-2. Inclusive, na MM-3 e na MM-4 o número de características é menor (uma a menos). Dessa forma, o mapeamento obtido é diferente, pois dependeu da interpretação do projetista.

LPS-BET [28] apoia o gerenciamento de transporte urbano. Dentre as características disponíveis estão o pagamento de transporte por meio de cartão eletrônico, abertura automática do portão de embarque e pagamento de viagens de integração. Foi concebida baseando-se em três produtos de software de controle de transporte de cidades brasileiras. Dentre as LPS utilizadas no presente trabalho, a LPS-BET é a de maior porte, envolvendo maiores números de elementos arquiteturais, características e variabilidades.

Todas as PLAs são baseadas em componentes e seguem o estilo arquitetural em camadas. Há componentes gerenciadores que fornecem interfaces para viabilizar a comunicação entre as camadas de GUI, de sistema e de negócio. Aos nomes dos componentes foram adicionados sufixos para facilitar a identificação da camada arquitetural a qual pertencem. Os sufixos utilizados são “GUI”, “Ctrl” e “Mgr” para os componentes da camada de GUI, de sistema e de negócio, respectivamente.

5.2 Estudo 1: Avaliando a MOA4PLA

O primeiro estudo empírico foi realizado com o intuito de avaliar as principais características da MOA4PLA, e teve como objetivos responder a três questões de pesquisa (RQs), descritas a seguir:

RQ1: O tratamento multiobjetivo dado pela MOA4PLA para o problema de projeto de PLA é adequado? Esta questão tem como objetivo avaliar o trata-

mento multiobjetivo dado pela abordagem para o referido problema, quando comparado a um tratamento mono-objetivo em termos das soluções encontradas e dos valores das métricas. Para apoiar a análise quantitativa que embasa a resposta desta questão, para cada solução do PF_{true} foram calculados os percentuais de melhoria no valor da função objetivo $FM(pla)$. Dessa forma, foi possível comparar as soluções com pior percentual, melhor percentual e melhor *trade-off*, lembrando que esta última solução é a que tem o menor valor do indicador ED.

RQ2: O operador Feature-driven Operator, que foi implementado como um operador de mutação, modulariza efetivamente as características das PLAs? Para responder essa questão, foram realizadas análises das soluções obtidas por dois algoritmos: um que aplica somente operadores de busca convencionais e outro que aplica o Feature-driven Operator além dos operadores convencionais. Uma análise quantitativa baseada nos valores de ED e uma análise qualitativa apoiam a resposta desta questão.

RQ3: As métricas usadas no modelo de avaliação são apropriadas para avaliar os projetos de PLAs no contexto da MOA4PLA? É importante analisar se as métricas aplicadas são sensíveis às mudanças aplicadas pelos operadores de busca no contexto das PLAs usadas no estudo. A resposta desta questão é baseada em uma análise qualitativa da sensibilidade das métricas em avaliar o *fitness* das soluções com melhor e pior percentuais de melhoria em $FM(pla)$.

Nas próximas seções são apresentadas as configurações do estudo, os principais resultados alcançados e as respostas para as questões de pesquisa. Informações mais detalhadas sobre o estudo podem ser obtidas no Apêndice C.

5.2.1 Configuração do Estudo 1

O estudo foi realizado utilizando cinco PLAs: AGM-1, AGM-2, LPS-BET, MM-1 e MM-2. O modelo de avaliação envolveu duas funções objetivo: $FM(pla)$ e $CM(pla)$ (Equações 4.1 e 4.2). Lembrando que $FM(pla)$ se refere às métricas dirigidas a características e $CM(pla)$ às métricas convencionais.

Três alternativas de GAs foram implementadas para responder a **RQ1**. Como GAs são mono-objetivos e dois objetivos estão sendo usados no modelo de avaliação, funções de agregação ponderadas para FM e CM foram utilizadas para avaliar o *fitness*. Três configurações diferentes de pesos foram adotadas. Para verificar a influência empírica de cada métrica no projeto de PLA as configurações foram montadas da seguinte maneira: (i) minimizar somente CM (experimento denominado GAC), (ii) minimizar somente FM (experimento denominado GAF) e, (iii) dar igual importância a FM e CM (experimento denominado GA). Na configuração (i) o peso de FM foi ajustado como zero. Na configuração (ii) o peso de CM foi ajustado como zero. Na última configuração o peso de cada função é 0,5.

Com o objetivo de responder **RQ2**, tanto os GAs como o NSGAII foram implementados em duas versões: uma usando apenas os operadores de mutação convencionais e outra usando os operadores convencionais e o Feature-driven Operator. Assim, esse estudo empírico envolveu seis experimentos. Os experimentos denominados GA, GAC, GAF e NSGAII empregam somente os operadores convencionais. Nos experimentos denominados GA-FM, GAC-FM, GAF-FM e NSGAII-FM, o Feature-driven Operator foi adicionado.

Nesse estudo, todos os operadores de mutação tinham duas limitações de implementação: (i) não se preocupavam em identificar as camadas arquiteturais às quais os elementos alvo das mutações pertenciam e, (ii) não eram aplicados à classes que participavam em relacionamentos de generalização.

5.2.2 Análise dos Resultados do Estudo 1

As Tabelas 5.3 e 5.4 apresentam o *fitness* das soluções encontradas pelos GAs e NSGAIIs, respectivamente, seguindo o formato (FM, CM). Na Tabela 5.3 o tempo de execução médio (em segundos) para obter cada PF_{approx} é mostrado abaixo do *fitness* da solução.

Como os experimentos realizados com o NSGAII retornam um conjunto de soluções não dominadas, na Tabela 5.4 é apresentado o número de soluções que forma o conjunto PF_{true} de cada PLA. A segunda e a quinta colunas apresentam o número de soluções que formam cada conjunto PF_{known} e, entre parênteses, o número de soluções de PF_{known}

Tabela 5.3: Estudo 1 - *Fitness* das soluções encontradas pelos GAs

PLA	GA	GA-FM	GAC	GAC-FM	GAF	GAF-FM
AGM-1	(186.0,23.25) R: 115.24	(184.0,21.72) R: 108.21	(491.0,17.34) R: 103.88	(438.0,17.34) R: 113.75	(184.0,25.0) R: 129.14	(182.0,28.16) R: 103.44
AGM-2	**(237.0,28.62)** R: 123.17	(225.0,29.48) R: 120.10	(467.0,20.75) R: 113.01	(476.0,20.57) R: 117.08	**(236.0,42.51)** R: 132.62	**(231.0,32.64)** R: 118.83
MM-1	(306.0,31.11) R: 125.29	(225.0,27.16) R: 113.75	(588.0,29.17) R: 112.90	(540.0,27.8) R: 124.60	(307.0,31.11) R: 137.18	(221.0,32.13) R: 116.81
MM-2	(229.0,26.83) R: 119.12	**(178.0,20.76)** R: 106.19	(492.0,23.52) R: 104.05	(443.0,21.77) R: 109.84	(226.0,33.83) R: 129.30	(181.0,21.26) R: 107.66
LPS-BET	(629.0,160.6) R: 578.07	**(475.0,174.4)** R: 557.84	**(671.0,147.1)** R: 589.76	**(660.0,148.1)** R: 573.56	(625.0,171.4) R: 595.38	(477.0,208.0) R: 549.07

Tabela 5.4: Estudo 1 - *Fitness* das soluções encontradas pelos NSGAIIs

PLA	# PF_{true}	NSGAII		NSGAII-FM	
		# PF_{know}	*Fitness*	# PF_{know}	*Fitness*
AGM-1	8	7 (6) R: 103.2	**(298.0, 17.34)** (201.0, 21.47) **(261.0, 17.66) (247.0, 18.34)** **(212.0, 18.66) (209.0, 20.66)** **(267.0, 17.47)**	4 (2) R: 106.8	**(182.0, 20.75) (181.0, 21.25)** (332.0, 19.25) (304.0, 19.5)
AGM-2	12	7 (4) R: 112.2	**(329.0, 18.62) (314.0, 19.62)** **(326.0, 19.57)** (276.0, 22.76) (301.0, 20.57) (277.0, 21.76) **(303.0, 19.76)**	7 (5) R: 118.9	**(244.0, 24.09) (253.0, 23.50)** (459.0, 19.72) **(262.0, 20.57)** **(261.0, 22.57)** (239.0, 29.50) **(243.0, 28.50)**
MM-1	4	3 (3) R: 119.1	**(347.0, 24.9) (332.0, 25.11)** **(326.0, 26.11)**	2 (1) R: 125.7	**(219.0, 26.20)** (437.0, 26.08)
MM-2	6	5 (0) R: 110.3	(236.0, 25.83) (249.0, 20.67) (238.0, 21.83) (246.0, 21.67) (339.0, 20.16)	6 (5) R: 112.5	**(276.0, 19.13) (204.0, 20.43)** **(258.0, 19.25) (243.0, 19.92)** **(236.0, 20.22)** (178.0, 21.29)
LPS-BET	31	8 (0) R: 580.4	(647.0, 154.8) (635.0, 159.6) (639.0, 157.6) (640.0, 156.8) (636.0, 158.6) (642.0, 156.6) (643.0, 155.8) (637.0, 157.8)	30 (28) R: 563.7	**(609.0, 154.1)(574.0, 156.1)(615.0, 153.1)(612.0, 153.4)** **(595.0, 155.1)(546.0, 157.4)(596.0, 154.4)(487.0, 168.2)** **(490.0, 166.4)(479.0, 173.8)**(476.0, 177.4) **(543.0, 158.8)** **(541.0, 159.6)(493.0, 164.6)(480.0, 173.8)(538.0, 159.8)** **(537.0, 160.6)(520.0, 160.8)(519.0, 161.6)(512.0, 161.8)** **(511.0, 162.6)586.0, 155.1)(553.0, 156.4)(483.0, 171.8)** **(542.0, 159.1) (551.0, 157.1)**(477.0, 174.4) **(498.0, 162.8)(482.0, 172.6)(481.0, 173.4)**

que está incluído em PF_{true}. Nessas mesmas colunas também é apresentado o tempo de execução médio (em segundos) utilizado para obter cada PF_{approx}. O tempo de execução de todos os experimentos é semelhante.

Nas duas tabelas, as soluções destacadas em negrito constituem o conjunto PF_{true}, aquele que contém as melhores soluções encontradas entre todos os experimentos para uma determinada PLA. Os experimentos que usam algoritmos multiobjetivos (NSGAII e NSGAII-FM) são os que encontraram o maior número de soluções pertencentes a PF_{true}.

A Tabela 5.5 mostra os percentuais de melhoria nas funções FM e CM referentes às soluções encontradas por NSGAII e NSGAII-FM. Os valores com os maiores percentuais estão destacados em negrito.

Tabela 5.5: Estudo 1 - Porcentagens de Melhoria no *Fitness*

PLA	NSGAII			NSGAII-FM		
	Melhor FM	**Melhor *trade-off***	**Pior FM**	**Melhor FM**	**Melhor *trade-off***	**Pior FM**
AGM-1	(201.0, 21.47)	(201.0, 21.47)	(298.0, 17.34)	(181.0, 21.25)	(182.0, 20.75)	(332.0, 19.25)
	48.49%, 32.52%	48.59%, 32.52%	23.78%, **45.50**%	**53.70**%, 33.21%	53.45%, 34.78%	15.08%, 39.50%
AGM-2	(276.0, 22.76)	(277.0, 21.76)	(329.0, 18.62)	(239.0, 29.50)	(244.0, 24.09)	(459.0, 19.72)
	44.68%, 35.30%	44.48%, 38.14%	36.04%, **47,07**%	**52,10**%, 16.14%	51.10%, 31.52%	8.01%, 43.94%
MM-1	(326.0, 26.11)	(326.0, 26.11)	(347.0, 24.9)	(219.0, 26.20)	(219.0, 26.20)	(437.0, 26.08)
	46.38%, 33.57%	46.38%, 33.57%	42.92%, 33.57%	**63,98**%, 33.35%	63.98%, 33.35%	28.12%, **33.65**%
MM-2	(236.0, 25.83)	(236.0, 25.83)	(339.0, 20.16)	(178.0, 21.29)	(178.0, 21.29)	(276.0, 19.13)
	55.80%, 22.84%	55.80%, 28.82%	36,51%, 39.78%	**66.66**%, 36.40%	66.66%, 36.40%	48.31%, **42.86**%
LPS-BET	(635.0, 159.65)	(635.0, 159.65)	(647.0, 154.88)	(476.0, 177.43)	(490.0, 166.43)	(615.0, 153.14)
	5.78%, 9.72%	5.78%, 9.72%	4.0%, 12.41%	**29.37**%, -0.33%	27.20%, 5.88%	8.75%, **13.40**%

A Tabela 5.6 apresenta o *fitness* das soluções que têm o menor valor de ED encontradas pelos experimentos executados. Além disso, a tabela contém o *fitness* da solução ideal, que consiste dos menores valores encontrados por todos os experimentos para cada um dos objetivos utilizados. As menores EDs estão em negrito.

Tabela 5.6: Estudo 1 - Menores ED encontradas pelos experimentos

PLA	Solução Ideal	NSGAII		NSGAII-FM		GA-FM	
		ED	***Fitness***	**ED**	***Fitness***	**ED**	***Fitness***
AGM-1	(181.0,17.34)	20.4	(201.0,21.47)	**3.5**	**(182.0,20.75)**	5.3	(184.0,21.72)
AGM-2	(239.0,18.62)	38.1	(277.0,21.76)	**7.4**	**(244.0,24.09)**	10.8	(225.0,29.48)
MM-1	(219.0,24.9)	107.0	(326.0,26.11)	**1.3**	**(219.0,26.20)**	6.4	(225.0,27.16)
MM-2	(178.0,19.13)	58.3	(236.0,25.83)	2.1	(178.0,21.29)	**1.6**	**(178.0,20.76)**
LPS-BET	(475.0,147.14)	160.4	(635.0,159.65)	**24.2**	**(490.0,166.43)**	27.2	(475.0,174.40)

5.2.2.1 Análise Qualitativa

Além da análise quantitativa envolvendo os dados apresentados até aqui, foi realizada uma análise qualitativa das soluções com melhor e pior porcentagem de melhoria em FM e menor ED (melhor *trade-off*) em comparação às PLAs originais. Nos projetos originais havia componentes de sistema e de negócio associados com diversas características e, as características estavam espalhadas em mais de um componente de sistema. De forma geral, nas soluções obtidas nos experimentos, novos componentes foram criados para modularizar características, especialmente na camada de sistema. Nas soluções geradas, as características estão menos espalhadas e menos entrelaçadas.

Um exemplo para ilustrar a modularização de características é apresentado na Figura 5.1. Essa figura mostra alguns elementos do projeto de PLA encontrado para AGM-2

com o menor valor para FM. Os elementos arquiteturais que realizam as características *ranking* e *logging* são apresentados nesta imagem. O projeto completo da PLA é mostrado em alto nível na Figura 5.2. Na PLA original, *ranking* estava entrelaçada com três outras características e espalhada por cinco classes e interfaces de diferentes componentes. No projeto alcançado (Figura 5.1), *ranking* está entrelaçada somente com *logging*. *Ranking* está modularizada em $Component303554$ e é realizada por $ILogin$ e $Class71341$. *Logging* estava originalmente associada juntamente com *ranking* em duas operações, por isso, ainda está entrelaçada com tal característica. Apesar disso, *logging* está modularizada na classe $Player$ e há duas operações associadas com *logging* em $Interface243570$. Essa interface e $Component302639$ não estão associadas com outras características, por isso devem ter sido criados pelo operador de mutação AddComponent que não se preocupa com a associação de características aos elementos arquiteturais. No entanto, essa característica não foi completamente modularizada em apenas um componente, já que está associada com uma classe de $GameBoardMgr$ ($Player$) e com $Interface243570$.

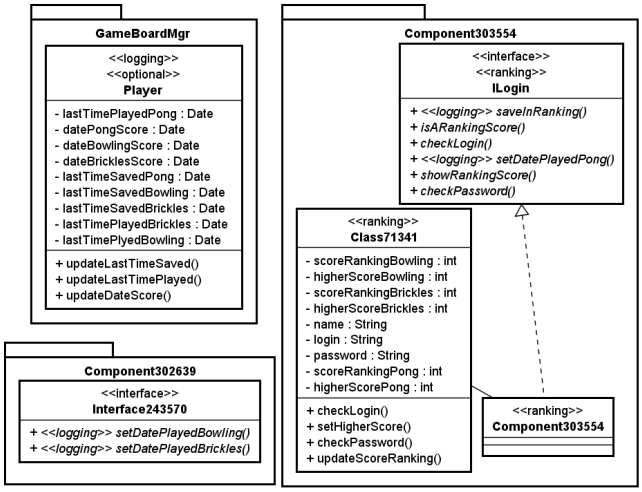


Figura 5.1: Estudo 1 - Características *ranking* e *logging* em uma solução para AGM-2

Um ponto positivo observado na análise qualitativa é a minimização do número de elementos arquiteturais candidatos a apresentarem anomalias arquiteturais já que algumas classes inicialmente tinham um grande número de responsabilidades e se tornaram menos sobrecarregadas nas soluções obtidas. Isso foi observado em várias soluções alcançadas para AGM-1 e AGM-2. Por exemplo, na Figura 5.1, a classe *Player* está associada a

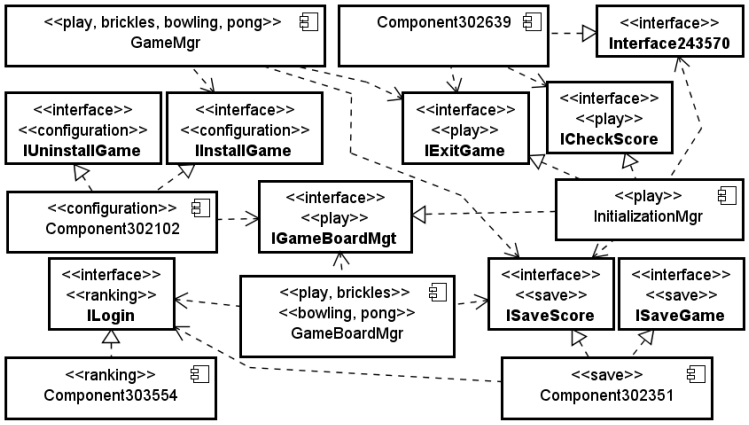


Figura 5.2: Estudo 1 - PLA com o melhor valor de FM para AGM-v2

uma única característica e seus atributos e operações estão altamente relacionados, o que torna esse projeto mais coeso que o original. Considerando o tamanho da classe do projeto original, *Player* era a segunda maior classe. Sabe-se que as classes com muitas responsabilidades e um maior número de atributos e métodos são candidatas a manifestar *bad smells* como *god class* e *large class*. Uma vez que MOA4PLA permite a obtenção de melhores projetos para PLAs como a AGM, para LPS maiores, o benefício de prevenir anomalias arquiteturais relacionadas à falta de modularização tende a ser mais significativo.

A escalabilidade da MOA4PLA é um ponto a ser considerado. Por se tratar de um problema ainda inexplorado no campo de SBSE, é difícil mensurar qual tamanho de LPS justificaria a otimização do projeto por meio de um algoritmo de busca. No entanto, pode-se afirmar que o número de objetivos a serem otimizados, o número de elementos arquiteturais do projeto inicial da PLA e o número de características da LPS influenciam diretamente no tamanho do espaço de soluções. Quando se considera a modularização de características, por exemplo, é possível entender que quanto maiores os números de características da LPS e de elementos arquiteturais, maior o número de soluções que pode ser obtido, já que diferentes possibilidades de modularização das características nos elementos arquiteturais podem existir. Ainda assim, um menor entrelaçamento entre as características tende a facilitar o processo de otimização, influenciando o tamanho do espaço de soluções. A título de ilustração, considerando um processo de busca com 2 objetivos e tomando como base a PLA AGM-2, a qual tem 9 componentes arquiteturais e 11 características, seria possível alcançar um espaço de busca em torno de 198 soluções.

Caso fossem considerados 4 objetivos, teoricamente poder-se-ia alcançar em torno de 396 soluções. No entanto, além das variáveis mencionadas acima, as quais influenciam nesses cálculos, ainda existe a possibilidade de criação de novos componentes arquiteturais por parte dos operadores de busca.

Com relação ao tempo que um MOEA leva para encontrar uma solução, os experimentos envolvendo a LPS-BET - a maior LPS do estudo empírico - demoraram cerca de dez minutos para executar cada uma das trinta rodadas. O tempo de execução das demais PLAs foi ainda menor, menos de dois minutos para cada rodada. Logo, ainda que a LPS a ser otimizada seja muito grande, deduz-se que o tempo de execução para resolver o problema é viável e menor do que o tempo necessário para o arquiteto avaliar e melhorar o projeto original. É possível que ao adotar um maior tamanho de população, a convergência do algoritmo seja maior e a solução seja retornada mais rapidamente. Porém, isso precisaria ser verificado experimentalmente em estudos futuros. Quanto maior a experimentação e a consolidação da MOA4PLA, menor será o esforço demandado para que um arquiteto utilize a abordagem e usufrua dos seus benefícios.

Alguns pontos negativos também foram observados ao analisar os resultados do estudo empírico, a saber:

- Os efeitos da aplicação de operadores convencionais MoveMethod, MoveAttribute e AddClass não foram percebidos nas soluções geradas pelos experimentos. Provavelmente as soluções geradas após a aplicação desses operadores não sobreviveram ao longo de gerações porque as mudanças impactaram negativamente a função FM.

- Outras características poderiam ser modularizadas para um componente em particular em algumas soluções se o Feature-driven Operator tivesse sido aplicado mais vezes.

- A porcentagem de melhoria em FM para LPS-BET foi menor do que nas demais PLAs porque as características estão melhor modularizadas na PLA original e há muitas generalizações que restringem a aplicação dos operadores de busca.

- Os operadores de busca quebraram regras definidas pelo estilo arquitetural em ca-

madas adotado nas PLAs e misturaram elementos arquiteturais que originalmente estavam em camadas diferentes.

Essas observações, somadas aos resultados, permitiram a identificação de algumas oportunidades de pesquisa para ser realizadas e incluem: (a) a aplicação de outras métricas no modelo de avaliação, (b) a aplicação de operadores de mutação em qualquer tipo de relacionamento, (c) investigar se uma maior taxa de aplicação do Feature-driven Operator fornece projetos de PLA com maior modularização de características e, (d) investigar maneiras de considerar as regras de estilos arquiteturais nos operadores de busca.

5.2.2.2 Respondendo as questões de pesquisa

A resposta para **RQ1** é que o tratamento multiobjetivo dado pela MOA4PLA é adequado. Essa resposta é embasada em três motivos:

- O conflito entre os valores alcançados para FM e CM percebido ao analisar o *fitness* das soluções encontradas nos experimentos na Tabela 5.5. Por exemplo, as soluções de AGM-2, MM-1 e LPS-BET que têm o melhor percentual de melhoria em FM têm o pior percentual em CM.

- Algoritmos multiobjetivos alcançam melhores soluções que algoritmos mono-objetivos para o contexto desta tese. Todas as soluções encontradas pelos GAs para AGM-1 e MM-1 são dominadas pelas soluções encontradas pelos experimentos utilizando algoritmos multiobjetivos. Algumas soluções não dominadas encontradas pelos GAs para LPS-BET estão mais longe da solução ideal do que as soluções encontradas pelos experimentos com algoritmos multiobjetivos.

- Os algoritmos multiobjetivos obtêm maior diversidade de soluções do que os algoritmos mono-objetivos. Os algoritmos mono-objetivos se assemelham mais ao processo manual de projeto no qual geralmente os arquitetos de LPS produzem uma única solução de projeto de PLA. Por outro lado, a maior diversidade proporcionada pelos algoritmos multiobjetivos melhora o espaço de soluções de projeto a ser analisado pelos arquitetos.

Com relação à **RQ2** pode-se afirmar que o Feature-driven Operator propicia alcançar projetos de PLA com melhor modularização de características do que os operadores de busca convencionais. Esta afirmação baseia-se nos seguintes fatos:

- Quando comparados com os experimentos que usaram somente os operadores convencionais, os experimentos que empregam o Feature-driven Operator encontram melhores resultados em termos de ED para todas as PLAs. NSGAII-FM encontrou a solução de menor ED para AGM-1, AGM-2, MM-1 e LPS-BET. GA-FM foi o melhor em termos de ED para MM-2.

- As soluções com o maior percentual de melhoria em FM (coluna Melhor FM da Tabela 5.5) foram encontradas por NSGAII-FM. Esse experimento também alcançou soluções com melhor *trade-off* que NSGAII. Entre as soluções de pior percentual de melhoria em FM as mais altas porcentagens para CM foram alcançadas pelo NSGAII-FM para MM-1, MM-2 e LPS-BET, indicando que o Feature-driven Operator também permite alcançar PLAs mais estáveis de acordo com CM.

A questão **RQ3** envolve uma análise da eficiência das métricas usadas no modelo de avaliação para obter projetos de PLA com as propriedades desejadas. Nesse caso, PLAs mais estáveis e mais modulares em termos de características. As seguintes conclusões foram alcançadas para essa questão:

- Uma comparação entre as soluções com os melhores e os piores valores de FM mostra que FM é sensível à modularização de características nos projetos de PLA. As PLAs com melhores valores de FM têm componentes com maior coesão baseada em característica e maior número de características modularizadas, levando a um menor número de características espalhadas e entrelaçadas no projeto. Uma comparação entre as soluções de menor ED contra as soluções de pior porcentagem de melhoria em FM mostra que o número de características espalhadas nas primeiras soluções é menor do que nas últimas. Além disso, as soluções que nitidamente sofreram a ação dos operadores de busca AddComponent e MoveOperation têm pior valor de FM porque esses operadores não se preocupam com a modularização de características.

- Algumas métricas da função CM não foram preponderantes na avaliação do *fitness* das soluções encontradas para as PLAs utilizadas. As alterações nos valores das métricas convencionais CDepIn e CDepOut não foram representativas já que as referidas métricas não sofrem a ação dos operadores de busca porque a maioria dos relacionamentos entre classes nas PLAs utilizadas é do tipo generalização. O valor de NumOps se tornou mais alto em várias soluções que têm um número menor de interfaces mantendo o mesmo número de operações. Apesar de isso não ser um problema, essa mudança de valor não foi expressivo no *fitness*, uma vez que somente três soluções tiveram valores de CM piores que o valor original.

As respostas para as três questões de pesquisa são discutidas com mais detalhes no artigo incluído no Apêndice C.

5.3 Estudo 2: Avaliando o Operador de Cruzamento

Tendo em mente que o benefício do operador de cruzamento para a otimização de projeto de PLA ainda é uma questão de pesquisa em aberto, é preciso analisar a viabilidade de empregar, durante o processo evolutivo, o operador Feature-driven Crossover proposto para a instanciação da MOA4PLA usando MOEAs.

O segundo estudo empírico foi realizado para avaliar essa questão de pesquisa. Se o benefício do referido operador não for realmente interessante pode ser mais vantajoso não dispender tempo de processamento na sua aplicação. As configurações do estudo e os principais resultados alcançados são abordados nas próximas seções. Informações mais detalhadas sobre o estudo, tais como aspectos de implementação e os conjuntos PF_{true} e PF_{known} encontrados pelos algoritmos, podem ser obtidas no Apêndice D.

5.3.1 Configuração do Estudo 2

Para responder a questão de pesquisa desse estudo, dois experimentos usando diferentes versões do NSGAII foram executados, incluindo diferentes combinações de operadores de busca. No primeiro experimento adotou-se um método sem cruzamento no qual fo-

ram aplicados somente os operadores de mutação. Esse experimento foi denominado NSGAII-M. O segundo experimento, denominado NSGAII-MC, usa tanto mutação como o operador de cruzamento Feature-driven Crossover. Assim como nos trabalhos relacionados [81, 90], o percentual de probabilidade de aplicação do cruzamento é baixo. Após uma configuração seguindo resultados empíricos foi definido o valor de 0.1, que corresponde a 10% de probabilidade de aplicação do operador. Todas as PLAs descritas na Seção 5.1 foram utilizadas no estudo que envolve as funções de *fitness* $FM(pla)$ e $CM(pla)$.

Desta versão da implementação da OPLA-Tool em diante, foi resolvida paliativamente a limitação relacionada à identificação das camadas arquiteturais às quais pertencem os elementos alvo das mutações. O procedimento adotado consiste em não permitir que elementos arquiteturais de uma camada sejam movidos para outra. Dessa forma, evita-se que inconsistências relacionadas às regras do estilo arquitetural sejam geradas. No entanto, soluções mais completas devem ser proporcionadas pelo módulo OPLA-ArchStyles da OPLA-Tool que, como mencionado anteriormente, encontra-se em desenvolvimento em outro trabalho de pesquisa.

A limitação dos operadores de mutação com relação aos relacionamentos de generalização foi alterada. A partir desse estudo os operadores de mutação podem ser aplicados às classes participantes em generalizações. Porém, atributos e métodos dessas classes não podem ser movidos. Desse modo, a única mudança que pode ser aplicada a uma hierarquia de herança é mover a hierarquia como um todo para um outro componente. A aplicação do operador de cruzamento ficou restrita às hierarquias de herança contidas um mesmo componente. Dessa forma, o cruzamento não foi aplicado se a característica selecionada para realizar o cruzamento estivesse associada a alguma classe participante de generalizações envolvendo mais de um componente.

5.3.2 Análise dos Resultados do Estudo 2

Uma síntese dos resultados deste estudo empírico é apresentada nessa seção. Os resultados detalhados, incluindo *fitness* das soluções encontradas pelos experimentos, tempo de execução e os conjuntos PF_{true} de cada PLA, estão relatados no Apêndice D.

Todos os conjuntos PF_{true} obtidos são compostos por soluções alcançadas por um único experimento, isto quer dizer que as soluções de um experimento dominam todas as soluções do outro. Sob esse ponto de vista, NSGAII-MC teve melhor desempenho que NSGAII-M, exceto para as PLAs LPS-BET e MM-1. Apesar disso, em termos de tempo de execução, o experimento que inclui o operador de cruzamento (NSGAII-MC) demora cerca de 9% mais de tempo que o NSGAII-M para concluir o processo de busca.

Em alguns casos, NSGAII-M encontrou um menor número de soluções que NSGAII-MC. Nesses casos, as populações alcançadas pelo NSGAII-M são mais homogêneas em termos de dispersão das soluções no espaço de busca. Isso pode ser uma evidência de que o algoritmo ficou preso num ótimo local, sem conseguir explorar outras áreas do espaço de busca. Por outro lado, NSGAII-M teve melhor desempenho que NSGAII-MC para LPS-BET e MM-1. No caso da LPS-BET os dois experimentos alcançaram resultados muito similares, em termos de heterogeneidade das populações e número de soluções encontradas. LPS-BET é a maior PLA. Talvez por isso NSGAII-M não tenha ficado preso num ótimo local, permitindo que ele obtivesse melhores soluções do que para as outras PLAs.

A Tabela 5.7 apresenta os valores médios de hipervolume considerando as 30 execuções de cada experimento. O número entre parênteses representa o desvio padrão. Devido à natureza estocástica dos algoritmos, para realizar uma comparação estatística, o teste de Wilcoxon [26] foi utilizado em um nível de significância de 5%. A última coluna da tabela apresenta o *p-value* de cada teste. Os valores apresentados em negrito correspondem àqueles em que há diferença significativa entre os experimentos. Quanto a esse indicador, MM-3 é a única PLA para a qual não há diferença estatística entre os experimentos. NSGAII-M é o melhor para LPS-BET e MM-1 e NSGAII-MC é o melhor para as outras seis PLAs.

O indicador ED corrobora os resultados de hipervolume para cada PLA. A Tabela 5.8 mostra o *fitness* da solução ideal, bem como a menor ED e o *fitness* da solução mais próxima da ideal obtida em cada experimento. Há uma grande distância entre as soluções com a menor ED encontrada pelos experimentos para a maioria dos projetos de PLA. As distâncias são mais semelhantes para PLAs onde NSGAII-M foi o melhor (LPS-BET

e MM-1) e para a MM-3, onde não há diferença entre os experimentos em termos de hipervolume.

Tabela 5.7: Estudo 2 - Média do Hipervolume

PLA	NSGAII-M	NSGAII-MC	
	Média (Desvio Padrão)	**Média (Desvio Padrão)**	*p-value*
AGM-1	162611.1 (9590.6)	**186297.1** (14140.7)	9.54e-07
AGM-2	272010.2 (9158.0)	**298136.7** (21537.7)	3.82e-04
AGM-3	54854.9 (1781.6)	**61571.3** (6783.0)	1.17e-02
AGM-4	179225.6 (4954.2)	**202557.3** (11506.0)	7.61e-16
MM-1	**237509.7** (19400.1)	221099.6 (27746.0)	0.04789
MM-2	455344.2 (26797.4)	**473216.0** (39243.6)	0.007301
MM-3	16528.3 (1392.3)	17467.3 (2276.8)	0.1153
MM-4	29446.2 (1489.9)	**31877.2** (4060.3)	0.03577
LPS-BET	**346436.9** (4374.0)	341895.4 (5568.2)	0.005302

Tabela 5.8: Estudo 2 - Soluções com as menores EDs

PLA	Solução Ideal	NSGAII-M		NSGAII-MC	
		ED	*Fitness*	**ED**	*Fitness*
AGM-1	(131.0,13.1)	49,1	(180.0,16.3)	**3,2**	**(131.0,16.3)**
AGM-2	(163.0,14.3)	105,3	(268.0,23.0)	**5,0**	**(164.0,19.2)**
AGM-3	(101.0,12.3)	68,1	(169.0,16.5)	**2,6**	**(101.0,15.0)**
AGM-4	(160.0,14.2)	66,3	(226.0,20.5)	**4,4**	**(160.0,18.6)**
MM-1	(337.0,25.0)	**0,0**	**(337.0,25.0)**	7,0	(344.0,25.5)
MM-2	(245.0,14.6)	54,5	(299.0,22.0)	**4,4**	**(245.0,19.0)**
MM-3	(356.0,30.2)	10,7	(366.0,34.3)	**3,5**	**(356.0,33.8)**
MM-4	(310.0,24.3)	21,3	(331.0,27.9)	**3,5**	**(320.0,25.8)**
LPS-BET	(496.0,130.4)	**18,8**	**(506.0,146.4)**	21,4	(509.0,147.4)

Na análise qualitativa, foram comparadas as PLAs de menor ED obtidas em cada experimento, a fim de analisar quais tipos de mudanças foram realizadas a partir do projeto original, especialmente no que se refere à modularização de características.

As soluções encontradas pelos dois experimentos são semelhantes, tendo em vista o número de componentes, de interfaces, de classes e de componentes associados com uma única característica. A maior diferença entre elas refere-se à modularização de características. Algumas características estão mais entrelaçadas e dispersas em soluções do NSGAII-M, o que leva a uma menor coesão baseada em características. Além disso, a pior modularização de características implica em maior acoplamento. É por isso que o *fitness* das soluções do NSGAII-MC é melhor do que das soluções do NSGAII-M para as métricas convencionais e dirigidas à características (CM e FM).

Diferenças desse tipo são ilustradas nas Figuras 5.3 e 5.4, as quais mostram excertos das soluções de menor ED encontradas para a PLA MM-2. Nos dois casos, é possível visualizar os componentes que modularizam as características *create/delete* e *view/playMedia* na camada de sistema (componentes com o sufixo Ctrl). Na solução do NSGAII-M (Figura 5.3), *create/delete* está entrelaçada com as características *SMS* e *labelMedia* enquanto *view/playMedia* está entrelaçada com *SMS* e *linkMedia* em nível de operação. Além disso, *SMS* está espalhada em duas interfaces de diferentes componentes. Por outro lado, na solução do NSGAII-MC (Figura 5.4), *SMS* foi modularizada no *Component*3679*Ctrl*, aumentando a coesão baseada em característica (métrica LCC) de *create/delete* e *view/playMedia*. Apesar de não aparecerem na figura, há outros pontos a favor da solução obtida pelo NSGAII-MC, por exemplo, as operações associadas com *linkMedia* foram modularizadas em outro componente.

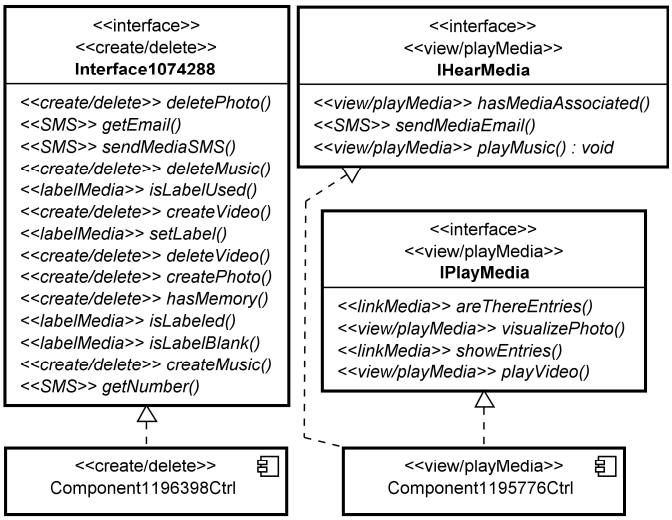


Figura 5.3: Estudo 2 - Solução Parcial do NSGAII-M para MM-2

As duas soluções são melhores do que a PLA original. Os resultados mostram que os operadores de mutação são suficientes para otimizar um projeto de PLA. NSGAII-M alcança soluções satisfatórias em termos de *fitness* e qualidade das soluções. Os valores de FM e CM são muito melhores que os originais para todas as PLAs. No entanto, de forma geral, as soluções do NSGAII-MC são melhores que as do NSGAII-M como ilustrado no exemplo acima. Então, é possível afirmar que o operador Feature-driven Crossover permite alcançar boas soluções com relação à modularização de características de LPS.

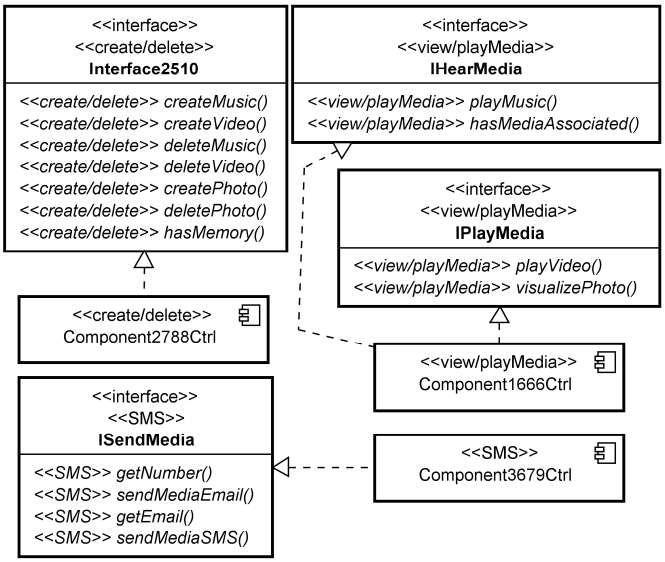


Figura 5.4: Estudo 2 - Solução Parcial do NSGAII-MC para MM-2

Apesar disso, o referido operador tem seu ponto fraco: a garantia de consistência das soluções geradas é mais difícil quando se aplica um operador de cruzamento. Algumas operações foram perdidas em certas soluções do NSGAII-MC. Uma explicação para que isso tenha acontecida é devido à exclusão, na solução filha, de interfaces inteiras associadas com a característica em modularização (linhas 8 e 12 do Algoritmo 4). Depois, as interfaces da solução pai, associadas com a característica em modularização, são adicionadas à solução filha (linhas 9 e 13), mas como elas eventualmente estavam melhor modularizadas para a característica em questão, operações associadas a outras características que foram excluídas no primeiro momento, não foram adicionadas depois.

Isto sugere que alguma melhoria é necessária para evitar casos como este. No estudo objeto desta tese, as características estão mapeadas para elementos arquiteturais em granularidade fina, ou seja, todos os elementos arquiteturais (componentes, interfaces, operações de interfaces, classes, atributos e métodos) estão associados com pelo menos uma característica. No entanto, as características são geralmente mapeadas em granularidade grossa em níveis de componente, interface e classe, o que impede esse tipo de ocorrência.

Mesmo assim, a aplicação do operador Feature-driven Crossover no NSGAII-MC permitiu encontrar resultados ainda melhores que no experimento utilizando somente mutação.

Nesse caso, as populações são mais heterogêneas, sem prejudicar a qualidade das soluções. A análise qualitativa mostrou que o operador de cruzamento proposto também ajuda a encontrar soluções de maior qualidade em termos de coesão baseada em características e menor espalhamento e entrelaçamento de características no projeto da PLA.

5.4 Estudo 3: Avaliando o PAES

O MOEA PAES (descrito na Seção 3.1.2) segue um processo evolutivo diferente do NSGAII, porque se baseia apenas na mutação e não envolve o conceito de população. As melhores soluções ficam num arquivo externo e são utilizadas durante o processo evolutivo. O PAES alcançou melhores resultados que o NSGAII para resolver outros tipos de problemas, tais como, planejamento de projeto de software [14] e integração e teste de software [6]. Por esses motivos, é interessante averiguar se ele consegue ter um desempenho melhor que o NSGAII no contexto da MOA4PLA.

Isso motivou o desenvolvimento do terceiro estudo empírico para comparar o desempenho do PAES em relação as duas versões do NSGAII utilizadas no estudo anterior: NSGAII-M e NSGAII-MC. Caso o PAES alcance tão bons resultados quanto o NSGAII-MC, o problema de inconsistência em algumas soluções geradas pelo operador de cruzamento deixaria de existir já que o PAES somente aplica operadores de mutação.

5.4.1 Configuração do Estudo 3

Certos parâmetros adotados para o PAES foram os mesmos utilizados para o NSGAII, tais como, número de gerações = 300 e número de avaliações de *fitness* = 30000. O tamanho do arquivo externo foi definido como 100 para ficar igual ao tamanho da população do NSGAII. Duas taxas de mutação foram experimentadas: 0.9 e 1.0. Assim, no experimento denominado PAES-90 adotou-se 90% de probabilidade de aplicação da mutação. No experimento PAES-100, essa probabilidade foi de 100%.

O modelo de avaliação utilizado foi o mesmo do estudo anterior. Ele envolve as funções de *fitness* $FM(pla)$ e $CM(pla)$. O estudo envolveu as mesmas PLAs utilizadas no estudo

anterior e os resultados do PAES foram comparados com os resultados alçancados pelo NSGAII-M e pelo NSGAII-MC. Os resultados foram comparados em termos de *fitness* e da solução de menor ED. Esses resultados são apresentados na próxima seção.

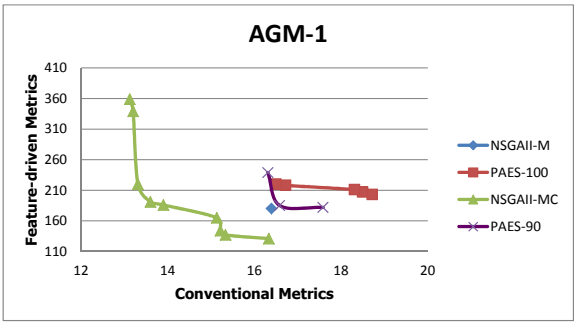
5.4.2 Análise dos Resultados do Estudo 3

As Figuras 5.5 e 5.6 apresentam os conjuntos de soluções não-dominadas retornadas por cada experimento (PF_{known}). Nesse estudo, o objetivo era minimizar as funções $FM(pla)$ e $CM(pla)$, portanto, quanto mais próxima do ponto (0,0) estiver a curva, melhor o resultado do experimento. Em todos os casos as versões do NSGAII (NSGAII-MC e NSGAII-M) alcançaram melhores soluções em termos de *fitness* do que as versões do PAES (PAES-90 e PAES-100).

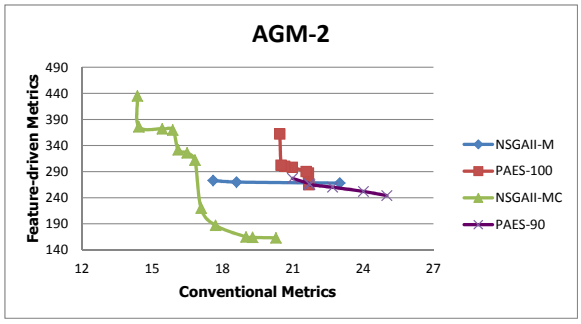
Entre as duas versões do PAES, na grande maioria dos casos, a versão utilizando 90% de probabilidade de mutação (PAES-90) alcançou melhores resultados do que a versão que adota 100% de probabilidade (PAES-100). Em termos de tempo de execução, os tempos requeridos pelo PAES-90 e pelo PAES-100 foram muito similares aos tempos dispendidos pelas duas versões do NSGAII (NSGAII-M e NSGAII-MC).

Para complementar a análise, a Tabela 5.9 apresenta as soluções de menor ED dos experimentos NSGAII-M, NSGAII-MC e PAES-90. As menores EDs estão destacadas em negrito. PAES conseguiu alcançar soluções com ED melhores que as soluções alcançadas pelo NSGAII-M somente para as PLAs AGM-4, MM-2, MM-3 e MM-4. No entanto, a análise baseada nesse indicador confirma que, no contexto deste estudo, o PAES não consegue resultados melhores que NSGAII-MC.

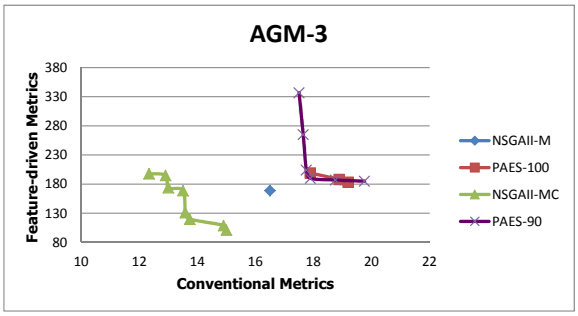
Dessa forma, é possível concluir que, considerando as PLAs e o modelo de avaliação utilizados, o NSGAII é melhor do que o PAES para otimizar o projeto de PLA no contexto da MOA4PLA.



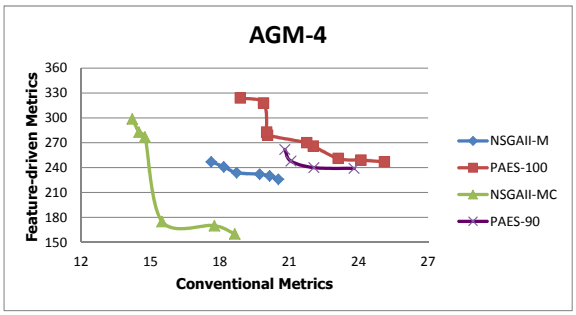
(a) AGM-1



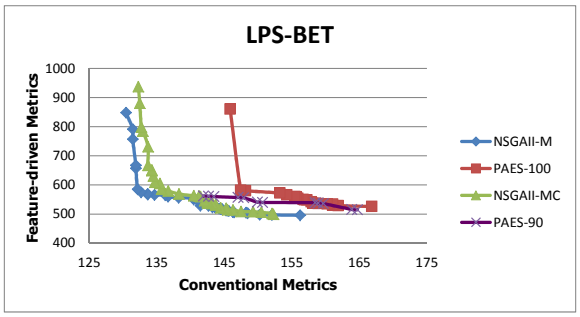
(b) AGM-2



(c) AGM-3



(d) AGM-4

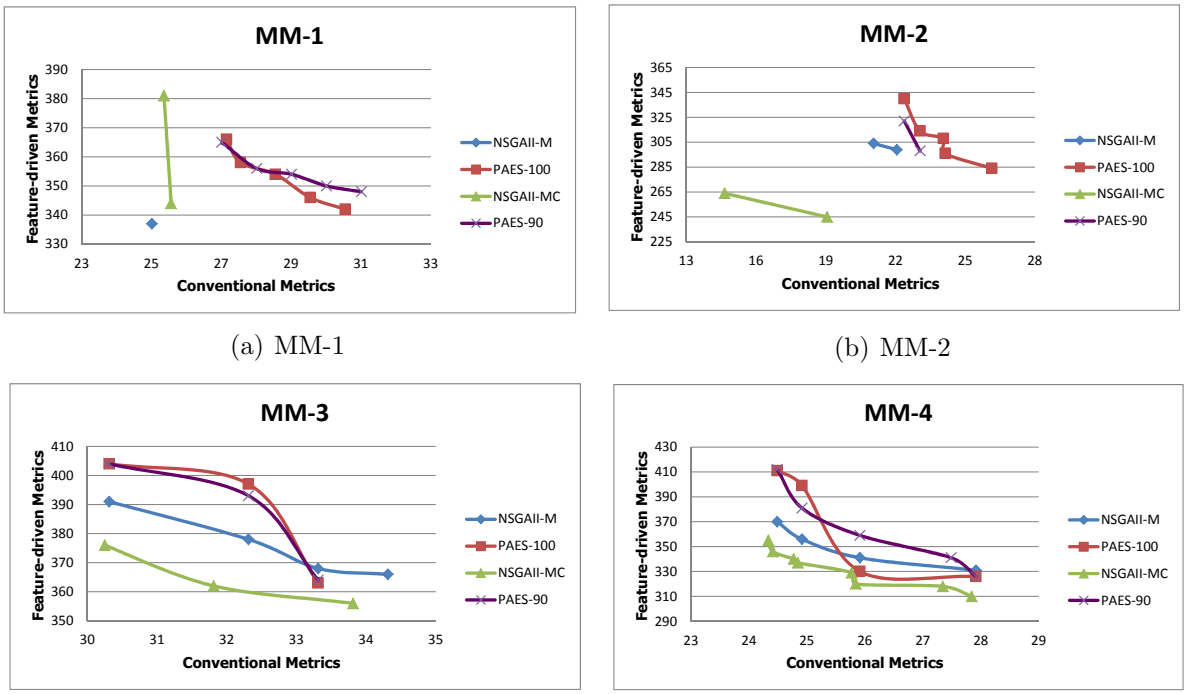


(e) LPS-BET

Figura 5.5: Estudo 3 - Espaço de Busca para PLAs da AGM e LPS-BET

Tabela 5.9: Estudo 3 - Soluções com menor ED

PLA	Solução Ideal	*Fitness* Original	NSGAII-M		NSGAII-MC		PAES-90	
			ED	*Fitness*	ED	Fitness	ED	Fitness
AGM-1	(131.0, 13.1)	(391.0, 31.8)	49.1	(180.0, 16.3)	**3.2**	(131.0, 16.3)	51.1	(182.0, 17.5)
AGM-2	(163.0, 14.3)	(499.0, 35.1)	105.3	(268.0, 23.0)	**5.0**	(164.0, 19.2)	81.6	(244.0, 25.0)
AGM-3	(101.0, 12.3)	(360.0, 31.8)	68.1	(169.0, 16.5)	**2.6**	(101.0, 15.0)	84.3	(185.0, 19.7)
AGM-4	(160.0, 14.2)	(462.0, 35.1)	66.3	(226.0, 20.5)	**4.4**	(160.0, 18.6)	79.5	(239.0, 23.8)
MM-1	(337.0, 25.0)	(608.0, 39.3)	**0,0**	(337.0, 25.0)	7.0	(344.0, 25.5)	12.5	(348.0, 31.0)
MM-2	(245.0, 14.6)	(534.0, 33.4	54.5	(299.0, 22.0)	**4.4**	(245.0, 19.0)	53.6	(298.0, 23.0)
MM-3	(356.0, 30.2)	(434.0, 39.3)	10.7	(366.0, 34.3)	**3.5**	(356.0, 33.8)	8.5	(364.0, 33.3)
MM-4	(310.0, 24.3)	(430.0, 33.4)	21.3	(331.0, 27.9)	**3.5**	(320.0, 25.8)	16.3	(326.0, 27.9)
LPS-BET	(496.0, 130.4)	(674.0, 176.8)	**18.8**	(506.0, 146.4)	21.4	(509.0, 147.4)	38.9	(516.0, 163.8)



(a) MM-1

(b) MM-2

(c) MM-3

(d) MM-4

Figura 5.6: Estudo 3 - Espaço de Busca para PLAs da Mobile Media

5.5 Estudo 4: Avaliando as funções de *fitness*

O objetivo da MOA4PLA de gerar projetos extensíveis e com boa modularização de características motivou a realização do último estudo empírico. Como o modelo de avaliação pode ser configurado para usar diferentes funções de *fitness*, nesse estudo esse modelo foi configurado de três formas diferentes, a fim de responder as seguintes questões de pesquisa:

RQ1: A presença da função $Ext(pla)$ no modelo de avaliação contribui para alcançar projetos de PLA mais extensíveis? Como essa função não foi utilizada nos estudos empíricos anteriores, convém avaliar sua adequação no contexto da MOA4PLA.

RQ2: A função $Ext(pla)$ compromete a obtenção de projetos de PLA com melhor modularização de características? É possível que a métrica de extensibilidade seja conflitante com as métricas de modularização de características, de modo que ao otimizar uma métrica haja prejuízo da outra.

RQ3: Uma maior probabilidade de aplicação do Feature-driven Operator melhora a modularização das características? A probabilidade de aplicação do referido operador foi dobrada nesse último estudo. Neste sentido, é importante averiguar

se as PLAs obtidas nesse último estudo ficaram melhor modularizadas com relação às características do que as do segundo estudo empírico.

RQ4: Quais as peculiaridades dos projetos de PLAs gerados pelas diferentes combinações de funções de *fitness*? O alvo dessa questão é a identificação de diretrizes que apontem para o usuário da MOA4PLA qual modelo de avaliação é mais adequado para gerar projetos de PLA com determinadas peculiaridades. Essas diretrizes podem facilitar o uso da abordagem proposta.

RQ5: Os experimentos realizados conseguem obter alternativas de projeto de PLA consideradas boas com relação aos objetivos da MOA4PLA? A abordagem proposta tem como objetivo avaliar e melhorar projetos de PLAs no que tange a modularização de características, estabilidade do projeto e extensibilidade de LPS. Os experimentos envolvendo os operadores de mutação e de cruzamento e diferentes modelos de avaliação permitem analisar as soluções alcançadas em relação às PLAs originais fornecidas como entrada para o processo de otimização.

A configuração do estudo é descrita na próxima seção e os resultados são apresentados e discutidos na Seção 5.5.2.

5.5.1 Configuração do Estudo 4

O estudo foi realizado utilizando todas as PLAs relacionadas na Seção 5.1. O modelo de avaliação foi configurado de três diferentes maneiras. Na primeira configuração foram utilizados dois objetivos sendo $FM(pla)$ e $CM(pla)$. Na segunda configuração também foram utilizados dois objetivos: $FM(pla)$ e $Ext(pla)$ (referente à métrica de extensibilidade de LPS apresentada na Equação 4.3). Na terceira configuração, três objetivos foram adotados: $FM(pla)$, $CM(pla)$ e $Ext(pla)$. Cada configuração do modelo de avaliação foi executada nas duas versões do NSGAII: NSGAII-M e NSGAII-MC. Assim, cada configuração foi avaliada utilizando somente operadores de mutação e utilizando mutação e cruzamento. Portanto, seis experimentos foram executados nesse estudo, nomeados como segue:

- NSGAII-M-FC: versão do NSGAII que utiliza somente os operadores de mutação e o modelo de avaliação com as funções $FM(pla)$ e $CM(pla)$;

- NSGAII-M-FE: versão do NSGAII que utiliza somente os operadores de mutação e o modelo de avaliação com as funções objetivo $FM(pla)$ e $Ext(pla)$;

- NSGAII-M-FCE: versão do NSGAII que utiliza somente os operadores de mutação e o modelo de avaliação com as funções $FM(pla)$, $CM(pla)$ e $Ext(pla)$;

- NSGAII-MC-FC: versão do NSGAII que utiliza tanto os operadores de mutação como o operador de cruzamento. O modelo de avaliação é composto pelas funções $FM(pla)$ e $CM(pla)$;

- NSGAII-MC-FE: versão do NSGAII que utiliza tanto os operadores de mutação como o operador de cruzamento. O modelo de avaliação é composto pelas funções $FM(pla)$ e $Ext(pla)$; e,

- NSGAII-MC-FCE: versão do NSGAII que utiliza tanto os operadores de mutação como o operador de cruzamento. O modelo de avaliação é composto pelas funções $FM(pla)$, $CM(pla)$ e $Ext(pla)$.

Os mesmos parâmetros utilizados no segundo estudo empírico foram adotados para configurar o NSGAII, tais como, número de avaliações de *fitness*, tamanho da população, probabilidades de mutação e de cruzamento. Nos demais estudos, todos os operadores de mutação tinham a mesma probabilidade de ser aplicados durante o processo evolutivo. Porém, nesse estudo, a probabilidade de aplicação do Feature-driven Operator foi dobrada. Dessa forma, este operador tem o dobro de chance de ser aplicado em relação aos operadores de mutação convencionais. Essa decisão foi tomada porque nos estudos anteriores percebeu-se que seria possível modularizar um número maior de características.

A limitação da implementação do operador de cruzamento que restringia a sua aplicação somente às hierarquias de herança contidas em um mesmo componente foi resolvida. Desse modo, nesse estudo, o referido operador foi aplicado independentemente do tipo e abrangência dos relacionamentos envolvidos.

Como mencionado na Seção 4.4.2, duas novas restrições foram adicionadas nesse estudo para evitar que soluções inválidas fossem geradas nos experimentos usando o operador de cruzamento Feature-driven Crossover. Uma solução foi considerada inválida quando: (i) possuía alguma interface não vazia desconectada no projeto, isto é, sem clientes ou *suppliers* e, (ii) algum ponto de variação não estava presente no projeto. As soluções inválidas foram contadas e descartadas porque recuperá-las seria mais custoso.

Como o número de objetivos e as métricas utilizadas no modelo de avaliação são diferentes, não é possível comparar os resultados dos experimentos utilizando os indicadores de qualidade. Neste sentido, decidiu-se por avaliar os resultados utilizando as métricas que fornecem indicadores a respeito da elegância do projeto por meio da função $Eleg(pla)$ definida na Equação 4.4. Isso se justifica porque essas métricas não foram utilizadas no processo de otimização, portanto, elas podem ser empregadas como um indicador para comparar as soluções obtidas nos diferentes experimentos executados.

5.5.2 Análise dos Resultados do Estudo 4

O tempo médio de execução dos experimentos é apresentado na Tabela E.1 (Apêndice E). As três configurações de modelo de avaliação demandaram tempos de execução similares. Porém, na maior parte das vezes os experimentos NSGAII-M-FE e NSGAII-MC-FE demandaram menor tempo de execução que os demais experimentos. A variação do número de objetivos não influenciou no tempo de execução dos experimentos. Como esperado, a presença do operador de cruzamento implica em um maior tempo de execução do que os experimentos que aplicam somente mutação.

As Tabelas E.2 e E.3 (Apêndice E) apresentam, para cada experimento, as soluções não dominadas e seus respectivos *fitness*. Analisando somente os conjuntos PF_{known} de cada experimento não é possível determinar um comportamento padrão em relação ao número de soluções encontradas. Entretanto, pode-se destacar que NSGAII-M-FE e NSGAII-MC-FE retornaram somente uma solução para cada PLA e, houve somente um caso no qual NSGAII-M-FC, NSGAII-M-FCE e NSGAII-MC-FC retornaram somente uma solução.

Tabela 5.10: Estudo 4 - Melhores soluções do NSGAII-M

PLA	NSGAII-M-FE		NSGAII-M-FC		NSGAII-M-FCE	
	Elegância	***Fitness***	**Elegância**	***Fitness***	**Elegância**	***Fitness***
AGM-1	**0.26101**	(**275.0**, 0.52)	**0.26029**	(278.0, 16.66)	0.26978	(**277.0**, 18.16, 0.52)
			0.26029	(**276.0**, 16.72)	**0.26101**	(280.0, 16.37, 0.52)
AGM-2	**0.33635**	(**321.0**, 0.61)	**0.31870**	(325.0, 19.44)	**0.31774**	(**323.0**, 20.45, 0.61)
			0.3262	(**324.0**, 19.46)		
AGM-3	**0.26085**	(**196.0**, 0.52)	**0.26911**	(**178.0**, 15.80)	**0.27026**	(**187.0**, 15.80, 0.52)
AGM-4	**0.31901**	(**243.0**, 0.61)	**0.31901**	(**239.0**, 18.04)	**0.31757**	(**243.0**, 19.66, 0.61)
MM-1	**0.16888**	(**396.0**, 0.77)	**0.16188**	(**396.0**, 25.53)	**0.16222**	(**390.0**, 25.53, 0.77)
MM-2	**0.17428**	(**365.0**, 0.77)	**0.16715**	(**370.0**, 22.75)	**0.16715**	(**370.0**, 22.75, 0.77)
MM-3	**0.17666**	(**362.0**, 0.77)	**0.17234**	(**347.0**, 32.94)	**0.16204**	(**356.0**, 31.94, 0.77)
MM-4	**0.17834**	(**305.0**, 0.77)	**0.17170**	(309.0, 26.51)	0.18461	(**301.0**, 26.80, 0.77)
			0.17170	(319.0, 24.45)	**0.18385**	(322.0, 23.32, 0.77)
			0.17170	(314.0, 25.1)		
			0.17170	(310.0, 25.78)		
			0.18294	(**308.0**, 27.52)		
LPS-BET	**0.5071**	(**526.0**, 1000.0)	0.51825	(**519.0**, 144.87)	**0.50621**	(**528.0**, 148.25, 1000.0)
			0.51778	(561.0, 129.25)		
			0.51778	(564.0, 129.06)		

Tabela 5.11: Estudo 4 - Melhores soluções do NSGAII-MC

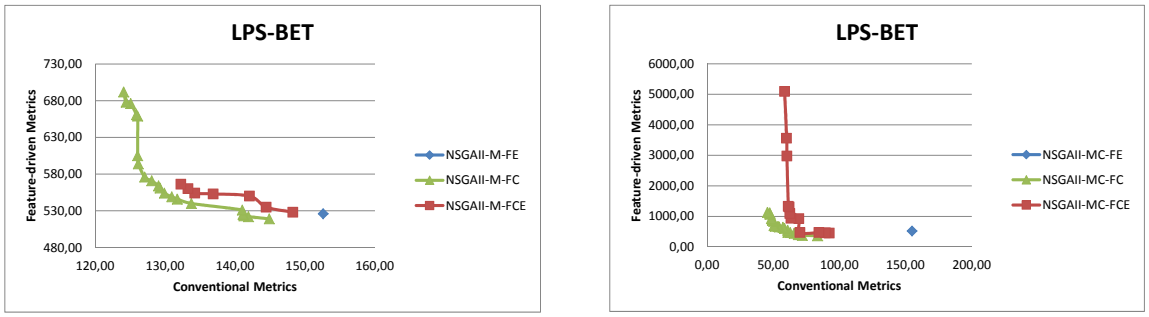
PLA	NSGAII-MC-FE		NSGAII-MC-FC		NSGAII-MC-FCE	
	Elegância	***Fitness***	**Elegância**	***Fitness***	**Elegância**	***Fitness***
AGM-1	**0.2688**	(**271.0**, 0.52)	**0.2610**	(280.0, 16.66)	**0.2420**	(**276.0**, 16.72, 0.52)
			0.2645	(**273.0**, 18.11)		
AGM-2	**0.2764**	(**313.0**, 0.61)	**0.2920**	(**302.0**, 18.0)	**0.2698**	(**321.0**, 20.34, 0.61)
AGM-3	**0.2691**	(**196.0**, 0.52)	**0.2464**	(194.0, 16.88)	**0.2608**	(**193.0**, 15.80, 0.52)
			0.2610	(**169.0**, 19.15)		
AGM-4	**0.2866**	(**232.0**, 0.61)	**0.2950**	(214.0, 19.54)	**0.2703**	(**216.0**, 21.54, 0.61)
			0.2950	(**213.0**, 22.54)		
MM-1	**0.1688**	(**407.0**, 0.77)	**0.1494**	(548.0, 25.20)	**0.1337**	(383.0, 21.29, 0.54)
			0.1665	(**385.0**, 27.40)	0.1615	(**356.0**, 26.27, 0.77)
MM-2	**0.1742**	(**362.0**, 0.77)	**0.1671**	(1084.0, 21.75)	**0.1463**	(1042.0, 19.90, 0.77)
			0.1671	(1065.0, 22.22)	0.1742	(**338.0**, 22.58, 0.77)
			0.1742	(**364.0**, 22.75)		
MM-3	**0.1757**	(**362.0**, 0.77)	**0.1271**	(169.0, 25.54)	**0.1124**	(219.0, 19.16, 0.17)
			0.1618	(**175.0**, 24.52)	0.1320	(**136.0**, 24.25, 1.74)
MM-4	**0.1798**	(**305.0**, 0.77)	**0.1159**	(**197.0**, 22.22)	**0.1720**	(374.0, 19.86, 0.44)
			0.1159	(221.0, 20.95)	**0.1720**	(305.0, 20.03, 0.44)
			0.1159	(207.0, 21.56)	0.1776	(**201.0**, 21.78, 0.77)
LPS-BET	**0.5154**	(**524.0**, 1000.0)	**0.4331**	(542.0, 60.88)	**0.4600**	(924.0, 69.46, 1000.0)
			0.4665	(**367.0**, 83.49)	0.4885	(**453.0**, 92.45, 1000.0)

O *fitness* das soluções que têm a melhor elegância são apresentados nas Tabelas 5.10 e 5.11. Além disso, também são apresentadas as soluções que têm a melhor modularização de características de acordo com o *fitness* (menor valor de FM). Os menores valores estão em negrito. Em alguns casos, a solução mais elegante é também a que tem a melhor modularização de características. Houve também casos em que várias soluções alcançaram

o mesmo valor para as métricas de elegância. Nesses casos, todas essas soluções estão relacionadas nas referidas tabelas.

Comparando as duas versões do NSGAII: NSGAII-M e NSGAII-MC, destaca-se que:

- Para a PLA AGM-2, os experimentos NSGAII-M-FCE e NSGAII-MC-FCE têm maior diversidade de soluções que os demais experimentos. Isso mostra que para essa PLA, o modelo de avaliação com as três funções objetivos é mais adequado.
- Para a LPS-BET, os experimentos NSGAII-M-FC e NSGAII-MC-FC obtêm os melhores resultados em termos de convergência, pois suas soluções estão mais próximas dos objetivos mínimos. Esse fato pode ser observado na Figura 5.7 que mostra as soluções no espaço de busca.
- Na maioria das PLAs, as soluções encontradas pelos experimentos usando as funções FCE (FM, CM e Ext) e FC (FM e CM) são mais elegantes que as encontradas por FE (FM e Ext).



(a) NSGAII-M (b) NSGAII-MC

Figura 5.7: Estudo 4 - Espaço de Busca para LPS-BET

Nas duas próximas seções são analisados os resultados dos experimentos que aplicam somente mutação e dos experimentos que empregam os operadores de mutação e de cruzamento, respectivamente.

5.5.2.1 Otimização somente com mutação

Nesta seção os resultados obtidos pelos experimentos que usam somente operadores de mutação (NSGAII-M-FE, NSGAII-M-FC e NSGAII-M-FCE) são analisados. Além dos

dados quantitativos, uma análise qualitativa foi realizada para as soluções com melhor elegância e melhor MSI e com diferentes valores de extensibilidade da AGM-1, AGM-4, MM-1 e MM-3. Nessa análise foram considerados os seguintes aspectos: incompletude e/ou inconsistência considerando a PLA original, número de características modularizadas em nível de componente, número de componentes com características entrelaçadas em nível de componente, número de componentes da PLA, distribuição dos atributos e métodos pelas classes do projeto, a existência de componentes desconectados do restante da PLA, comparação da extensibilidade em relação à PLA original.

Extensibilidade das Soluções Encontradas

Como mencionado anteriormente, todas as PLAs do experimento utilizando somente as funções FM e Ext retornaram uma única solução não-dominada em todas as trinta rodadas. Isso ocorreu, porque o valor da métrica de extensibilidade não se alterou em relação ao valor original. Dessa forma, somente a função FM foi otimizada não gerando qualquer compromisso entre as funções. No experimento NSGAII-M-FCE, todas as soluções têm extensibilidade igual a original, com exceção de uma solução de AGM-2 e de uma solução de MM-1, as quais têm extensibilidade pior que a original.

O valor da métrica de extensibilidade não foi melhorado porque os operadores de busca aplicados não alteram a relação de métodos abstratos por métodos concretos nas classes envolvidas em generalizações, e a grande maioria dos pontos de variação medidos pela referida métrica estão justamente em classes envolvidas neste tipo de relacionamento.

Todos os pontos de variação de AGM-1 e AGM-3 estão em classes com generalização. Na AGM-2 e na AGM-4 somente a classe *Player* é ponto de variação não envolvido em herança. Uma solução de AGM-2 tem pior extensibilidade porque um método concreto foi movido da classe $AnimationLoopMgr$ para *Game* alterando a proporção entre métodos abstratos e concretos dessa última classe, a qual é ponto de variação.

Em todas as versões da MM, somente a classe *Entry* é ponto de variação não envolvido em herança. Na solução de MM-1 cuja extensibilidade piorou, um atributo e dois métodos concretos foram movidos da classe $User$ para $Media$ alterando a proporção entre métodos abstratos e concretos dessa última classe, a qual é ponto de variação. Inclusive, essa

solução tem pior modularização de características que a outra solução encontrada. O projeto ficou pior e não faz muito sentido ter métodos de *User* em *Media*.

Tanto na referida solução de AGM2 como na solução de MM-1, essas alterações devem ter sido causadas pelos operadores FeatureMutation e/ou MoveMethod. Apesar de verificar se as classes origem da aplicação da mutação são pontos de variação ou variantes, a mesma verificação não é realizada para as classes destino da mutação. Dessa forma, previne-se a remoção de elementos de classes que são pontos de variação e variantes, porém, os operadores podem adicionar atributos ou métodos nesses tipos de classes.

Na LPS-BET somente as classes *Viacao-Num-Cartoes* e *CombinacaoRestrita* são pontos de variação não envolvidos em herança. Todas as classes que são ponto de variação não possuem métodos abstratos, o que resulta na extensibilidade igual a zero. O objetivo é maximizar a extensibilidade, mas como o algoritmo é de minimização, o valor de extensibilidade está sendo invertido. Para isso o algoritmo retorna um valor alto (1000,0) quando a extensibilidade é zero, para indicar que o resultado é ruim.

Modularização das Características das Soluções Encontradas

Com relação à modularização de características medida pela função FM, os valores são melhores na maioria das soluções obtidas independentemente da configuração do modelo de avaliação. Isso mostra que de fato o operador dirigido à características cumpre o seu papel. Somente duas das vinte e nove soluções da LPS-BET têm valor de FM pior que o original. No caso da AGM-4, por exemplo, as soluções encontradas têm um maior número de características modularizadas do que a PLA original. A solução de melhor modularização de características encontrada pelo NSGAII-M-FC alcançou 48% de melhoria em FM, enquanto as soluções de NSGAII-M-FCE e NSGAII-M-FE melhoraram 47%. Nessa última solução as características estão mais entrelaçadas em nível de componente do que na solução de NSGAII-M-FC (métrica CIBC). Na solução de NSGAII-M-FCE as características estão mais entrelaçadas em nível de interface (métrica IIBC).

No caso da MM-3, NSGAII-M-FC teve 20% de melhoria em FM, NSGAII-M-FCE teve 18% e NSGAII-M-FE alcançou 16% de melhoria. Na solução de NSGAII-M-FC, as características se entrelaçam menos nos níveis de componente e de interface do que nas

outras duas soluções. As referidas soluções encontradas por NSGAII-M-FC e NSGAII-M-FE são apresentadas nas Figuras 5.8 e 5.9, respectivamente. Em ambas figuras vários detalhes foram omitidos, tais como, atributos e métodos de classes, operações de interfaces e detalhes da especificação das variabilidades. Os elementos arquiteturais estão associados a características da LPS. Para melhorar a legibilidade do projeto, fica subentendido que as classes estão associadas às mesmas características de seus componentes, assim como as operações em relação às suas interfaces. Operações de algumas interfaces são mostradas para salientar algumas diferenças de projeto entre as duas soluções.

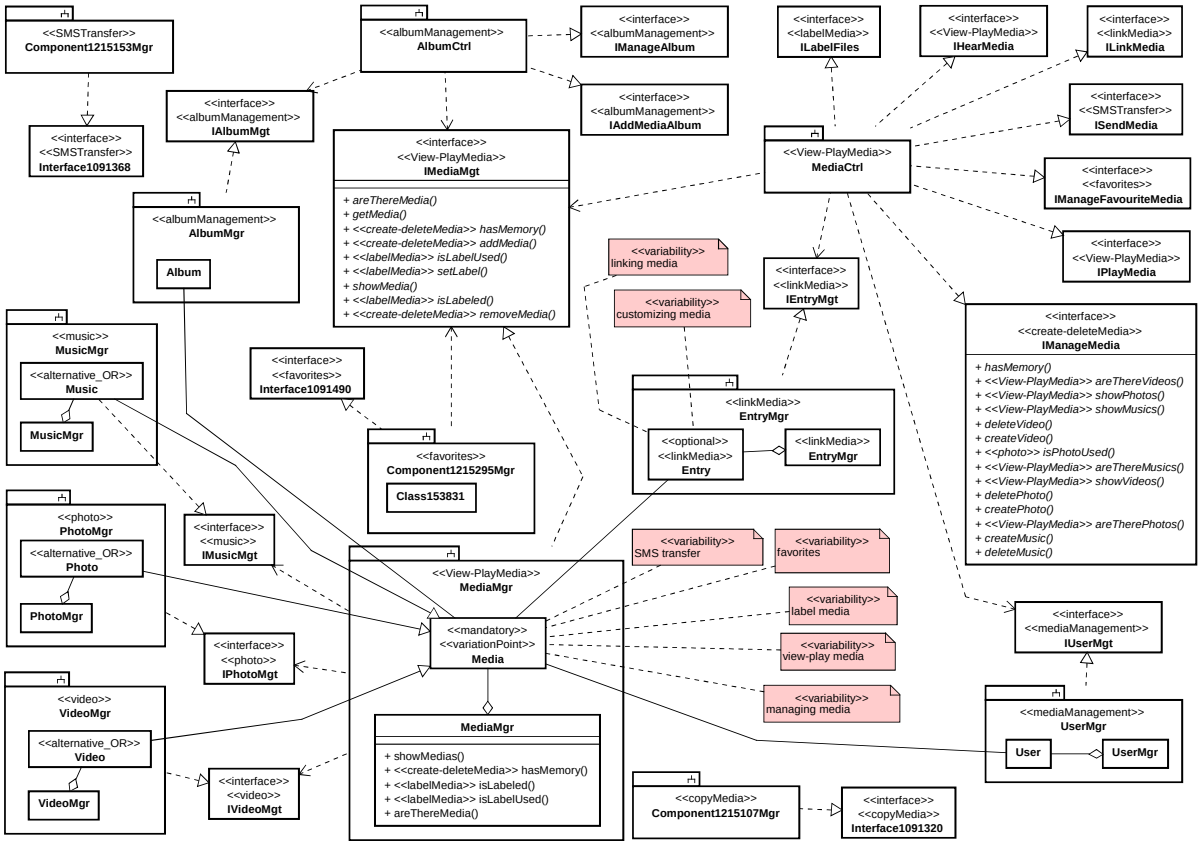


Figura 5.8: Estudo 4 - Solução obtida pelo NSGAII-M-FE para MM-3

Na solução de NSGAII-M-FE percebe-se que a interface $IManageMedia$ está associada à característica *create-deleteMedia* e contém operações associadas com *View-PlayMedia*. Além disso, o componente $MediaCtrl$ (também associado com *View-PlayMedia*) implementa interfaces associadas a diferentes características. Na solução de NSGAII-M-FC, as operações associadas com *View-PlayMedia* que estavam em $IManageMedia$

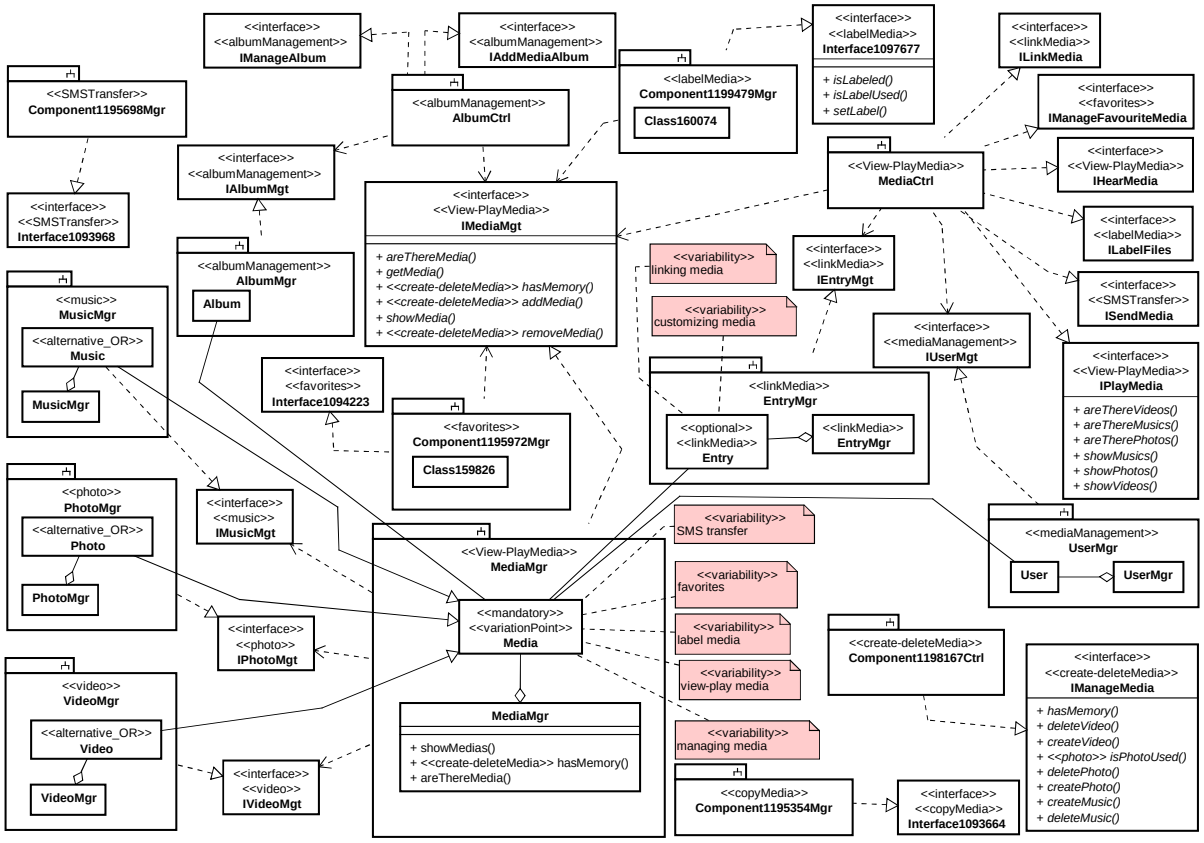


Figura 5.9: Estudo 4 - Solução obtida pelo NSGAII-M-FC para MM-3

na primeira solução fazem parte da interface $IPlayMedia$, que já continha operações associadas a essa característica. A interface $IManageMedia$ é realizada por um componente criado especificamente para modularizar o interesse *create-deleteMedia*. Esse mesmo tipo de mutação foi aplicado para modularizar a característica *labelMedia* em $Component1199476Mgr$ que implementa $Interface1097677$ onde estão aquelas operações associadas a *labelMedia* que aparecem na interface $IMediaMgt$ da primeira solução. Além disso, a classe $MediaMgr$ não contém métodos associados a *labelMedia*. Portanto, a solução de NSGAII-M-FC tem menor entrelaçamento de características do que a solução de NSGAII-M-FE. Entretanto, a segunda solução tem três componentes desconexos ($Component1198167Ctrl$, $Component1195354Mgr$ e $Component1195698Mgr$) enquanto a primeira solução possui somente dois componentes desconexos. Algumas melhorias podem ser realizadas na implementação da OPLA-Tool visando a evitar esse tipo de situação. No entanto, para que o arquiteto utilize soluções como essas duas, bastaria conectar esses componentes aos componentes que contêm os dados de que necessitam.

Elegância das Soluções Encontradas

Ao analisar a elegância das soluções encontradas pelos três experimentos, percebe-se que na maioria das vezes os projetos de PLA encontrados são mais elegantes. Somente uma solução tem pior elegância que o projeto original e algumas poucas mantiveram o valor inicial dessa métrica. A maioria das soluções encontradas por NSGAII-M-FCE e NSGAII-M-FC são mais elegantes que as encontradas por NSGAII-M-FE. Isso pode ser ilustrado para a PLA MM-3, onde a solução encontrada pelo NSGAII-M-FCE alcançou a melhor elegância: 0,1620 (11,2% melhor que a original) seguida pelo NSGAII-M-FC: 0,172 (5,7% de melhoria). O projeto obtido por NSGAII-M-FCE tem menor desvio padrão de acoplamento externo das classes do projeto e menor desvio padrão entre os números de atributos e de métodos entre as classes do que nas outras soluções.

As soluções de melhor elegância encontradas para a AGM-4 também ilustram isso. Em termos de elegância todas são muito parecidas, porém mais elegantes que a PLA original. NSGAII-M-FCE alcançou a solução com a melhor elegância (6% de melhoria). As três soluções têm o mesmo número de classes. No entanto, a principal diferença entre as três soluções é o desvio padrão entre os números de atributos e de métodos das classes, que é menor na solução encontrada por NSGAII-M-FCE que nas soluções alcançadas pelos outros dois experimentos.

Finalizando, dentre os experimentos que utilizam somente os operadores de mutação, NSGAII-M-FC alcançou o melhor valor de FM e de *Eleg* em 4 PLAs, NSGAII-M-FCE alcançou o melhor valor de elegância em 3 PLAs e NSGAII-M-FE alcançou o melhor valor de FM em 3 PLAs. Por outro lado, NSGAII-M-FE encontrou as soluções com pior modularização de características e pior elegância em 4 casos.

5.5.2.2 Otimização com mutação e cruzamento

Nesta seção os resultados obtidos pelos experimentos que usam os operadores de mutação e o operador de cruzamento Feature-driven Crossover (NSGAII-MC-FE, NSGAII-MC-FC e NSGAII-MC-FCE) são analisados. Nesse grupo de experimentos foram descartadas as soluções consideradas inválidas, conforme descrito na Seção 5.5.1. O número de soluções

descartadas foi insignificante. Mesmo para LPS-BET, a LPS que teve o maior número de soluções descartadas, o percentual é menor que 1%.

Além dos dados quantitativos, uma análise qualitativa foi realizada para as soluções com melhor elegância, melhor MSI e com diferentes valores de extensibilidade das PLAs AGM-1, AGM-4, MM-1, MM-3 e MM-4. Foram analisados os mesmos aspectos considerados nos experimentos que empregaram somente mutação.

Aproximadamente 50% das soluções obtidas pelos experimentos utilizando o operador de cruzamento proposto são soluções incompletas ou inconsistentes. Soluções encontradas para AGM-2 e AGM-4 são incompletas porque atributos e métodos da classe *Player* associados com as características *ranking* e/ou *logging* não estão presentes na PLA. Nas soluções incompletas encontradas para MM-1, MM-2, MM-3 e MM-4, algumas ou todas as subclasses de $Media$ foram perdidas durante o processo de cruzamento. Atualmente não existe qualquer restrição que verifique casos como esses, por isso, as soluções incompletas sobreviveram ao longo das gerações.

A conclusão que se pode alcançar para justificar a obtenção de soluções incompletas é que quando o operador de cruzamento é aplicado, a depender da característica selecionada para efetuar o cruzamento, pode ser que alguns elementos como atributos e métodos associados com características que estavam entrelaçadas em uma mesma classe, acabem se perdendo após o cruzamento. A Figura 5.10 mostra um exemplo parcial que pode contribuir para facilitar o entendimento do problema. Supondo que: (i) $Parent1$ contenha a classe *Player* cujos atributos e métodos estão associados a duas diferentes características *ranking* e *logging*, (ii) $Parent2$ tenha sofrido uma mutação na qual a partir da classe *Player* de $Parent1$ a característica *logging* tenha sido modularizada em $Class1$ e, (iii) *ranking* seja a característica selecionada, quando do cruzamento de $Parent1$ e $Parent2$, $Child1$ será uma solução incompleta, pois: (a) primeiramente a classe *Player* que $Child1$ herdou de $Parent1$ será excluída, já que está associada com *ranking* e, (b) depois a classe *Player* de $Parent2$ será adicionada a $Child1$. Ocorre que os métodos associados com *logging* que estavam em *Player* de $Parent1$ foram excluídos e não foram adicionados em nenhum outro elemento de $Child1$, deixando a solução incompleta.

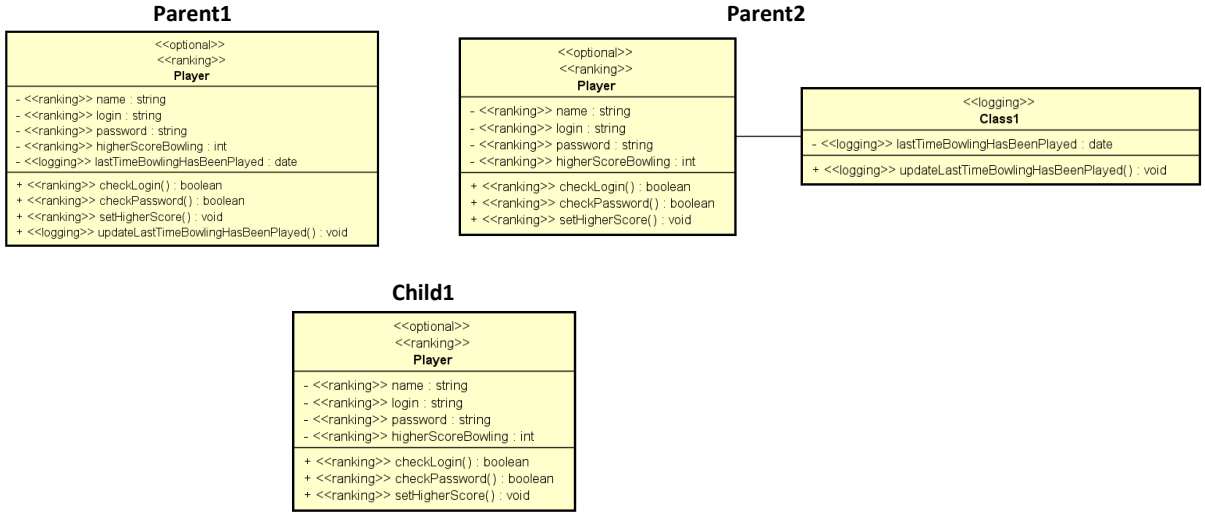


Figura 5.10: Exemplo parcial de cruzamento que gera uma solução incompleta

Extensibilidade das Soluções Encontradas

Nas soluções completas, o valor da métrica de extensibilidade não foi melhorado porque os operadores não alteram a relação de métodos abstratos por métodos concretos nas classes envolvidas em relacionamentos de herança, conforme explicação dada na análise dos experimentos que só usam mutação.

De todas as soluções de MM-1, somente uma delas melhora o valor de extensibilidade (encontrada por NSGAII-MC-FCE), porém se trata de uma solução incompleta. Todas as soluções alcançadas por NSGAII-MC-FCE para MM-3 tiveram valores de extensibilidade diferentes do original mas são incompletas. Algumas ou todas as subclasses de *Media* foram perdidas. Para MM-4, NSGAII-MC-FCE alcançou 6 soluções com menor valor de extensibilidade do que o original, as quais também são incompletas ou inconsistentes.

Modularização de Características das Soluções Encontradas

Considerando somente as soluções completas, todas elas têm características melhor modularizadas do que suas respectivas PLAs originais. De forma geral, NSGAII-MC-FCE e NSGAII-MC-FC alcançaram soluções com melhor modularização de características do que NSGAII-MC-FE. Na MM-1, por exemplo, a solução de NSGAII-MC-FCE tem melhor coesão baseada em características e menor difusão de características entre os elementos arquiteturais, porém as soluções de NSGAII-MC-FE e NSGAII-MC-FC têm menor interação entre as características nos níveis de interface e de componente. NSGAII-

MC-FCE tem o menor valor total de FM: 41% de melhoria, enquanto NSGAII-MC-FC melhorou em 36% e NSGAII-MC-FE melhorou em 33% o valor dessa função.

Elegância das Soluções Encontradas

Na maioria dos casos, as soluções encontradas têm melhor valor para a função referente às métricas de elegância do que as PLAs originais. As alternativas de projeto de PLA obtidas para AGM-1 que possuem a melhor elegância são mais elegantes que a PLA original. NSGAII-MC-FCE alcançou a solução com a melhor elegância (13% de melhoria). As soluções de NSGAII-MC-FCE e de NSGAII-MC-FC têm uma classe adicional enquanto a de NSGAII-MC-FE tem duas classes adicionais em relação à PLA original. No entanto, a maior diferença entre elas é que a solução encontrada por NSGAII-MC-FCE tem o menor desvio padrão entre os números de atributos e de métodos entre as classes da PLA. Isso se deve à presença das duas novas classes que fazem parte de novos componentes que modularizam características. Ao que tudo indica, essas classes foram criadas pelo operador Feature-driven Operator, as quais contêm métodos associados às características modularizadas por seus respectivos componentes. Dessa forma, além de beneficiar a modularização de características, a elegância do projeto foi melhorada.

Diante do exposto, observa-se que os experimentos NSGAII-MC-FC e NSGAII-MC-FCE alcançaram soluções com melhor modularização de características em 5 e 3 PLAs, respectivamente. NSGAII-MC-FCE também gerou as soluções mais elegantes para 4 PLAs. Por outro lado, NSGAII-MC-FE teve o pior desempenho desse grupo de experimentos, encontrando soluções de pior modularização de características em 6 PLAs e de pior elegância em 7 PLAs.

De forma geral, é possível concluir que as soluções alcançadas por NSGAII-MC-FC tendem a ser mais completas do que as soluções de NSGAII-MC-FCE porque ao incluir a extensibilidade como um objetivo, a mudança no seu valor permite que soluções incompletas sobrevivam entre as gerações.

5.5.2.3 Respondendo as questões de pesquisa

As PLAs utilizadas nos experimentos não permitiram avaliar se a presença da função $Ext(pla)$ no modelo de avaliação contribui para alcançar projetos de PLA mais extensíveis (**RQ1**) porque, conforme explicado anteriormente, classes que são os pontos de variação e variantes participam, em sua maioria, de relacionamentos de generalização. E os operadores de mutação não alteram a organização dos atributos e métodos de relacionamentos deste tipo, impedindo a alteração do valor da extensibilidade. Sendo assim, estudos com PLAs com diferentes estruturas precisam ser realizados, a fim de averiguar este tipo de situação. Apesar disso, a métrica se mostrou sensível a projetos menos extensíveis, que foram percebidos quando da geração de soluções incompletas ou inconsistentes, pois na análise qualitativa percebeu-se que de fato esses projetos não eram tão extensíveis quanto suas versões originais. Desse modo, pode-se afirmar que a métrica de extensibilidade de LPS é adequada para avaliar essa propriedade nas soluções geradas pela MOA4PLA.

Com relação à **RQ2**, no contexto dos experimentos realizados, a função $Ext(pla)$ não comprometeu a obtenção de projetos de PLA com melhor modularização de características do que as PLAs originais. Mesmo para as duas soluções de pior extensibilidade obtidas por NSGAII-M, os valores de FM foram bem melhores que o *fitness* original da PLA em questão. No entanto, os mesmos motivos que limitaram a resposta da **RQ1** também limitam a resposta da **RQ2**.

A **RQ3** refere-se ao grau de modularização das características das soluções obtidas nesse estudo empírico, em relação ao segundo estudo relatado na Seção 5.3. Quando comparadas às PLAs alcançadas no segundo estudo empírico, as PLAs alcançadas neste estudo tem um maior número de características modularizadas em componentes. Tomando a AGM-2 como ilustração, no segundo estudo havia no máximo 5 componentes para modularizar exclusivamente uma característica. No presente estudo, todos os projetos de PLAs analisados para AGM-2 têm exatamente 7 componentes que modularizam uma única característica. Isso mostra que a alteração na probabilidade de aplicação do Feature-driven Operator foi benéfica e propiciou encontrar projetos mais modulares em termos de características de LPS. Ainda assim, uma análise mais minuciosa, considerando

todos os aspectos envolvidos na avaliação de *fitness* das soluções, pode ser realizada para mensurar os pontos positivos e negativos de tal alteração.

A **RQ4** supõe a derivação de diretrizes indicando quais peculiaridades de projetos de PLAs podem ser esperadas pelo uso de cada uma das configurações do modelo de avaliação que foram experimentados. As seguintes diretrizes puderam ser identificadas:

- **Diretriz 1:** Para encontrar soluções com melhor modularização de características, independentemente do uso do operador de cruzamento, a configuração que emprega as funções FM e CM é mais adequada e suficiente.

- **Diretriz 2:** A configuração envolvendo três objetivos - FM, CM e Ext - propicia a geração de soluções de maior elegância, ainda que essa métrica não seja considerada como um objetivo.

- **Diretriz 3:** Quando o processo evolutivo envolve somente a aplicação dos operadores de mutação, a configuração usando as funções FM e Ext tem o pior desempenho em termos de modularização de características e elegância de projeto.

- **Diretriz 4:** Para garantir os princípios básicos de projeto e a diversidade do conjunto de soluções alcançadas pela MOAPLA, a presença da função CM no modelo de avaliação é imprescindível.

- **Diretriz 5:** As métricas dirigidas a características envolvidas na função FM além de beneficiar a modularização de características, colaboram para obter projetos mais elegantes e mais estáveis sob o ponto de vista das métricas usadas na função CM.

Nenhuma diretriz a respeito do uso da métrica de extensibilidade pode ser apontada devido aos projetos de PLA utilizados nos experimentos. Novos estudos envolvendo PLAs de diferentes tamanhos e com diferentes peculiaridades ajudariam a melhor caracterizar os benefícios alcançados pelas possíveis configurações de funções para o modelo de avaliação.

A **RQ5** refere-se à possibilidade de obtenção de boas soluções em termos de modularização de características, estabilidade do projeto e extensibilidade de LPS quando se utiliza MOA4PLA. Quando comparados às PLAs originais, os resultados evidenciam que

os experimentos que aplicam somente os operadores de mutação melhoram a modularização de características sem prejudicar a estabilidade e a extensibilidade do projeto de PLA. Inclusive, dentre os resultados obtidos, a estabilidade medida pela função CM foi maior em 100% das soluções e a elegância do projeto melhorou em 96% dos casos (144 de 150 soluções).

Com relação à extensibilidade, dadas as peculiaridades das PLAs utilizadas nos experimentos, não foi possível perceber melhoria significativa na função Ext. Porém, nas soluções alcançadas por NSGAII-M-FC e NSGAII-M-FCE para MM-3, a classe $Media$ (que é ponto de variação) pertencente ao componente $MediaMgr$ e a interface $IMediaMgt$ implementada por este componente estão associadas a um menor número de características do que no projeto original. Dessa forma, apesar de essa alteração não influenciar o valor de Ext, o projeto da PLA ficou mais facilmente reutilizável devido a presença de um menor número de características associadas ao componente em questão. Ainda assim, seria preciso realizar novos estudos envolvendo outras PLAs. Os experimentos envolvendo os operadores de mutação e de cruzamento conseguem atingir os mesmos tipos de resultados dos experimentos que aplicam somente mutação considerando-se somente as soluções completas. No entanto, têm a desvantagem de gerar soluções incompletas como mencionado anteriormente.

5.5.2.4 Lições Aprendidas

Algumas lições puderam ser aprendidas a partir da realização do presente estudo, as quais são discutidas nesta seção.

1. **É mais difícil garantir a completude e consistência das soluções quando o cruzamento é aplicado**. Confirmando a afirmação de Harman e Tratt [43] de que é difícil assegurar a exatidão da solução quando o operador de cruzamento é aplicado, os experimentos realizados mostraram que é mais difícil garantir a completude e consistência das soluções quando o cruzamento é aplicado no contexto da MOA4PLA. Novas restrições poderiam ser aplicadas com o intuito de garantir a consistência das soluções, mesmo assim, poderiam existir pontos descobertos. O

Feature-driven Crossover poderia ser implementado de outra forma, mas fugiria do princípio adotado quando da sua concepção. Ou ainda, algum outro operador de cruzamento poderia ser proposto. No entanto, como observado nos experimentos usando o NSGAII-M, aplicar somente operadores de mutação permite alcançar bons projetos de PLA em termos de modularização de características, sem ter problemas com a completude e a consistência das soluções encontradas. Além disso, comparando os resultados dos experimentos envolvendo NSGAII-M e NSGAII-MC, o primeiro consegue modularizar um número maior de características do que o segundo. O número de componentes que modularizam uma única característica é maior no NSGAII-M, diminuindo o entrelaçamento entre as características.

2. **As métricas LCC, CIBC e CDAC poderiam ser utilizadas para medir a modularização de características em nível de classes.** Apesar de não terem sido incluídas na função objetivo FM, as métricas LCC, CIBC e CDAC foram computadas para as classes. Uma análise dos seus valores mostra que essas métricas poderiam ser incluídas em FM para considerar a modularização de características também em nível de classes. É possível que a sua inclusão viabilizasse a obtenção de soluções de projeto contendo classes melhor modularizadas em termos de características.

3. **Alguma função objetivo deveria incluir a elegância do projeto dos componentes.** A análise da elegância das soluções obtidas permitiu perceber que as métricas de elegância conseguem identificar bem o melhor balanceamento entre atributos e métodos nas classes. No entanto, seria interessante considerar também o balanceamento entre os componentes da PLA. A maioria das soluções geradas tem um número de componentes maior que a original o que leva a componentes menos sobrecarregados. Ainda assim, há componentes com tamanhos muito diferentes, alguns muito grandes e outros extremamente pequenos. Porém, as métricas empregadas não consideram esse aspecto. Nesse sentido, as métricas de elegância de projeto poderiam ser adaptadas para o nível de componentes arquiteturais.

4. **É preciso melhorar a estratégia de detecção de componentes desconectados na MOA4PLA.** Grande parte das soluções encontradas possui componentes desconectados do restante do projeto. No entanto, não foi possível perceber uma relação direta entre o número de componentes desconectados e a aplicação do operador de cruzamento ou a configuração do modelo de avaliação. Além disso, nenhuma das métricas utilizadas consegue indicar a presença desses componentes na PLA. É preciso investigar a causa desse problema. Pode ser acrescentada uma restrição para cuidar disso ou ainda uma métrica para dar indicadores a esse respeito. Uma possível solução seria adaptar ou propor alguma métrica de elegância para componentes ou usar métricas como FanIn e FanOut.

5. **O Feature-driven Operator pode ser alterado para melhor modularizar características de interfaces.** A modularização de características de certas interfaces como $IMediaMgt$ não melhora muito porque a implementação do Feature-driven Operator não percorre as interfaces de componentes que só tratam de uma única característica. $IMediaMgt$ é um exemplo de interface que atende a várias características, mas que está em um componente associado a somente uma característica. Dessa forma, essa interface não sofre a ação do referido operador.

5.6 Ameaças à Validade dos Estudos Empíricos

Esta seção contém as principais ameaças à validade dos estudos empíricos conduzidos e relatados neste capítulo. A principal questão que se deve levar em conta - como um risco de afetar a validade de conclusão dos estudos - é o tamanho das LPSs utilizados na amostra.

Os projetos de PLA podem ser considerados pequenos se comparados com o *kernel* do Linux que tem 6320 características [9]. No entanto, eles permitem evidenciar que os objetivos da MOA4PLA podem ser alcançados até mesmo para pequenas LPSs. Assim, com LPS maiores, os resultados poderiam ser observados também. Ademais, essas LPSs permitiram a avaliação dos desempenhos dos operadores de busca propostos em

diferentes contextos de modularização de características. Apesar disso, a amostra pode ser aumentada durante futuras repetições desses estudos empíricos.

A validade de constructo com relação às métricas utilizadas é garantida pela sua aplicação bem sucedida em outros estudos anteriores, tais como [23, 33, 83]. O mapeamento de características para os elementos arquiteturais influencia a modularização de características das soluções obtidas. Nos estudos realizados, adotou-se o mapeamento realizado pelos projetistas originais de cada PLA. Diferentes mapeamentos podem levar a diferentes resultados porque em PLAs com características mais modularizadas os benefícios da MOA4PLA e dos operadores de busca propostos para melhorar a modularização de características serão observados em menor proporção do que em PLAs cujas características estejam menos modularizadas. Ainda assim, os benefícios da abordagem e de seus operadores devem ser notados. Um outro ponto é que a AGM, a MM e a LPS-BET são LPS não-comerciais. Apesar disso, essas LPSs permitiram observar o comportamento dos operadores de busca e as vantagens e desvantagens da MOA4PLA. Essas observações podem ser analisadas em estudos futuros, com objetivos semelhantes, envolvendo LPS comerciais.

Finalmente, o mesmo tamanho de população e número de gerações foram adotados nos experimentos independentemente do tamanho da PLA. Valores diferentes para esses parâmetros poderiam implicar em uma convergência diferente, possivelmente melhor, do NSGAII-MC para a maior PLA. Esse impacto, assim como o impacto de todos os riscos, podem ser analisados em estudos futuros.

5.7 Selecionando uma alternativa de projeto de PLA

O objetivo desta seção é discutir como um arquiteto pode selecionar, a partir do conjunto de possíveis soluções retornado pela MOA4PLA, uma solução para adotar como PLA para uma determinada LPS (atividade Transformação e Seleção (Seção 4.2.1)). Essa escolha pode ser baseada em uma métrica, utilizada como função objetivo, para ser priorizada de acordo com as necessidades da LPS ou metas organizacionais. Ou o arquiteto poderia escolher a solução com a menor ED porque esta solução tem o melhor *trade-off* entre os

objetivos usados. O módulo OPLA-GUI da OPLA-Tool deve dar suporte ao arquiteto para identificar essas soluções dentro do conjunto obtido de soluções.

Considerando que após o processo de otimização as soluções geradas pelo algoritmo de busca fossem as apresentadas nas Figuras 5.8 e 5.9 e, que as métricas CM, $Eleg$ e FM fossem utilizadas para avaliar essas soluções, essas duas soluções seriam avaliadas pelos valores apresentados na Tabela 5.12. Os três objetivos são de minimização. Na tabela os melhores valores estão destacados em negrito.

Tabela 5.12: Seleção de soluções arquiteturais

	Solução Arquitetural	CM	$Eleg$	FM	**CIBC**	**LCC**
a	Figura 5.9	32.9409	**0.1723**	**347.0**	**77.0**	28.0
b	Figura 5.8	**32.21637**	0.1766	362.0	99.0	**26.0**

As duas soluções são não dominadas, porque enquanto a solução b tem o menor valor para CM, a solução b é melhor nos outros dois objetivos. O arquiteto teria que selecionar uma delas para adotar como PLA da LPS em desenvolvimento. Caso estivesse interessado em adotar uma PLA com melhor modularização de características, ele deveria escolher a solução a, por outro lado, a segunda solução está melhor projetada em termos dos princípios básicos de projeto (CM). Se o arquiteto desejasse priorizar a elegância do projeto, poderia escolher a solução a apesar dos valores de elegância das duas soluções serem bastante parecidos. Desse modo, é possível perceber como as prioridades do arquiteto podem influenciar na seleção do projeto de PLA que será adotado para a LPS.

Essa avaliação também é muito dependente das métricas utilizadas. Em vez de usar FM, as métricas dirigidas a características CIBC e LCC poderiam ser utilizadas como objetivos separados. Os valores médios das métricas CIBC e LCC são mostrados nas últimas colunas da tabela. Nesse caso, a solução a tem menor entrelaçamento entre as características, enquanto a solução b tem menor falta de coesão baseada em características. Situações como essas influenciam diretamente na seleção da solução.

Outra observação é com relação às mudanças realizadas no projeto tendo em vista o projeto original. O arquiteto geralmente conhece o projeto fornecido como entrada, então se ele desejar analisar as mudanças antes de tomar sua decisão, é possível verificar: (i) como as características são modularizadas em componentes, observando as características

associadas com elementos arquiteturais, e (ii) quais são os novos componentes, interfaces e classes que foram adicionados. A identificação dos novos elementos é facilitada porque eles têm nomes genéricos, tais como $Component103$ ou $Interface94$. A propósito, depois de escolher um projeto de PLA, o arquiteto precisa renomear os novos elementos arquiteturais para melhorar o significado dos nomes desses elementos.

Retomando o primeiro estudo empírico no qual GAs foram aplicados para otimizar o projeto de PLA, poder-se-ia argumentar que ao utilizar GAs, o arquiteto não precisaria escolher uma solução para adotar como projeto de PLA porque GAs retornam uma única solução. Certamente isso facilitaria o trabalho do arquiteto, mas também tem desvantagens porque o arquiteto precisaria estabelecer o peso de cada objetivo a ser utilizado no processo evolutivo além de perder a diversidade de soluções geradas pelos algoritmos multiobjetivos contendo os melhores *trade-offs* entre os objetivos adotados. Grande parte dos experimentos usando MOEAs retornou mais de uma solução. Dessa forma, apesar do custo adicional para o arquiteto selecionar um projeto, o uso de MOEAs permite obter soluções mais variadas e possivelmente melhores.

5.8 Considerações Finais

Neste capítulo foram apresentados quatros estudos empíricos conduzidos para avaliar o uso da MOA4PLA, envolvendo diferentes algoritmos de busca, diferentes combinações de operadores evolutivos e diferentes combinações de métricas para compor o modelo de avaliação. Ainda foram apresentadas as PLAs utilizadas nos estudos realizados e as ameaças à validade desses estudos. A última seção tratou de como um arquiteto pode selecionar uma das soluções retornadas pela abordagem proposta, mostrando alternativas e abordando algumas questões práticas envolvidas.

Os resultados dos estudos mostram que soluções de projeto de PLA mais elegantes, com maior modularização de características e maior estabilidade podem ser obtidas pela MOA4PLA. Ainda assim, há melhorias que precisam ser feitas tanto no operador de cruzamento como nas restrições aplicadas a cada solução gerada. A Tabela 5.13 contém as principais conclusões alcançadas em cada um dos estudos.

Tabela 5.13: Síntese dos Resultados dos Estudos Empíricos

Estudo	**Principais Conclusões**
1	O tratamento multiobjetivo é adequado para projeto de PLA.
	NSGAII têm melhor desempenho que GAs no contexto da MOA4PLA.
	O Feature-driven Operator permite alcançar soluções com características mais modularizadas.
	A função FM é sensível à modularização de características.
2	NSGAII consegue alcançar bons resultados mesmo utilizando somente operadores de mutação.
	Feature-driven Crossover permite que NSGAII alcance resultados ainda melhores.
	Feature-driven Crossover pode gerar algumas soluções inconsistentes.
3	NSGAII obtém melhores resultados do que PAES no contexto de projeto de PLA.
4	Maior probabilidade de aplicação do Feature-driven Operator propicia alcançar melhores resultados.
	As funções FM e CM permitem encontrar soluções de melhor modularização de características do que as outras duas combinações de funções usadas.
	A presença da função CM no modelo de avaliação garante os princípios básicos de projeto e a diversidade de soluções.
	Os resultados mostram que os experimentos conseguem encontrar alternativas de projeto de PLA consideradas boas com relação aos objetivos da MOA4PLA.
	Feature-driven Crossover gerou muitas soluções incompletas e/ou inconsistentes.

Atualmente, a única forma de perceber que uma solução é incompleta é comparando o projeto obtido e o projeto original. No entanto, considerando que os projetos de PLAs reais costumam ser grandes, não seria confiável esperar que o arquiteto validasse a completude do projeto, além de ser trabalhoso. Assim, sugere-se que somente os operadores de mutação sejam aplicados para evitar a geração de soluções incompletas e inconsistentes.

Uma série de outras direções de pesquisa também podem ser estabelecidas a partir desta tese. Essas direções são discutidas no próximo capítulo, juntamente com as conclusões.

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

O desenvolvimento e a avaliação do projeto de PLA é uma tarefa primordial na engenharia de LPS, a fim de garantir o reúso dos componentes da PLA e permitir a evolução da LPS. No entanto, trata-se de uma tarefa difícil na qual há vários fatores envolvidos. Métricas arquiteturais podem auxiliar os arquitetos mas, ainda assim, o volume de dados a serem analisados e a análise do *trade-off* entre as métricas e demais fatores demanda grande esforço desses profissionais. O uso de técnicas de SBSE pode auxiliar os arquitetos a avaliar e melhorar um dado projeto de PLA automaticamente. Apesar de recente, a SBSE tem beneficiado várias atividades de Engenharia de Software. Entretanto, o projeto de PLA ainda não havia sido explorado nos trabalhos relacionados.

Nesse sentido, este trabalho investigou a aplicação de algoritmos de busca multiobjetivos para melhorar o espaço de soluções de PLAs apoiando os arquitetos de LPS durante as atividades de projeto e avaliação de PLA. Para isso, foi proposta a MOA4PLA, uma abordagem que inclui um processo para otimizar o projeto de PLA utilizando algoritmos multiobjetivos, uma representação do problema que permite a manipulação da PLA pelos algoritmos de busca, operadores de busca para otimizar o projeto, e aplica métricas convencionais e específicas para LPS durante o processo de otimização. Os resultados alcançados mostram que os algoritmos de busca multiobjetivos são capazes de explorar PLAs alternativas e oferecer indicadores relacionados a modularização de características e princípios básicos de projeto, diminuindo o esforço dos arquitetos. Os resultados mostram ainda que os projetos de PLA obtidos estão menos sujeitos à ocorrência do *bad smell Scattered Functionality* do que as PLAs originais utilizadas nos estudos empíricos.

Mesmo para LPSs de pequeno porte, a busca por diferentes alternativas de projetos de PLA mais modulares, mais reusáveis e/ou mais extensíveis é uma tarefa difícil. Em se tratando de LPS comerciais, com centenas ou milhares de características, tal feito se

torna praticamente impossível. Nesse sentido, é imprescindível dispor de uma abordagem automatizada. Por isso, a MOA4PLA foi proposta para identificar automaticamente as melhores alternativas de projeto para uma PLA de forma a atender ao objetivo geral deste trabalho. E os resultados empíricos confirmam a hipótese de que a otimização multiobjetivo pode contribuir para melhorar o projeto de PLA.

O conjunto de contribuições que confirma a referida hipótese é revisitado na próxima seção. Depois, na Seção 6.2 as limitações do trabalho discutidas ao longo desta tese são sumarizadas e direções de pesquisa para resolvê-las, bem como outros possíveis trabalhos futuros são apresentados.

6.1 Contribuições da Tese

As contribuições desta tese podem ser categorizadas em três grupos: (i) definições teóricas, (ii) implementação de apoio automatizado e, (iii) estudos de avaliação. Cada uma dessas categorias é apresentada a seguir.

6.1.1 Definições Teóricas

As contribuições teóricas envolvem todas as definições contidas na MOA4PLA, as quais incluem:

Processo da MOA4PLA: um processo composto por quatro atividades e seus artefatos de entrada e saída foram definidos para viabilizar a avaliação e melhoria de projetos de PLA por meio de algoritmos de busca multiobjetivos. O processo proposto é genérico o suficiente para que diferentes algoritmos de busca, métricas arquiteturais e operadores de busca sejam aplicados no processo de otimização, de acordo com os interesses do arquiteto. Além disso, o processo não está restrito a um determinado método de projeto de PLA, método de associação de elementos arquiteturais com as características da LPS ou método para representação das variabilidades nos artefatos da LPS.

Definição de uma representação do problema de projeto de PLA: uma representação direta de projeto de PLA baseada em um metamodelo foi definida para a otimização no contexto da MOA4PLA. A proposta dessa representação foi necessária

porque as representações adotadas pelos trabalhos relacionados não são adequadas para o projeto de PLA.

Proposta de funções objetivo para compor o Modelo de Avaliação: um conjunto de métricas convencionais e específicas para LPS foi selecionado para compor o Modelo de Avaliação. Essas métricas atendem aos principais objetivos da MOA4PLA. A partir dessas métricas, foram propostas quatro funções objetivo que foram incluídas na OPLA-Tool e que podem ser utilizadas pelo arquiteto durante o processo de otimização.

Adaptação de operadores de busca convencionais: operadores de busca - utilizados nos trabalhos envolvendo projeto de software baseado em busca - foram adaptados para o contexto de projeto de PLA para que pudessem ser aplicados no projeto de PLA baseado em busca viabilizado pela MOA4PLA.

Proposta de operador de busca dirigido a características: o Feature-driven Operator é um operador de busca que foi proposto visando à modularização de características de LPS em um projeto de PLA durante o processo de otimização realizado por um algoritmo de busca.

Proposta de operador de cruzamento dirigido a características: o Feature-driven Crossover foi proposto para permitir o cruzamento de dois projetos de PLA durante a aplicação de algoritmos evolutivos multiobjetivos na instanciação da MOA4PLA.

6.1.2 Projeto e Implementação do Apoio Automatizado

Concepção da OPLA-Tool e implementação do OPLA-Core: Visando à aplicação automatizada da MOA4PLA, foi concebida a ferramenta OPLA-Tool. Essa ferramenta inclui módulos para permitir: (i) a codificação e decodificação de um projeto de PLA em um formato manipulável pelos algoritmos de busca (OPLA-Encoding e OPLA-Decoding), (ii) a utilização de operadores de busca para aplicar padrões de projeto e estilos arquiteturais nos projetos de PLA (OPLA-Patterns e OPLA-ArchStyles), (iii) a comunicação com o usuário da ferramenta (OPLA-GUI) e, (iv) a otimização de fato do projeto de PLA por meio dos algoritmos de busca (OPLA-Core). O módulo OPLA-Core foi implementado no presente trabalho estendendo o *framework* jMetal. Várias classes foram criadas para

adicionar um novo problema de otimização, assim como novos operadores de busca, novos experimentos e novas métricas arquiteturais.

Instanciação da MOA4PLA: justamente por ser genérica, a abordagem proposta precisa ser instanciada para algum tipo de algoritmo de busca. Nesse sentido, neste trabalho, a MOA4PLA foi instanciada utilizando algoritmos evolutivos multiobjetivos, uma vez que este tipo de algoritmo tem se mostrado adequado nas abordagens desenvolvidas nos trabalhos relacionados. Além de uma versão utilizando GAs, foram implementados outros dois MOEAs que utilizam diferentes estratégias de evolução e diversificação: o NSGAII e o PAES.

Aplicação das métricas utilizadas no processo evolutivo: as métricas arquiteturais incluídas na MOA4PLA não ofereciam apoio automatizado para a medição de projetos de PLA. Por isso, foi preciso implementá-las para que pudessem ser utilizadas durante o processo evolutivo realizado pelo OPLA-Core da ferrramenta OPLA-Tool.

6.1.3 Estudos de Avaliação

Quatro estudos empíricos foram realizados a fim de avaliar diferentes aspectos referentes à utilização de algoritmos de busca para a otimização de projetos de PLA.

Estudo 1 - Avaliação das principais características da MOA4PLA. No primeiro estudo empírico foram avaliadas as principais características da abordagem proposta. Os resultados permitiram concluir que: (i) o tratamento multiobjetivo oferecido pela MOA4PLA para o problema é adequado, (ii) o Feature-driven Operator, que foi implementado como operador de mutação, é eficaz para produzir projetos de PLA com melhor modularização de características, melhor estabilidade e menos propensos a apresentar o *bad smell Scattered Functionality* e, (iii) as métricas dirigidas a características de LPS e as métricas convencionais utilizadas nas funções objetivo são apropriadas para a avaliação dos projetos de PLA obtidos nos conjuntos de soluções retornados para o problema.

Estudo 2 - Avaliação do operador de cruzamento. O operador de cruzamento Feature-driven Crossover foi experimentado nesse estudo e os resultados mostraram que ele

complementa o operador de mutação Feature-driven operator, de forma a alcançar PLAs com características mais modulares. No entanto, a garantia de consistência das soluções obtidas é um ponto fraco da aplicação dos operadores de cruzamento em algoritmos evolutivos, incluindo o operador proposto.

Estudo 3 - Avaliação do algoritmo PAES. Esse estudo envolveu a utilização do MOEA PAES para resolver o problema de projeto de PLA por três motivos: (i) devido a dificuldade de manter a consistência dos projetos de PLA derivados quando o NSGAII usa cruzamento, (ii) o fato do PAES não utilizar cruzamento no processo evolutivo e, (iii) os bons resultados que o PAES tem alcançado em relação ao NSGAII em outros problemas da Engenharia de Software. Apesar dessas motivações, os resultados mostraram que o NSGAII é ainda melhor que PAES para o problema em questão, seja utilizando somente operadores de mutação, seja utilizando mutação e cruzamento.

Estudo 4 - Avaliação de diferentes funções de *fitness*. Este estudo foi motivado pela possibilidade de configurar o modelo de avaliação da MOA4PLA de diversas maneiras. Por isso, três diferentes funções de *fitness* foram experimentadas em duas versões do NSGAII, com e sem cruzamento. Os resultados do experimento mostraram que ao aumentar a probabilidade de aplicação do Feature-driven Operator, projetos mais modulares tendem a ser alcançados. Com relação ao uso da métrica de extensibilidade de LPS nas funções de *fitness*, os resultados não são conclusivos devidos as peculiaridades estruturais das PLAs utilizadas nos experimentos. Então, nesse sentido, estudos com outras PLAs precisariam ser realizados. Apesar disso, ficou evidente que a adoção da métrica de extensibilidade de LPS como um dos objetivos a serem otimizados, não impede a obtenção de projetos com melhor modularização de características.

Foram derivadas algumas diretrizes a respeito do tipo de função de *fitness* a ser adotada em relação aos resultados gerados, dentre as quais pode-se destacar que (i) o uso conjunto das funções FM e CM permite obter soluções com melhor modularização de características, (ii) que FM contribui não só para melhorar a modularização de características mas também para melhor a elegância e estabilidade do projeto e, finalmente, (iii) que CM é imprescindível para garantir os princípios básicos de projeto e a diversidade das

soluções. Os experimentos mostraram que boas alternativas de projeto de PLA podem ser obtidas pela MOA4PLA em termos de princípios básicos de projeto e modularização de características sem piorar extensibilidade da LPS. Corroborando as evidências a respeito da dificuldade de garantir a consistência das soluções geradas pela aplicação do operador de cruzamento, nesse experimento várias soluções geradas são incompletas. Como esse problema não existe quando somente operadores de mutação são aplicados no processo evolutivo, a recomendação é que MOA4PLA seja utilizada sem a adoção do operador de cruzamento proposto.

6.2 Limitações e Trabalhos Futuros

Esta seção aborda trabalhos futuros que podem ser realizados a fim de dirimir as limitações relatadas nesta tese. Na Seção 6.2.1 são apresentadas algumas novas direções de pesquisa para dar continuidade ao trabalho que foi iniciado. Os trabalhos futuros são apresentados a seguir de acordo com as limitações que os motivam:

Operador de Cruzamento: Como destacado na seção anterior, ao aplicar o operador de cruzamento Feature-driven Crossover a completude e a consistência das soluções obtidas não podem ser garantidas. Apesar disso, quando soluções completas são geradas, os resultados alcançados são ainda melhores do que quando somente a mutação é empregada no processo evolutivo. Por isso, novos estudos podem ser realizados para melhorar o desempenho desse operador, de forma que seus princípios não sejam alterados e novas restrições também possam ser especificadas tendo em vista a sua aplicação. Ainda, operadores de cruzamento propostos em trabalhos relacionados podem ser adaptados para o contexto da MOA4PLA ou algum novo operador pode ser proposto.

Elementos arquiteturais desconexos no projeto: como relatado no Capítulo 5, algumas das soluções geradas contêm componentes e interfaces desconexos do restante do projeto da PLA. É preciso investigar e definir novas restrições e/ou métricas a serem utilizadas no processo evolutivo para eliminar por completo este tipo de ocorrência.

Aplicação dos operadores de busca em hierarquias de herança. Atualmente, todos os operadores de busca estão limitados a não movimentar atributos e métodos de

classes em hierarquias de herança. É interessante estudar se a aplicação dos operadores sem essa limitação será benéfica para a otimização de projeto de PLA.

Uso de outras métricas nas funções objetivo. Como evidenciado pelos resultados dos experimentos, seria interessante empregar outras métricas nas funções objetivo, tais como, as métricas de elegância e as métricas dirigidas a características no nível de classes. Além disso, as métricas de elegância poderiam ser adaptadas para o nível de componentes.

Melhoria no Feature-driven Operator. Esse operador poderia ser melhorado de forma a buscar a modularização de característica de interfaces associadas com várias características, ainda que sejam implementadas por componentes associados com uma única característica.

Realização de novos estudos empíricos. Estudos envolvendo outras PLAs permitiriam analisar a efetividade da MOA4PLA em LPS de diferentes tamanhos e domínios, além de sua capacidade para melhorar a extensibilidade de LPS. Estudos envolvendo outros valores para os parâmetros de configuração dos algoritmos evolutivos também forneceriam mais subsídios para caracterizar os benefícios da abordagem proposta.

6.2.1 Possíveis direções de pesquisa

Além dos trabalhos listados anteriormente, várias outras direções de pesquisa podem ser seguidas a partir do trabalho apresentado nesta tese. Dentre as quais pode-se destacar:

Estudo do impacto de usar diferentes mapeamentos de características sobre os elementos arquiteturais. Será que projetos de PLA mais modulares com relação às características podem ser tão beneficiados pela MOA4PLA quanto os projetos utilizados nos experimentos realizados nesta tese?

Estudo do impacto da presença de características transversais sobre o desempenho da MOA4PLA. É possível que a presença de muitas características transversais prejudique os benefícios da MOA4PLA com relação à modularização de características assim como impacte no custo computacional para solucionar o problema.

Expansão da MOA4PLA para possibilitar a definição de diferentes prioridades de modularização de características. Esse tipo de expansão se justifica tendo

em vista que em casos reais é costumeiro possuir uma lista das características envolvidas nos produtos da LPS que serão evoluídos, as quais deveriam ter prioridade de modularização. Além disso, a modularização de características opcionais ou alternativas costuma ser mais relevante do que a modularização de características comuns a todos os produtos da LPS.

Estudo de outras formas para derivar a população inicial. A aplicação de outros métodos para derivar a população inicial pode influenciar diretamente na diversidade da população e levar a diferentes, possivelmente melhores, resultados.

Análise dos *architectural smells* presentes nas PLAs originais e nas soluções obtidas após o processo de otimização. Além do *Scattered Functionality*, outros *architectural bad smells* eventualmente presentes nas PLAs originais podem ter sido eliminados durante o processo de otimização. Da mesma forma, *architectural bad smells* não presentes nas PLAs originais podem ter sido introduzidos. Uma análise envolvendo esses dois aspectos é primordial para identificar pontos de melhoria necessários para a MOA4PLA.

Aplicação de técnicas de otimização interativa na MOA4PLA. Uma tendência em SBSE é a interação com o usuário durante o processo de otimização, dessa forma, o usuário pode interferir nas soluções durante o processo de busca a fim de melhorar os resultados produzidos. Quando o módulo OPLA-GUI estiver concluído, poderá ser expandido neste sentido. As formas de interação do usuário podem ser as mais variadas, mas vão desde limitar alterações em partes da PLA que estão satisfatoriamente bem projetadas até a inclusão de novas partes. Essas novas partes poderiam ser soluções parciais vindas de *frameworks* ou de outros componentes.

Avaliação das alternativas de projeto de PLA por parte dos projetistas originais das PLAs. No momento em que o módulo OPLA-GUI estiver disponível, os projetistas originais das PLAs podem ser consultados a respeito da qualidade das alternativas de projeto geradas pela MOA4PLA. Essas opiniões seriam úteis tanto para validar a abordagem como para melhorar suas características.

Novas funções objetivo para o modelo de avaliação. As funções objetivo disponíveis na MOA4PLA são funções que agregam várias métricas e o número máximo de objetivos que se pode utilizar atualmente está limitado a 4 (o número de funções disponíveis). Futuros estudos podem incluir outros tipos de funções objetivo e permitir o uso de um número maior de objetivos. Novas funções de agregação que atribuam diferentes pesos às métricas utilizadas podem ser acrescentadas também. Por exemplo, dar um peso maior a alguma métrica que avalie se as características opcionais da LPS estão mais modularizadas.

Estudo da correlação entre as métricas utilizadas nas funções objetivo. Estudos para averiguar a correlação entre as métricas contidas no modelo de avaliação podem ser conduzidos de forma a subsidiar a configuração das funções objetivo a serem utilizadas no processo de otimização. A correlação e o conflito entre métricas permite identificar quais métricas podem ser agrupadas em um único objetivo ou devem ser separadas em objetivos distintos.

Instanciação da MOA4PLA com outros algoritmos de busca. A abordagem proposta pode ser instanciada com diferentes algoritmos de busca. Desse modo, novas instanciações podem ser realizadas utilizando outros algoritmos de busca. Tais instanciações podem permitir a obtenção de melhores resultados, a identificação de necessidades, tais como a proposta de novos operadores de busca ou ainda, a identificação de limitações da abordagem para determinada meta-heurística.

Utilização da MOA4PLA para aplicar padrões de projeto e estilos arquiteturais. A abordagem pode ser utilizada para manter ou aplicar estilos arquiteturais nas soluções geradas durante o processo de otimização bem como para aplicar padrões de projeto visando a melhorar a estrutura do projeto contido em cada solução obtida.

Extensão da MOA4PLA para incluir modelos comportamentais. MOA4PLA pode ser estendida para receber como entrada um projeto de PLA contendo também modelos comportamentais, tais como diagramas de sequência da UML. Tal tipo de extensão permitiria outros tipos de melhoria no processo de busca, como por exemplo, a aplicação

de padrões de projeto comportamentais, o que só é possível mediante uma especificação de interação entre os objetos do projeto.

Adaptação da MOA4PLA para projeto de PLAs desenvolvidas seguindo a abordagem reativa. A abordagem proposta pode ser adaptada para receber como entrada mais de um projeto de PLA, identificar partes comuns entre os projetos fornecidos e, depois, definir e otimizar um projeto de PLA para a LPS que está em desenvolvimento.

Geração de código. Esta direção de pesquisa é mais audaciosa, mas viável a longo prazo porque após dirimidas as limitações relatadas anteriormente e dispondo de uma ferramenta, que automatize a MOA4PLA, mais evoluída, será possível trabalhar em direção à geração do código do projeto de PLA que for selecionado pelo arquiteto.

REFERÊNCIAS

[1] A. Aleti, S. Bjornander, L. Grunske, e I. Meedeniya. Archeopterix: An extendable tool for architecture optimization of AADL models. *Proceedings of the 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'09)*, páginas 61–71, Washington, DC, USA, 2009. IEEE Computer Society.

[2] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, e I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, maio de 2013.

[3] M. Amoui, S. Mirarab, S. Ansari, e C. Lucas. A genetic algorithm approach to design evolution using design pattern transformation. *International Journal of Information Technology and Intelligent Computing*, 1:235–245, 2006.

[4] S. Apel e D. Beyer. Feature cohesion in software product lines: an exploratory study. *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, páginas 421–430, New York, NY, USA, 2011. ACM.

[5] A. Arcuri e L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, páginas 21–28, New York, NY, USA, 2011. ACM.

[6] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, e A. Pozo. A multi-objective optimization approach for the integration and test order problem. *Information Sciences*, 2014. http://dx.doi.org/10.1016/j.ins.2013.12.040. In Press.

[7] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, e A. T. R. Pozo. Generating integration test orders for aspect oriented software with multi-objective algorithms. *Revista de Informática Teórica e Aplicada (RITA)*, 20(2):301–327, 2013.

[8] T. Bäck, D.B. Fogel, e Z. Michalewicz (eds.). *Evolutionary Computation 2: Advanced Algorithms and Operators*. Institute of Physics Publishing, 2000.

[9] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, e Krzysztof Czarnecki. Variability modeling in the real: A perspective from the operating systems domain. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, páginas 73–82, New York, NY, USA, 2010. ACM.

[10] M. Bowman, L. C. Briand, e Y. Labiche. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. *IEEE Transactions on Software Engineering*, 36(6):817–837, novembro de 2010.

[11] L. C. Briand, S. Morasca, e V. R. Basili. Measuring and assessing maintainability at the end of high level design. *Proceedings of the Conference on Software Maintenance*, páginas 88–97, 1993.

[12] E. K. Burke e G. Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005.

[13] S. H. Chang, H. J. La, e S. D. Kim. Key issues and metrics for evaluating product line architectures. K. Zhang, G. Spanoudakis, e G. Visaggio, editors, *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE)*, páginas 212–219, 2006.

[14] J. F. Chicano, F. Luna, A. J. Nebro, e E. Alba. Using multi-objective metaheuristics to solve the software project scheduling problem. *Proceedings of the 13th Genetic and Evolutionary Computation Conference (GECCO'2011)*, páginas 1915–1922, 2011.

[15] S. R. Chidamber e C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20:476–493, June de 1994.

[16] P. Clements, F. Bachmann, Bass L., Garlan D., J. Ivers, R. Little, P. Merson, Nord R., e J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2010.

[17] J. L. Cochrane e M. Zeleny. *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, 1973.

[18] C. A. C. Coello, G.L. Lamont, e D.A. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation. Springer, Berlin, Heidelberg, 2nd edition, 2007.

[19] T. E. Colanzi e S. R. Vergilio. Applying search based optimization to software product line architectures: Lessons learned. *Proceedings of the 4th Symposium on Search Based Software Engineering (SSBSE)*, volume 7515, páginas 259–266, 2012.

[20] T. E. Colanzi e S. R. Vergilio. Representation of software product line architectures for search-based design. *Proceedings of the 1st International Workshop on Combining Modelling and Search Based Software Engineering, International Conference on Software Engineering (ICSE'2013)*, páginas 28–33, 2013.

[21] T. E. Colanzi e S. R. Vergilio. Aspectos de implementação da OPLA-Tool. Relatório Técnico RT-DINF 001/2014, Universidade Federal do Paraná, Março de 2014. http://web.inf.ufpr.br/pos/relatorios-tecnicos.

[22] T. E. Colanzi, S. R. Vergilio, W. K. G. Assunção, e A. Pozo. Search based software engineering: Review and analysis of the field in Brazil. *Journal of Systems and Software*, 86(4):970 – 984, 2013. Special Issue: Software Engineering in Brazil: Retrospective and Prospective Views.

[23] A. C. Contieri Junior, G. G. Correia, T. E. Colanzi, I. M. de S. Gimenes, E. A. Oliveira Junior., S. Ferrari, P. C. Masiero, e A. F. Garcia. Extending UML components to develop software product-line architectures: Lessons learned. I. Crnkovic, V. Gruhn, e M. Book, editors, *5th European Conference on Software Architecture*

(ECSA 2011), volume 6903 of *Lecture Notes in Computer Science*, páginas 130–138. Springer, 2011.

[24] K. Czarnecki. Variability in software: State of the art and future directions. *International Conference on Fundamental Approaches to Software Engineering*, FASE'13, páginas 1–5, 2013.

[25] K. Deb, A. Pratap, S. Agarwal, e T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182 –197, abril de 2002.

[26] J. Derrac, S. García, D. Molina, e F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3 – 18, 2011.

[27] E. Dincel, N. Medvidovic, e A. van der Hoek. Measuring product line architectures. *Revised Papers from the 4th International Workshop on Software Product-Family Engineering (PFE'01)*, páginas 346–352, London, UK, 2002. Springer-Verlag.

[28] P. M. Donegan e P. C. Masiero. Design issues in a component-based software product line. *Proceedings of Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, páginas 3–16, 2007.

[29] J. J. Durillo e A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[30] G. Edwards, C. Seo, e N. Medvidovic. Model interpreter frameworks: A foundation for the analysis of domain-specific software architectures. *Journal of Universal Computer Science*, 14(8):1182–1206, apr de 2008. http://www.jucs.org/jucs_14_8/model_interpreter_frameworks_a.

[31] R. Etemaadi, M. T. M. Emmerich, e M. R. V. Chaudron. Problem-specific search operators for metaheuristic software architecture design. *Proceedings of the 4th Symposium on Search Based Software Engineering (SSBSE)*, páginas 267–272, 2012.

[32] L. Etxeberria e G. Sagardui. Product-line architecture: New issues for evaluation. *Proceedings of the International Software Product Line Conference (SPLC 2005)*, páginas 174–185, 2005.

[33] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Castor Filho, e F. Dantas. Evolving software product lines with aspects: an empirical study on design stability. *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, páginas 261–270, New York, NY, USA, 2008. ACM.

[34] E. Figueiredo, C. Sant'Anna, A. Garcia, e C. Lucena. Applying and evaluating concern-sensitive design heuristics. *Journal of Systems and Software*, 85(2):227 – 243, 2012.

[35] R. Galvan, A.R. Pozo, e S.R. Vergilio. Establishing Integration Test Orders for Aspect-Oriented Programs with an Evolutionary Strategy. *Proceedings of the Latin-American Workshop on Aspect Oriented Software (LA-WASP'2010)*, 2010.

[36] J. Garcia, D. Popescu, G. Edwards, e N. Medvidovic. Toward a catalogue of architectural bad smells. *Proceedings of the 5th International Conference on the Quality of Software Architectures (QoSA)*, páginas 146–162, Berlin, Heidelberg, 2009. Springer-Verlag.

[37] L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, páginas 849–852, New York, NY, USA, 2006. ACM.

[38] J. V. Gurp, J. Bosch, e M. Svahnberg. On the notion of variability in software product lines. *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, páginas 45. Washington, DC, USA: IEEE Computer Society, 2001.

[39] M. Harman. The current state and future of search based software engineering. *Future of Software Engineering (FOSE'2007)*, páginas 342–357, Washington, DC, USA, 2007. IEEE Computer Society.

[40] M. Harman e R. Hierons. A new representation and crossover operator for search-based optimization of software modularization. *Procedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, páginas 1351–1358, 2002.

[41] M. Harman, S. A. Mansouri, e Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Relatório Técnico TR-09-03, King's College London, April de 2009. http://www.dcs.kcl.ac.uk/technical-reports/papers/TR-09-03.pdf.

[42] M. Harman, P. McMinn, J. T. Souza, e S. Yoo. Search based software engineering: Techniques, taxonomy, tutorial. Bertrand Meyer e Martin Nordio, editors, *Empirical Software Engineering and Verification*, volume 7007 of *Lecture Notes in Computer Science*, páginas 1–59. Springer Berlin Heidelberg, 2012.

[43] M. Harman e L. Tratt. Pareto optimal search based refactoring at the design level. *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, páginas 1106–1113, New York, NY, USA, 2007. ACM.

[44] Mark Harman, S. Afshin Mansouri, e Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, dezembro de 2012.

[45] SEI Software Engineering Institute. Arcade Game Maker pedagogical product line, 2014. http://www.sei.cmu.edu/productlines/ppl/.

[46] SEI Software Engineering Institute. Software Product Lines, 2014. http://www.sei.cmu.edu/productlines/.

[47] A. C. Jensen e B. H.C. Cheng. 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), páginas 1341–1348, New York, NY, USA, 2010. ACM.

[48] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, e A. S. Peterson. Feature-oriented domain analysis FODA feasibility study. Relatório técnico, CMU/SEI-90-TR-21, Carnegie-Mellon University - Software Engineering Institute (SEI), 1990.

[49] R. Karimpour e G. Ruhe. Bi-criteria genetic search for adding new features into an existing product line. *Proceedings of the 1st International Workshop on Combining Modelling and Search Based Software Engineering, International Conference on Software Engineering (ICSE'2013)*, páginas 34–38, 2013.

[50] T. Kishi, N. Noda, e T. Katayama. Design verification for product line development. *Proceedings of the 9th International on Software Product Line Conference (SPLC'05)*, páginas 150–161, 2005.

[51] J. Knowles, L. Thiele, e E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. Relatório técnico, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, fevereiro de 2006. Revised version.

[52] J. D. Knowles e D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8:149–172, June de 2000.

[53] R. Kolb, I. John, J. Knodel, D. Muthig, U. Haury, e G. Meier. Experiences with product line development of embedded systems at testo ag. *Proceedings of the 10th International on Software Product Line Conference (SPLC'06)*, páginas 172–181, Washington, DC, USA, 2006. IEEE Computer Society.

[54] P. Kruchten. Architectural blueprints - the "4+1" view model of software architecture. *IEEE Software*, 12 (6):42 – 50, November de 1995.

[55] C. Krueger. Eliminating the adoption barrier. *IEEE Software - Special Issue on Iniciating Software Product Lines*, páginas 28–31, July/August de 2002.

[56] X.G. Kwong, C.K.; Luo e J.F. Tang. A multiobjective optimization approach for product line design. *IEEE Transactions on Engineering Management*, 58:97 – 108, 2011.

[57] R. Li, M. R. V. Chaudron, e R. C. Ladan. Towards automated software architectures design using model transformations and evolutionary algorithms. M. Pelikan e J. Branke, editors, *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'10)*, páginas 2097–2098. ACM, 2010.

[58] N. López, R. Casallas, e A. van der Hoek. Issues in mapping change-based product line architectures to configuration management systems. *Proceedings of the 13th International Software Product Line Conference (SPLC'09)*, páginas 21–30, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[59] R. E. Lopez-Herrejon, D. Batory, e W. Cook. Evaluating support for features in advanced modularization technologies. *Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP)*, páginas 169–194, 2005.

[60] R. E. Lopez-Herrejon e S. Trujillo. How complex is my product line? The case for variation point metrics. *Proceedings of the 2nd International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS 2008)*, páginas 17–23, 2008.

[61] A. Martens, H. Koziolek, S. Becker, e R. Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW '10)*, páginas 105–116, New York, NY, USA, 2010. ACM.

[62] R. Martin. Stability. Relatório técnico, C++ Report, 1997. http://www.objectmentor.com/resources/articles/stability.pdf.

[63] E. Niemelä, M. Matinlassi, e A. Taulavuori. Practical evaluation of software product family architectures. *Proceedings of the 8th International on Software Product Line Conference (SPLC'04)*, páginas 130–145, 2004.

[64] W. N. Oizumi. Uma representação computacional para o problema de otimização de projeto de arquitetura de linha de produto de software, 2012. Monografia de Trabalho de Conclusão de Curso, Departamento de Informática, Universidade Estadual de Maringá.

[65] M. O'Keeffe e M. Ó Cinnéide. Search-based refactoring: an empirical study. *Journal of Software Maintenance and Evolution: Research and Practice (Special Issue Search Based Software Engineering)*, 20:345–364, September de 2008.

[66] M. K. O'Keeffe e M. Ó Cinnéide. Getting the most from search-based refactoring. *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, páginas 1114–1120, New York, NY, USA, 2007. ACM.

[67] E. A. Oliveira Junior. *SystEM-PLA: um método sistemático para avaliação de arquitetura de linha de produto de software baseada em UML.* Tese de Doutorado, Universidade de São Paulo, São Carlos/SP, Brasil, 2010.

[68] E. A. Oliveira Junior, I. M. S. Gimenes, e J. C. Maldonado. Systematic management of variability in UML-based software product lines. *Journal of Universal Computer Science*, 16(17):2374–2393, sep de 2010.

[69] E. A. Oliveira Junior, I. M. S. Gimenes, J. C. Maldonado, P. C. Masiero, e L. Barroca. Systematic evaluation of software product line architectures. *Journal of Universal Computer Science (J.UCS)*, 19:25 – 52, 2013.

[70] E. A. Oliveira Junior, J. C. Maldonado, e I. M. S. Gimenes. Uma revisão sistemática sobre avaliação de linha de produto de software. Relatório técnico, Relatórios Técnicos do ICMC/USP, n. 310. ISSN - 0103-2569., 2007.

[71] V. Pareto. *Manuel D'Economie Politique*. Ams Press, Paris, 1927.

[72] K. Praditwong, M. Harman, e X. Yao. Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*, 37(2):264–282, 2011.

[73] Roger S. Pressman. *Engenharia de Software*. McGraw-Hill, 6a. edition, 2006.

[74] F. Qayum e R. Heckel. Local search-based refactoring as graph transformation. *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering (SSBSE'09)*, páginas 43–46, Washington, DC, USA, 2009. IEEE Computer Society.

[75] I. Radziukyniene e A. Zilinskas. Evolutionary Methods for Multi-Objective Portfolio Optimization. *Proceedings of the World Congress on Engineering 2008 Vol II*, jul de 2008.

[76] A. Rahman. Metrics for the structural assessment of product line architecture. Dissertação de Mestrado, School of Engineering, Blekinge Institute of Technology, 2004.

[77] O. Räihä. A survey on search-based software design. *Computer Science Review*, 4(4):203 – 249, 2010.

[78] O. Räihä. *Genetic Algorithms in Software Architecture Synthesis*. Tese de Doutorado, School of Information Sciences, University of Tampere, Tampere, Finland, 2011.

[79] O. Räihä, K. Koskimies, e E. Mäkinen. Genetic synthesis of software architecture. X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. Tan, J. Branke, e Y. Shi, editors, *Simulated Evolution*

and Learning, volume 5361 of *Lecture Notes in Computer Science*, páginas 565–574. Springer Berlin / Heidelberg, 2008.

[80] O. Räihä, K. Koskimies, e E. Mäkinen. Empirical study on the effect of crossover in genetic software architecture synthesis. *Proceedings of the 2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, páginas 619–625. IEEE, 2009.

[81] O. Räihä, K Koskimies, e E. Mäkinen. Complementary crossover for genetic software architecture synthesis. *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA 2010)*, páginas 266–271. IEEE, 2010.

[82] O. Räihä, K. Koskimies, e E. Mäkinen. Generating software architecture spectrum with multi-objective genetic algorithms. *Proceedings of the 2011 Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*, páginas 29–36. IEEE, 2011.

[83] C. N. Sant'Anna. *On the Modularity of Aspect-Oriented Design: A Concern-Driven Measurement Approach*. Tese de Doutorado, PUC-Rio, Rio de Janeiro/RJ, Brasil, 2008.

[84] J. Savolainen. Improving product line development with subject-oriented programming. *Proceedings of the Workshop on Multi-Dimensional Separation of Concerns in Software Engineering (ICSE 2000)*, 2000.

[85] A. S. Sayyad, T. Menzies, e H. Ammar. On the value of user preferences in search-based software engineering: a case study in software product lines. *International Conference on Software Engineering (ICSE'2013)*, páginas 492–501, 2013.

[86] K. Schmid e M. Verlage. The economic impact of product line adoption and evolution. *IEEE Software*, 19:50–57, July de 2002.

[87] O. Seng, J. Stammel, e D. Burkhart. Search-based determination of refactorings for improving the class structure of object-oriented systems. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, páginas 1909–1916, New York, NY, USA, 2006. ACM.

[88] C. L. Simons. *Interactive Evolutionary Computing in Early Lifecycle Software Engineering Design*. Tese de Doutorado, University of the West of England, Bristol, UK, 2011.

[89] C.L. Simons e I.C. Parmee. Elegant object-oriented software design via interactive, evolutionary computation. *IEEE Transactions on Systems, Man, and Cybernetics*, 42(6):1797 –1805, nov. de 2012.

[90] C.L. Simons, I.C. Parmee, e R. Gwynllyw. Interactive, evolutionary search in upstream object-oriented class design. *IEEE Transactions on Software Engineering*, 36(6):798 –816, nov.-dec. de 2010.

[91] Software Product Line Conferences (SPLC). Product line hall of fame, 2014. http://splc.net/fame.html.

[92] R. N. Taylor, N. Medvidovic, e E. M. Dashofy. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2010.

[93] S. Tsafarakis, Y. Marinakis, e N. Matsatsinis. Particle swarm optimization for optimal product line design. *International Journal of Research in Marketing*, In Press, Corrected Proof:–, junho de 2010.

[94] A. van der Hoek, E. Dincel, e N. Medvidovic. Using service utilization metrics to assess the structure of product line architectures. *Proceedings of the 9th International Symposium on Software Metrics*, páginas 298–308, Washington, DC, USA, 2003. IEEE Computer Society.

[95] F. van der Linden, J. Bosch, E. Kamsties, K. Känsälä, e H. Obbink. Software product family evaluation. Robert Nord, editor, *Software Product Lines*, volume

3154 of *Lecture Notes in Computer Science*, páginas 107–109. Springer Berlin / Heidelberg, 2004.

[96] F. van der Linden e E. Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[97] David A. Van Veldhuizen e Gary B. Lamont. Evolutionary Computation and Convergence to a Pareto Front. John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, jul de 1998. Stanford University Bookstore.

[98] S.R. Vergilio, A. Pozo, J.C. Árias, R.V. Cabral, e T. Nobre. Multi-objective optimization algorithms applied to the class integration and test order problem. *International Journal on Software Tools for Technology Transfer (STTT)*, 14(4):461–475, 2012. 10.1007/s10009-012-0226-1.

[99] S. Wang, S. Ali, e A. Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. *Proceedings of the 15th Genetic and Evolutionary Computation Conference (GECCO 2013)*, páginas 1493–1500, 2013.

[100] J. Wüst. SDMetrics, 2014. Disponível em http://www.sdmetrics.com/.

[101] C. Q. Xavier. Análise de estabilidade de diferentes versões de arquiteturas de linha de produto de software, 2011. Monografia de Trabalho de Conclusão de Curso, Departamento de Informática, Universidade Estadual de Maringá.

[102] T. Yu, A. Yassine, e D. Goldberg. An information theoretic method for developing modular architectures using genetic algorithms. *Research in Engineering Design*, 18:91–109, 2007. 10.1007/s00163-007-0030-1.

[103] E. Zitzler, M. Laumanns, e L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, e T. Fogarty, editors, *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*,

páginas 95–100, Athens, Greece, 2001. International Center for Numerical Methods in Engineering.

[104] E. Zitzler e L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[105] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, e V. G. Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2003.

[106] D. Zubrow e G. Chastek. Measures for software product lines. Relatório técnico, CMU/SEI-2003-TN-031, 2003. http://www.sei.cmu.edu/reports/03tn031.pdf.

APÊNDICE A

ARTIGO PUBLICADO NO SSBSE 2012

Applying Search Based Optimization to Software Product Line Architectures: Lessons Learned

Thelma Elita Colanzi[1,2] and Silvia Regina Vergilio[2]

[1] DIN - State University of Maringa, Av. Colombo, 5790, Maringá, Brazil
[2] DInf - Federal University of Parana, CP: 19081, CEP 19031-970, Curitiba, Brazil
thelma@din.uem.br,silvia@inf.ufpr.br**

Abstract. The Product-Line Architecture (PLA) is a fundamental SPL artifact. However, PLA design is a people-intensive and non-trivial task, and to find the best architecture can be formulated as an optimization problem with many objectives. We found several approaches that address search-based design of software architectures by using multi-objective evolutionary algorithms. However, such approaches have not been applied to PLAs. Considering such fact, in this work, we explore the use of these approaches to optimize PLAs. An extension of existing approaches is investigated, which uses specific metrics to evaluate the PLA characteristics. Then, we performed a case study involving one SPL. From the experience acquired during this study, we can relate some lessons learned, which are discussed in this work. Furthermore, the results point out that, in the case of PLAs, it is necessary to use SPL specific measures and evolutionary operators more sensitive to the SPL context.

Keywords: SPL; MOEAs; software architecture optimization

1 Introduction

Software Product Line (SPL) is a systematic approach for software reuse applied to a family of specific products within a well-defined domain. SPL approach allows to shift from the reuse of individual components to the large-scale reuse of a product-line architecture (PLA) [4]. A PLA defines the architectural elements required to realize feature commonality and variation across similar products.

The PLA design and its evolution is even more complex than traditional software architecting since, in order to maximize the reuse, some quality requirements such as modularity, stability and reusability play an important role. PLA design is a people-intensive, non-trivial and demanding task for software engineers to perform [12]. Software architecture design can be formulated as an optimization problem and has been investigated in related work [1, 9, 12]. These works have used Multi-Objective Evolutionary Algorithms (MOEAs) to optimize the design of software architectures and have achieved promising results. However, they have not been applied in the SPL context and do not consider specific characteristics of the PLA design.

** The authors would like to thank to CNPq for financial support.

Conventional architectural metrics [5] can be used to analyse the stability and modularity of any kind of software architecture, including PLAs, but it is also crucial to consider SPL metrics to evaluate PLAs due to its specific characteristics, such as variabilities, variation points, features assigned to elements, etc. In this sense, the research goal of this work is to investigate, through an exploratory study, how suitable is the use of existing search-based design approaches to the SPL context. To do this, we extend the existing approaches to PLA design by adding two specific metrics conceived to evaluate the cohesion of PLAs: CRSU (Compound Required Service Utilization) and CPSU (Compound Provided Service Utilization) [3]. Our extended approach is evaluated in a case study with the SPL Arcade Game Maker (AGM) [11]. Results of this study are presented, as well as, the main lessons learned.

The paper is organized as follows. Sections 2 and 3 respectively present related work and AGM. Section 4 describes how the case study was conducted. In the sequence, Section 5 relates the results and addresses the lessons learned during the study conduction. Finally, Section 6 concludes the paper.

2 Search-based Software Design

Räihä published a survey of works on search-based software design [8]. Among them, the works most related to ours address the architectural design outset. They use search-based techniques in order to optimize software architectures under development and consider two problems: the Class Responsibility Assignment (CRA) problem [1, 12] and the application of design patterns to design [9]. All of them use coupling and cohesion measures to evaluate the fitness of the solutions. There are two works based on the class diagram and MOEAs (SPEA2 and NSGA-II). They aim at improving the design by moving methods, attributes and relationships from one class to another, and adding and deleting classes [1, 12]. Another work [9] applies architectural design patterns for mutations and two quality metrics for improving modifiability and efficiency of the design. Empirical studies [10, 12] observed that the crossover operator does not offer significant advantage in the design optimization, when implemented in a random fashion. We can observe that the mentioned works successfully optimize software architectures, however, they have not been applied to optimize PLAs.

3 Arcade Game Maker (AGM)

Our study used the SPL AGM [11], which encompasses three arcade games: Brickles, Bowling, and Pong. It was created by the Software Engineering Institute (SEI) and conceived proactively in [2]. Its main variations are [7]: (i) specific rules of those games; (ii) kind, number and behavior of elements; and (iii) physical environment where the games take place. All variabilities of AGM were identified and represented rely on the use of the stereotype ≪variable≫ in the architecture models. This stereotype indicates that the component is formed by a set of classes with variabilities. For the sake of illustration, the model in Figure 1

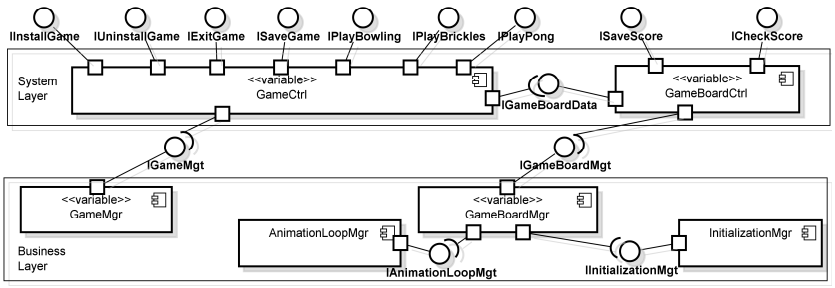


Fig. 1. AGM Component-based Architecture

represents a partial, high-level view of the PLA. Features are explicitly assigned to architectural elements using stereotypes, such as ≪Rule≫[1].

This architecture is in an intermediate stage since it is component-based, follows the layered architectural style and has manager classes that provide interfaces to enable the communication between layers. In order to improve the architectural quality, PLAs usually are conceived by using architectural styles, design patterns and frameworks. The AGM PLA has the system and the business layers. The system components have the suffix Ctrl and provide operations to the interface and dialog layers. The business components have the suffix Mgr and provide services to the system layer. The PLA is composed by 6 components and 45 classes.

The AGM PLA used in this work was analised in [2] by three specialists. The `GameCtrl` component was considered to accumulate too many responsibilities and, as a consequence, a candidate to manifest an architectural design anomaly.

4 Study Setting

Definition of the evaluation model: Aiming at obtaining highly cohesive, low coupling between components, stable and reusable PLAs, we used as objectives five metrics, supported by SDMetrics [13]: 3 conventional metrics, used in related work, and 2 SPL metrics. The metric Distance from the Main Sequence (D) [5] establishes a relation between abstraction and instability of a package. D values near to 0 represent components near the main sequence, i.e., there is a good balance between stability and abstraction [5]. The coupling metric Dependency of Packages (DepPack) [13] measures the number of packages on which classes and interfaces of this package depend. And, the size metric Number of Operations by Interface (NumOps) [13] counts the number of operations in each interface. CPSU and CRSU [3] allow to assess the internal cohesion of a PLA. CPSU is the rate of the number of services provided by components that is actually used by other PLA components. CRSU is the rate of the number of services required

[1] The detailed class diagram and all the solutions of our study case can be found at http://www.inf.ufpr.br/gres/apoio_en.html.

by components that are actually used by other PLA components. CPSU and CRSU values close to 1 signify self-contained, fully functional architectures.

From these metrics, we defined five objectives in our evaluation model: (a) Sum of D; (b) Mean of NumOps of all the PLA interfaces; (c) Mean of DepPack; (d) CPSU; and (e) CRSU. We intended to minimize the three first objectives and to maximize the last two ones. CPSU and CRSU, D and DepPack provide, respectively, indicators about cohesion, stability and coupling of the PLA components. We believe that the size metric NumOps allows to observe some aspect of the PLA reusability since small interfaces are, in general, easier to reuse. We worked with models of the PLA in two levels: high-level and detailed level. DepPack, D and NumOps were applied on the detailed level of the PLA, and CPSU and CRSU were applied on the high-level PLA.

Evolutionary and Multi-objective Approach: The approach is based on the evaluation model described before that considers specific SPL metrics in addition to the conventional ones. Each individual consists on a PLA, represented as a class diagram. Such diagram, converted to a XMI file, serves as input to SDMetrics. The MOEA used is inspired on NSGA-II. Each population is formed by 5 individuals (alternative design to the AGM PLA). One mutation operator is applied on each individual from parent population in order to obtain the offspring. For each individual, some mutation operator is chosen randomly, as well as, the source and the target architectural elements to be mutated. The mutations are based on [1, 12]: *Move Method, Move Attribute, Add Class*. As the AGM PLA has manager classes with interfaces, we use two additional operators: *Move Operation* and *Add Manager Class*. The first operator moves operations between interfaces belonging to manager classes. The second one moves an attribute/operation to a new manager class. Architectural elements must maintain the assigned feature even if it is moved in the PLA. We did not apply the crossover operator since there is not a consensus on the benefits of such operator in this context [10, 12]. Although, it is an interesting further work to perform.

The initial population is composed by the AGM PLA [2] and other four mutated individuals [9]. After achieving an offspring population, the algorithm selects the best 5 individuals (from parent and offspring populations) to obtain the next generation. The fitness of all individuals is obtained by using the metrics of the evaluation model with five objectives. As a result, the next generation is composed by non-dominated solutions from the first front, plus the solutions from the second front, and so on until that the new population is completed. The evolutionary process consists on five generations.

5 Results and Analysis

Table 1 contains the fitness of the solutions obtained at the end of the evolutionary process. All these non-dominated solutions have the same number of classes. They have the same CPSU and CRSU values and the best values for each other metric are in bold. Their fitness are slightly better than the original PLA's fitness (2.562, 3.57, 1.8, 0.579, 1). So, according to the metrics, we can state that

Table 1. Fitness of the non-dominated solutions found

Solution	Sum of D	Mean of NumOps	Mean of DepPack	CPSU	CRSU
4b	**2.528**	3.26	**1.75**	0.579	1
5b	2.736	**3.20**	2.50	0.579	1
6d	2.562	**3.20**	2.50	0.579	1
6e	**2.528**	3.26	**1.75**	0.579	1

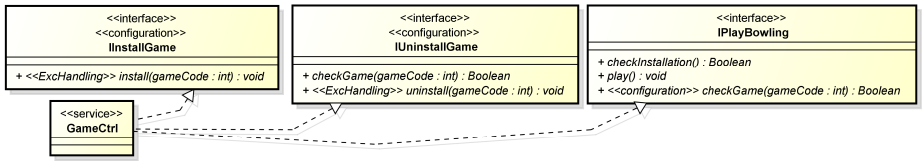


Fig. 2. Partial PLA Corresponding to Solution 4c

our solutions are more stable and more reusable than the original PLA. One responsibility of the component `GameCtrl` in Solutions **5b** and **6e** was delegated to a new manager class in only 5 generations. Therefore, in solutions found after many generations would be possible that some responsibilities assigned to this component were assigned to another one. As a result, `GameCtrl` would be not a candidate to manifest an architectural design anomaly.

From this study, the following lessons were learned:

Features Modularization: The used metrics are not sensitive to features modularization. The mutations can generate better fitness for some objectives, but in some cases they are not good changes since some SPL features became more scattered or tangled. To illustrate this, Figure 2 presents a fragment of Solution **4c**, which was obtained after the application of the mutation operator *Move Operation* for moving `checkGame` from `IInstallGame` to `IPlayBowling` in `GameCtrl`. Before the mutation, the feature *configuration* was assigned only to interfaces `IInstallGame` and `IUninstallGame` of `GameCtrl`. After that, this feature becames more scattered because there are 3 interfaces assigned to it in Solution **4c**. Due to their definitions, none of the used metrics was sensitive to this kind of scattering. On the other hand, in Solution **6c** (Figure 3) the feature *configuration* became more modularized and less tangled with other features after the mutation that moved the operation `checkInstallation` from the interface `IGameMgt` of `GameMgr` to the interface `Interface_1` of `Classe_2`. Despite of these two facts, none metric was sensitive to this benefit for the PLA. The solution that better modularizes the feature *configuration*, is a dominated solution, and due to this, it is not included in the MOEA output.

PLA Cohesion: In our case study, CRSU and CPSU were not so sensitive to changes performed in the PLAs. In some cases, there are significant changes in the PLA, not captured by these metrics. It occurs because their definition are based on the satisfaction and utilization of the component services. So, changes inside a component that improve its internal cohesion are not detected. Many times changes inside components do not change the provided or required services. In addition to the feature modularization, in the SPL context it is interesting to have a PLA where the features are more cohesive as possible. However, CRSU

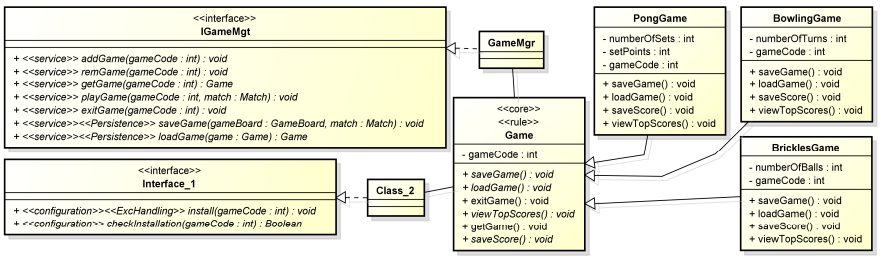


Fig. 3. Partial PLA Corresponding to Solution 6c

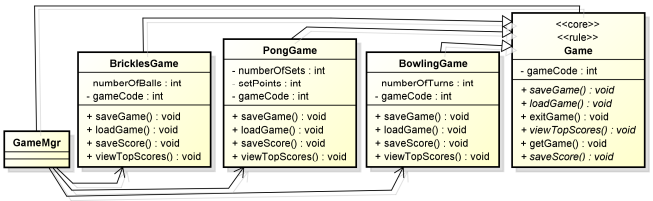


Fig. 4. Partial PLA Corresponding to Solution 5b

and CPSU are not sensitive to feature cohesion. For example, Solution **6c** (Figure 3) has better feature cohesion than Solution **6d** regarding to the feature *configuration*, but this fact is not captured by these three metrics.

PLA Design Stage: The design stage of the PLA provided as the input to the search-based optimization impacts on the obtained results. With the AGM PLA the mutation operators could not obtain design solutions much better than it. In addition, the values of some metrics (CPSU and CRSU) did not change after mutations. For sure if we have performed a lot of iterations on the evolutionary process, we could obtain better design solutions. For instance, in some solutions the attribute `menu` was moved from `GameBoard` to a new class. This mutation could carry to a solution where all responsibilities regarding to the menu of the game were moved to that new class, so `GameBoard` would become a less bloated class. However, we could observe that the used mutation operators and metrics were not enough to achieve satisfactory results. So, considering that AGM PLA is in an intermediate stage, we can state that other mutation operators and other metrics more suitable to SPL context are necessary, for instance, mutation operators associated to design patterns or architectural styles.

Restrictions for the genetic operators: As mentioned before, the original PLA follows the layered style, where the business layer should not require services from the system layer. Despite of this, one solution (Solution **2d**) does not follow this rule. However, we have not established restrictions concerning to the architectural styles to apply the mutation operators. So, we can conclude that the genetic operators must be aware of the rules defined by the adopted architecture style. This can be done by defining restrictions to apply the operators.

There are some threats to our study: the sample size since our study focused on only one SPL; small populations; and few generations. However, this reduced number allowed us to perform an in-depth qualitative analysis that took

a wide range of perspectives into account. The construct validity of the study with respect to the used metrics and operators is guaranteed by their previous successful application and evaluation by other researchers [1, 3, 5, 6, 12].

6 Concluding Remarks

The main contributions of this work are: the first case study where an search-based design approach was applied to SPL context and the lessons learned about such study. We also extended an existing approach by using two metrics defined to evaluate PLAs. The results point out that existing search-based approaches can be used to achieve better PLA design. Although, the results would be better if some points discussed in the lessons learned were improved. The two SPL metrics used were not sensitive to the points highlighted in the lessons learned.

Our ongoing work encompasses: a) the use of metrics to assess cohesion and modularization of SPL features; b) evolutionary operators sensitive to the SPL context, including modularization of features; and c) extensibility of the PLA. We also intend to investigate the crossover operator feasibility in the SPL context.

References

1. Bowman, M., Briand, L.C., Labiche, Y.: Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. IEEE Transactions on Software Engineering 36(6), 817–837 (Nov 2010)
2. Contieri Junior, A.C., et al.: Extending UML Components to develop software product-line architectures: Lessons learned. In: Proceedings of Eur. Conf. on Software Architecture (ECSA). pp. 130–138 (2011)
3. Hoek, A.v.d., Dincel, E., Medvidovic, N.: Using service utilization metrics to assess the structure of product line architectures. In: Proceedings of the 9th Int. Symposium on Software Metrics. pp. 298–308. IEEE, Washington, DC, USA (2003)
4. Linden, F.v.d., Schmid, F., Rommes, E.: Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering. Springer (2007)
5. Martin, R.: Stability – C++ Report. Tech. rep. (1997), http://www.objectmentor.com/resources/articles/stability.pdf
6. Martin, R.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall (2003)
7. Oliveira Junior, E.A., et al.: Systematic Management of Variability in UML-based Software Product Lines. J. of Universal Computer Science 16(17), 2374–2393 (2010)
8. Räihä, O.: A survey on search-based software design. Computer Science Review 4(4), 203 – 249 (2010)
9. Räihä, O.: Genetic Algorithms in Software Architecture Synthesis. Ph.D. thesis, School of Information Sciences, University of Tampere, Tampere, Finland (2011)
10. Räihä, O., Koskimies, K., Mäkinen, E.: Empirical study on the effect of crossover in genetic software architecture synthesis. In: Proceedings of World Congress on Nature and Biologically Inspired Computing (NaBIC). pp. 619–625. IEEE (2009)
11. SEI: Arcade Game Maker pedagogical product line (2012), http://www.sei.cmu.edu/productlines/ppl/
12. Simons, C.L.: Interactive Evolutionary Computing in Early Lifecycle Software Engineering Design. Ph.D. thesis, University of the West of England, Bristol (2011)
13. Wüst, J.: SDMetrics (2012), available at http://www.sdmetrics.com/

APÊNDICE B

ARTIGO PUBLICADO NO CMSBSE 2013

Representation of Software Product Line Architectures for Search-Based Design

Thelma Elita Colanzi[1,2], Silvia Regina Vergilio[1]
[1]Computer Science Department Federal University of Paraná (UFPR)
CP: 19081, CEP: 19031-970, Curitiba, Brazil
[2]Computer Science Departament of State University of Maringá (UEM)
CEP: 87.020-900, Maringá, Brazil
{thelmae, silvia}@inf.ufpr.br

***Abstract*—The Product-Line Architecture (PLA) is the main artifact of a Software Product Line (SPL). Search-based approaches can provide automated discovery of near-optimal PLAs and make its design less dependent on human architects. To do this, it is necessary to adopt a suitable PLA representation to apply the search operators. In this sense, we review existing architecture representations proposed by related work, but all of them need to be extended to encompass specific characteristics of SPL. Then, the use of such representations for PLA is discussed and, based on the performed analysis, we introduce a novel direct PLA representation for search-based optimization. Some implementation aspects are discussed involving implementation details about the proposed PLA representation, constraints and impact on specific search operators. Ongoing work addresses the application of specific search operators for the proposed representation and the definition of a fitness function to be applied in a multi-objective search-based approach for the PLA design.**

***Index Terms*—architecture modelling, software product line, multi-objective search-based approach.**

I. Introduction

A Software Product Line (SPL) is comprised by core assets that have explicit common and variable features [1]. Features are attributes of a software system that affect directly the final users and they are usually represented in feature models. A product of the SPL is given by a combination of its features.

Several SPL development activities can be benefitted by search-based approaches. The design of software Product Line Architectures (PLAs) is one of them. PLAs entail a design that is common to all the products derived from the SPL [1]. The design of a PLA should encompass the components realizing all the mandatory and varying features in a domain [2]. So, a PLA is a key asset in SPL Engineering since it allows the large-scale reuse. The focus of this work is the PLA design, represented by UML class diagrams, since this kind of model is commonly used to model software architectures in the detailed level [3].

The presence of many inter-related variation points, the modularity maximization, the fulfillment of quality requirements, etc., turn the PLA design into a people-intensive task. Search-based approaches can provide automated discovery of near-optimal PLAs and make its design less dependent on human architects. The PLA design can be optimized to fulfill, in addition to basic design principles like coupling and cohesion, other quality requirements such as modularization of the features, extensibility, complexity, etc. Independently of the used search-based technique, it is important to have a suitable representation to PLAs. Besides our previous work [4], in which we present lessons learned about the PLA design optimization, we could not to find, in the literature, studies where the PLA design is the target of search-based optimization. So, from the best of our knowledge, there is not proposed representations for PLA design.

In the context of software architecture design, several studies adopted Search-Based Software Engineering (SBSE) techniques to object-oriented software design, service-oriented software design, software modularization and software refactoring [5]. Most studies adopt representations that are not natural and not able to represent variabilities and features, such as integer vector [3] and supergene representations [6]. Furthermore, the use of such representations may turn complex the search operators implementation.

Metaheuristic approaches, such as evolutionary algorithms, usually have standard representations (bit strings, real-values vectors, etc.). To apply them to a complex search space, usually a genotype-phenotype mapping is necessary, i.e., a mapping between the standard representation and the represented solution [7]. On the other hand, it is possible to apply search operators directly on the natural representation of a solution, for instance, class diagrams. It is noticeable that some related works adopt direct representations for the software architecture design and other ones perform a genotype-phenotype mapping. In the direct representation the natural representation of the architecture is the target of search operators.

Given the aforementioned context, an open question can be posed:*"What is the most suitable PLA representation? Can we use a direct PLA representation or do we need a genotype-phenotype mapping?"*. Such questions are addressed in this paper in the following way. First, we review existing architecture representations proposed by related work. They are compared briefly. After this, we make some comments about the use of such representations for PLA. Based on the performed analysis we introduce a direct PLA representation for search-based optimization.

TABLE I
EXISTING REPRESENTATIONS USED IN RELATED WORK

	Study	Context	Representation	Applied to SPL
Genotype-Phenotype Mapping	Grunske [8]	Deployment Architectural Refactoring	Vector of integers	No
	Martens et al. [9]	Deployment Architectural Refactoring	Vector of characters	No
	Räihä [6]	Application of design patterns to architecture design	Vector of supergenes	No
	Bowman et al. [3]	Class Responsibility Assignment	Vector of integers	No
Direct Representation	Simons et al. [10]	Class Responsibility Assignment	Object-oriented representation	No
	Etemaadi et al. [7]	Application of patterns and antipatterns to architecture design	ADL	No

The paper is organized in sections. Section II presents the representations used in related work. Section III addresses the use of the existing representation to search-based PLA design. Section IV introduces a direct representation. Finally, Section V concludes the paper.

II. ARCHITECTURE REPRESENTATIONS: A REVIEW IN THE SEARCH-BASED CONTEXT

As mentioned before, we have not found works addressing PLA representations for search-based design. The subject of the most related existing works is on software architecture optimization. Räihä has recently published a *survey* describing studies on search-based software design [5]. Some of them apply SBSE techniques to object-oriented design and they are briefly reviewed below, emphasizing the adopted representation.

Some studies focus on the refactoring of existing architectures aiming at improving their structural quality [11]–[14]. They represent the architecture in class diagrams but optimize the architectural design indirectly since the search operators are applied on graphs that represent the architectural transformations. Hence the search space is composed by the alternatives of transformation sequences.

However, the architectural transformation sequence cannot be so relevant in the search-based architecture design outset, so the optimization of the architecture is more natural despite more complex. Another point to be considered in this case is to maintain the consistency of the solutions after the search operators application.

In several studies the architectural transformations (through search operators) are applied directly to the software design. That is, the possible variants of the design form the search space. To apply metaheuristics, some studies perform a genotype-phenotype mapping and apply the search operators on strings or vectors [3], [6], [8], [9]; and other ones perform the search using the natural representation of the design [7], [10].

Some of them are concerned to quality requirements optimization in the software architecture [8], [9]. Other ones focus on the architectural design outset by addressing two problems related to software design: the Class Responsibility Assignment (CRA) [3], [10] and the application of design patterns to software design [6], [7]. Table I presents the kind of representations adopted by related works that optimize the own software architecture. The following sections present more details about the representations used.

A. Genotype-Phenotype Mapping

1) Deployment Architectural Refactoring: Grunske [8] introduces a multi-objective strategy based on architecture refactoring that performs architectural transformations and identifies alternatives considering functional and quality requirements. Two objectives are adopted: to reduce development cost of satellites and to improve the reliability on the two main functions of the satellites. The representation consists of a vector of integers where each position represents a component and its value identifies the number of redundant components used. The strategy allows automatic architectural transformations in a deployment view to improve the quality requirements without impact on the system behaviour.

Martens et al. [9] propose a multi-objective approach to optimize the performance, reliability and cost of component-based architectures in a deployment view. They employ the metaheuristic NSGA-II and represent each chromosome by a vector of chosen design options.

2) Architectural design outset: Bowman et al. [3] present a multi-objective and evolutionary approach to optimize the CRA problem, considering coupling and cohesion measures. The approach works with the class diagram and the algorithm SPEA2. Aiming at obtaining a better domain model, the approach moves methods, attributes and relationships from one class to another. In addition, the algorithm adds and deletes classes. The chromosome representation tracks the attributes and methods and the class to which they belong. So, attribute and method are assigned to a gene within the chromosome. Chromosomes simply consist of integer values, where the position of the gene within the chromosome represents the class member (attribute or method) and the gene's integer value denotes their class assignment. Dependency information is represented by a dependency matrix, stored separately since the dependency information does not change during the search.

Räihä et al. [6] introduce an approach to synthesize software architecture by using genetic algorithms (GA). The approach applies architectural design patterns for mutations and two quality metrics for evaluating individual architectures: modifiability and efficiency. The fitness function is expressed in terms of coupling and cohesion metrics, and crossover operation can be performed by merging two architectures without breaking

existing pattern instances. The initial architecture obtained from the functional requirements is provided as the input to the GA. The architecture is encoded in a chromosome that consists of a vector of supergenes to represent all the operations (responsibilities) of the system. Each supergene is a vector of integers that contains information about one operation such as: name, type, the class it belongs to, the interface that it implements, design patterns associated, invoked operations, etc. It is an operation-driven representation due to the patterns applied are concerned to operations. Raiha et al. state that using such representation makes it easier to keep the architecture consistent, as no responsibility can be left out of the architecture at any point, and there is no risk of breaking the dependencies between responsibilities.

B. Direct Representation

Simons et al. [10] propose a search-based approach to find the best object-oriented conceptual software design. This approach encompasses a direct, novel object-oriented representation and genetic operators for evolutionary search. The genetic operators change methods and attributes from one class to another, and also add new classes. The fitness of each solution is obtained from coupling and cohesion metrics and the algorithm used was NSGA-II. The representation comprises classes, methods, and attributes. The design search space thus comprises a set of attributes and a set of methods, derived from use cases. Classes are represented as groupings of methods and attributes.

Etemaadi et al. [7] introduce a new hybridization process for solving the architecture design problem, in which it uses quality improvement heuristics within an evolutionary algorithm. This approach uses a direct representation within a modeling language, and specific search operators that can be implemented by the model-to-model transformation language. So, the solution is represented in a system model representation, such as an ADL (Architecture Description Language), instead of genotype-phenotype mapping approach. Authors state that existing patterns and antipatterns can be implemented as search operators to optimize the system architecture design.

We can observe that the mentioned works successfully optimize software architectures, however, they have not been applied in the SPL context in order to optimize PLAs. Furthermore, they do not consider specific characteristics of SPLs, such as variabilities, features assigned to architectural elements, and so on. In this sense, it is necessary to define a novel representation specific to PLAs. Such a representation is introduced in the next section.

III. An analysis considering PLA design

Aiming at applying search-based approaches to optimize the PLA design it is necessary to define an appropriate representation since it impacts on the search operators implementation. As no related work uses PLA representation at class diagram level, we have analysed the adaptation possibilities of each representation used in related work to the PLA design context.

Representations based on trees of the genetic programming or directed graphs of the evolutionary programming seem interesting representations, but due to the non-hierarchical structure of the software architectures such representations do not fit well [10]. Such graphs are adequated to represent the architecture transformation sequences used in [11], [13], [14], as mentioned in Section II. However we are interested in the optimization of the architecture design instead of transformation sequences.

The representation used in [3] with vector of integers is adaptable to PLA design. However it is necessary to find ways to represent variabilities and its variants and features assigned to architectural elements by using integers. In the same way, the supergene adopted in [6] can be used despite it needs to be complemented to accommodate variabilities and features. In both cases, information about dependencies and features assigned to architectural elements would be represented in a matrix separated from the chromosome. However, information regarding to the variabilities, their variation points, possible variants, and the places where each of them should be inside the chromosome since they can be changed during the search process. Such adaptation would increase the complexity of the representation becoming the application of standard search operators infeasible and making their implementation more complex.

Similarly to the approach proposed by Etemaadi et al. [7], we could evolve the PLA design specified by an ADL. However, the ADLs that support SPL specification do not encompass representation of classes [15]–[19]. They are limited to model PLAs by using only the component-connector model. In addition to this, only LightPL-Acme [18], Koala [19] and xADL-SML [17] provide automated support.

In this sense, the object-oriented representation [10] seems to be more suitable and natural than others. It is easier to extend it in order to represent variabilities and features. So, we proposed a novel representation that comprises all the needs regarding to SPL aiming at PLA optimization. This representation is described in next section.

IV. The proposed PLA representation

Considering the analysis of the existing representations, we introduce in this section a direct PLA representation. It consists on a metamodel, presented in Figure 1. This metamodel is based on the work of Chang et al. [20].

A PLA contains architectural elements (components, interfaces, operations and their inter-relationships) besides to be conform to some architectural style or design pattern. Each architectural element is assigned to feature(s) by using UML stereotypes and it can be common to all SPL products or variable, being present only in some product(s). Variable elements are associated to variabilities that have variation points and their variants.

Figure 2 depicts an example of a PLA for the SPL Arcade Game Maker (AGM) [21] that can be represented by the proposed metamodel. AGM encompasses three simple arcade games and it was created to support learning about and

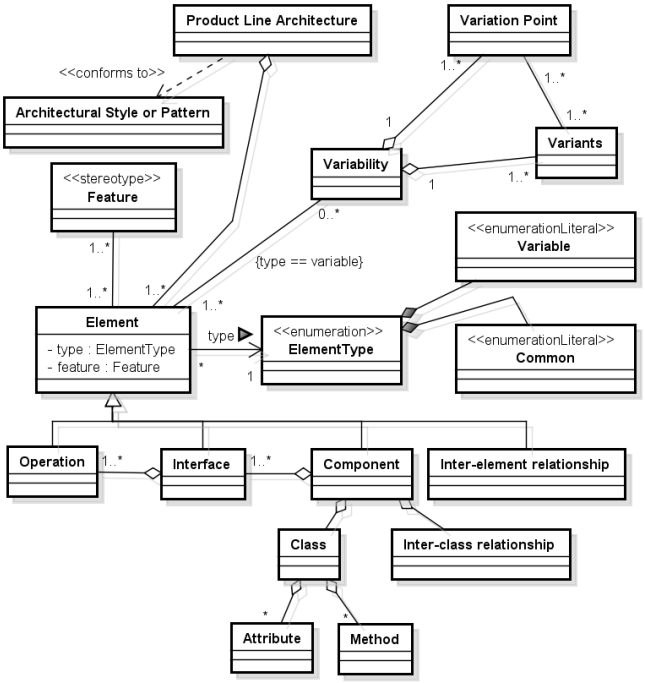


Fig. 1. PLA Representation by Metamodel

experimenting with SPL. In this example, components are represented by packages and attributes and operations were omitted in classes and interfaces. It is possible to see features assigned to some architectural elements by using stereotypes, such as action, rule, configuration and service. We are using SMarty [22] to represent variabilities in PLAs. According to the SMarty notation, there are four variabilities presented in notes assigned to classes. Variation points are labeled with the stereotype $<< variationPoint >>$ and variants are labeled with stereotype $<< alternative\_OR >>$. Mandatory variation points are also labeled with the stereotype $<< mandatory >>$.

Adopting a direct PLA representation implies in to propose specific search operators and to establish constraints to maintain the consistency of the PLA design during the search process. Though if we had extended any standard representation probably it would be necessary to adapt the standard search operators as well. In the next section we discuss some implementation aspects of using the proposed PLA representation.

A. Implementation Aspects

In our research, we have decided to start the optimization of PLA design by applying the direct PLA representation with multi-objective algorithms since they have achieved good results in SBSE. We are instantiating the framework JMetal to the context of our problem.

1) Implementing the Representation: We are modeling a PLA by using the tool Poseidon[1] and exporting the model in a XMI[2] file. In our search-based approach [23], the PLA representation is generated from this XMI file and following the proposed metamodel (Figure 1). During the XMI processing to generate the PLA representation, for each XMI element identified an object of the metamodel is instantiated according to the type of XMI element. So, the PLA representation will contain objects to represent all ar chitectural elements, their inter-relationships, all variation points, variants and features assigned to each architectural element.

During the evolutionary process the search operators are applied on the PLA representation, and, at the end, a set of PLA representations is generated as output. This set of PLA representations should be converted in a legible view to the architect, i.e., XMI file to be seen in Poseidon. One alternative that prioritizes some objective(s) can be adopted as the PLA according to the organization priorities.

We are using the framework UML2[3] of the Eclipse Foundation to generates XMI files containing the PLA design alternatives obtained from the search process. However, Poseidon cannot read the XMI files generated by UML2 due to some specific details that Poseidon includes in its XMI files. Hence we could not read the XMI files generated as output of our implementation in Poseidon. Nowadays, we are trying to adapt our existing code to work with XMI files generated by the modeling tool Papyrus[4] that is integrated with XML2 in order to create and to read the input and output of the search-based approach, respectively.

2) Search operators: We have implemented five mutation operators for software architecture based on mutation operators used in related work [3], [24] plus one mutation operator and one crossover operator specific for PLA design optimization proposed by us. The proposed operators are driven to PLA feature modularization, considering that a feature usually is solved by an architectural elements group. The mutation operators are described below:

- **Move Method**: moves a method to other existing class into the component. The class that receives the moved method becomes client from the original class of the method.
- **Move Attribute**: moves an attribute to other existing class into the component. The class that receives the moved attribute becomes client from the original class of the attribute.
- **Add Class**: moves a method or an attribute to a new class into the component. The addition of classes is necessary since optimal class assignments may require additional classes that were not identified in the design outset. The new class becomes client from the original class of the moved method/attribute.

[1] $< http://www.gentleware.com/ >$
[2] $< http://www.omg.org/spec/XMI/2.1.1/ >$
[3] $< www.eclipse.org/uml2/ >$
[4] $< http://www.eclipse.org/modeling/mdt/papyrus/ >$

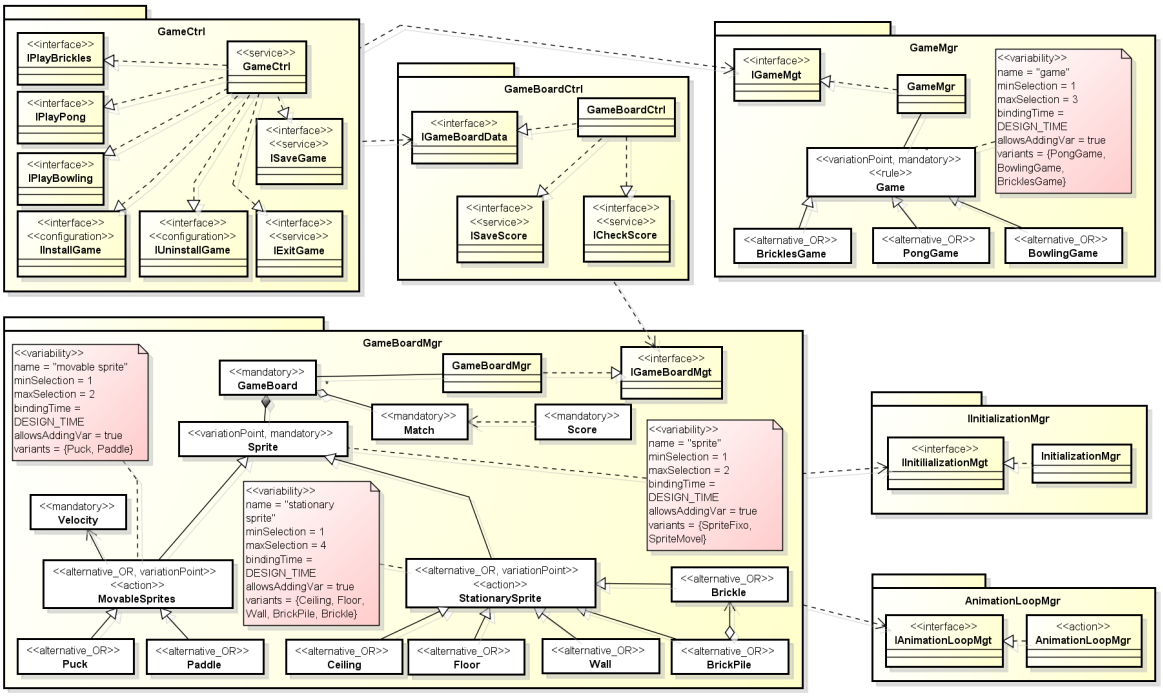


Fig. 2. Example of a PLA

- **Move Operation**: moves an operation from an interface to another. The component that receives the moved operation becomes client from the original interface of the operation.
- **Add Component**: similar to *Add Class*, it creates a new component and a new interface for this component, and then moves an operation to the new interface. The new component becomes client from the original interface of the operation.
- **Feature-driven Mutation**: it aims at modularizing a feature tangled with others in a component. It selects an arbitrary component (c_x), and, if c_x have architectural elements (interfaces, classes, operations, methods and attributes) assigned to different features, an arbitrary feature (f_x) is selected to be modularized in a new component (c_z). All architectural elements from c_x assigned to f_x are moved to c_z. c_z becomes client from the original components from the moved architectural elements.

As the merging of two architectures seems not to be natural and there are evidences that an arbitrary crossover does not offer significant advantage in the software architecture optimization [24], [25], we have proposed an specific crossover operator based on modularization of SPL features. The **Feature-driven Crossover** randomly selects a feature (f_x) and then creates new individuals swapping the architectural elements (classes, interfaces, operations, etc.) that realize the feature f_x. From two parents ($P1$ and $P2$), two offsprings are generated as follow: $Child1$ contains the elements (and their relationships) that realize f_x from $P2$ and the other elements from $P1$; $Child2$ contains the elements (and their relationships) that realize f_x from $P1$ and the other elements from $P2$. Thus, it is possible to obtain individuals that were mutated and that better realize the feature f_x.

The manipulation of the PLA representation was natural and easy during the implementation of all operators. We believe that by using other kind of representation with genotype-phenotype mapping the implementation of the operators would be more difficult.

3) Constraints: During the search process, some constraints are applied aiming at preserving the consistency of each generated solution. They are:

- Classes and their subclasses cannot be separated.
- The moved element must maintain the original assigned feature even if it was moved to an element assigned to a different feature.
- Components, interfaces and classes cannot be empty in the design.

The two first constraints were implemented into the search operators. So, if a superclass was selected to be moved, its subclasses are moved together. We decided to repair the solutions that not satisfy the last constraint instead of excluding them from the population. By reparing a solution we can maintain the population diversity and satisfactory alternatives of design.

V. Concluding Remarks

In this paper we analysed the use of existing architecture representations proposed in the literature for the PLA context. From the performed analysis we conclude that both representations, direct and through genotype-phenotype mapping,

imply on the implementation of specific search operators and the existing representations need to be adapted to be applied in the search-based PLA design. So, we introduced a novel direct PLA representation for the application of search-based approaches to the PLA design since it is more natural and makes the implementation of specific search operators easier. We also discussed some implementation aspects regarding the proposed representation.

In addition to the proposed representation and specific search operators, other point to be investigated is related to metrics for fitness evaluation in PLA search-based design. Such subject was addressed in our previous work [4], but this still is an open question.

In this sense, our ongoing work involves the application of specific search operators and the definition of a fitness function to be applied in a multi-objective approach for the PLA design optimization.

ACKNOWLEDGMENT

This work is supported by CNPq.

REFERENCES

[1] F. v. d. Linden, F. Schmid, and E. Rommes, *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[2] A. C. Contieri Junior, G. G. Correia, T. E. Colanzi, I. M. de Souza Gimenes, E. A. Oliveira Junior, S. Ferrari, P. C. Masiero, and A. F. Garcia, "Extending UML components to develop software product-line architectures: Lessons learned," in *Proceeding of the 5th European Conference on Software Architecture*, ser. ECSA'11, I. Crnkovic, V. Gruhn, and M. Book, Eds., vol. 6903. Springer, 2011, pp. 130–138.

[3] M. Bowman, L. C. Briand, and Y. Labiche, "Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 817–837, 2010.

[4] T. E. Colanzi and S. R. Vergilio, "Applying search based optimization to software product line architectures: Lessons learned," in *Proceedings of the 4th International Symposium on Search Based Software Engineering*, ser. SSBSE'12, vol. 7515, 2012, pp. 259–266.

[5] O. Räihä, "A survey on search-based software design," *Computer Science Review*, vol. 4, no. 4, pp. 203 – 249, 2010.

[6] O. Räihä, K. Koskimies, and E. Mäkinen, "Generating software architecture spectrum with multi-objective genetic algorithms," in *2011 Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*. IEEE, 2011, pp. 29–36.

[7] R. Etemaadi, M. T. M. Emmerich, and M. R. V. Chaudron, "Problem-specific search operators for metaheuristic software architecture design," in *Proceedings of the 4th International Symposium on Search Based Software Engineering*, ser. SSBSE'12, 2012, pp. 267–272.

[8] L. Grunske, "Identifying "good" architectural design alternatives with multi-objective optimization strategies," in *Proceeding of ICSE'06*. New York, NY, USA: ACM, 2006, pp. 849–852.

[9] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," in *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW '10)*. New York, NY, USA: ACM, 2010, pp. 105–116.

[10] C. Simons, I. Parmee, and R. Gwynllyw, "Interactive, evolutionary search in upstream object-oriented class design," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 798 –816, nov.-dec. 2010.

[11] M. Amoui, S. Mirarab, S. Ansari, and C. Lucas, "A genetic algorithm approach to design evolution using design pattern transformation," *International Journal of Information Technology and Intelligent Computing*, vol. 1, pp. 235–245, 2006.

[12] M. Harman and L. Tratt, "Pareto optimal search based refactoring at the design level," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*. New York, NY, USA: ACM, 2007, pp. 1106–1113.

[13] A. C. Jensen and B. H. Cheng, "On the use of genetic programming for automated refactoring and the introduction of design patterns," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*. New York, NY, USA: ACM, 2010, pp. 1341–1348.

[14] F. Qayum and R. Heckel, "Local search-based refactoring as graph transformation," in *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering (SSBSE'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 43–46.

[15] E. A. Barbosa, T. Batista, A. Garcia, and E. Silva, "Pl-aspectualacme: an aspect-oriented architectural description language for software product lines," in *Proceedings of the 5th European Conference on Software Architecture*, ser. ECSA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 139–146.

[16] R. Bashroush, T. Brown, I. Spence, and P. Kilpatrick, "Adlars: An architecture description language for software product lines," *Software Engineering Workshop, Annual IEEE/NASA Goddard*, vol. 0, pp. 163–173, 2005.

[17] E. M. Dashofy, A. v. d. Hoek, and R. N. Taylor, "A comprehensive approach for the development of modular software architecture description languages," *ACM Transactions on Software Engineering Methodology*, vol. 14, no. 2, pp. 199–245, Apr. 2005.

[18] E. Silva and E. Cavalcante. (2012) LightPL-ACME Studio. [Available at: http://www.dimap.ufrn.br/lightplacmestudio/index.php. Acessed in 12/12/2012.].

[19] J. C. Trigaux and P. Heymans, "Software Product Lines: State of the art," Institut d'Informatique FUNDP, Technical Report, Sep. 2003.

[20] S. H. Chang, H. J. La, and S. D. Kim, "Key issues and metrics for evaluating product line architectures," in *SEKE*, K. Zhang, G. Spanoudakis, and G. Visaggio, Eds., 2006, pp. 212–219.

[21] S. S. E. Institute. (2013) Arcade Game Maker pedagogical product line. Http://www.sei.cmu.edu/productlines/ppl/.

[22] E. A. O. Junior, I. M. S. Gimenes, and J. C. Maldonado, "Systematic management of variability in uml-based software product lines," *Journal of Universal Computer Science*, vol. 16, no. 17, pp. 2374–2393, sep 2010.

[23] T. Colanzi, "Search based design of software product lines architectures," in *2012 34th International Conference on Software Engineering (ICSE), Doctoral Symposium*, june 2012, pp. 1507 –1510.

[24] C. L. Simons, "Interactive evolutionary computing in early lifecycle software engineering design," Ph.D. dissertation, University of the West of England, Bristol, UK, 2011.

[25] O. Räihä, K. Koskimies, and E. Mäkinen, "Empirical study on the effect of crossover in genetic software architecture synthesis," in *Proc. of NaBIC*. IEEE, 2009, pp. 619–625.

APÊNDICE C

ARTIGO SUBMETIDO - ICSE 2014

A Search-based Approach for Software Product Line Design

Thelma Elita Colanzi
State University of Maringa
Federal University of Parana
Curitiba, Brazil
thelma@din.uem.br

Silvia Regina Vergilio
Federal University of Parana
Curitiba, Brazil
silvia@inf.ufpr.br

Itana M. S. Gimenes,
Willian Nalepa Oizumi
State University of Maringa
Maringa, Brazil
{itana,ra51205}@din.uem.br

ABSTRACT

Product Line Architecture (PLA) is the main core asset of a Software Product Line (SPL). During the PLA design, SPL architects need to deal with different factors, such as modularizing features, avoiding architectural bad smells, manipulating several metrics values to improve the design. Search-based approaches allow treating PLA design as a multi-objective problem and can help architects to automatically find better designed PLAs. For this end, this paper introduces MOA4PLA, a search-based approach to PLA design. MOA4PLA encompasses a process, includes a feature-driven search operator to modularize features, and uses feature-driven and conventional metrics to evaluate the obtained PLA designs. An empirical study was carried out with five PLA designs. Results point out MOA4PLA provide a set of solutions with different trade-offs between the used metrics. Also, the feature-driven operator and metrics allow obtaining PLA design with well modularized features, contributing to improve features reusability and evolvability.

Categories and Subject Descriptors

Software and its engineering [**Software organization and properties**]: Software system structures—*Software architectures - Object oriented architectures*; Computing methodologies [**Artificial intelligence**]: [Search methodologies]; General and reference [**Cross-computing tools and techniques**]: [Design, Experimentation, Empirical studies]

General Terms

Design, Experimentation, Measurement

Keywords

Software product lines, multi-objective algorithms, search-based PLA design

ICSE '2014 Hyderabad, India

1. INTRODUCTION

A Software Product Line (SPL) consists of core assets that have explicit common and variable features [18]. A product of the SPL is given by a combination of its features. SPL Engineering has been widely adopted by software industry. Some successful case studies include Boeing, Philips, Bosch, Nokia and Toshiba [28].

Product Line Architecture (PLA) is the SPL main core asset. Its design includes components to realize common and variable features. SPL architects reason about programs in terms of features. Despite their crucial importance, features are rarely modularized. Modularizing features is difficult because feature-specific code often cuts across class boundaries. Even so, it is natural to consider modularizing features as a way to modularize programs [19]. Thus, a higher feature modularization implies in a lower functionality scattering in PLA design. Scattered Functionality [15] is one of the most relevant architectural bad smells because it negatively impacts on the PLA reusability and extensibility. In particular, these properties are important for PLA design.

Metrics help architects to identify design flaws and improve the design. Studies about PLA design evaluation often consider a set of metrics that provide indicators about different factors [2, 4]. Examples of these factors are design stability, feature modularization and PLA extensibility [14, 2, 25, 20]. Nevertheless, architects need to analyse a lot of metrics values. So, finding the best trade-off among the used metrics to configure the design turns PLA design into a people-intensive task. Furthermore, some metrics can be in conflict and architects need to choose which metric should be prioritized. In this context, to recognize a good design can be easy for architects, but it is difficult to obtain.

Search-based software engineering (SBSE) techniques have been proven to be effective in automatically discover near-optimal solutions to such kind of software engineering problems [1, 16, 22]. In this sense, SBSE techniques can provide automated support to evaluate the potential configurations for a PLA design. These techniques can also improve the design. The automated evaluation and improvement support the architect's decisions and contribute to reduce the design flaw propagation during the SPL development. However, there is not previous work that improves the PLA design based on SBSE techniques.

Most related studies focus on software design in general. Approaches based on Multi-Objective Algorithms (MOAs) have achieving promising results to solve the software design problem [22, 3, 24, 29]. Software design is considered a multi-objective problem since there are different factors

and measures that affect it. These factors are often in conflict and usually there is not a single solution for the problem. The idea is to find the solutions with better trade-off between the objectives. These solutions are named non-dominated and form the Pareto front [21]. One solution is better than another if it is better in one objective and it is not worse in any other one. There are some similarities between software design and PLA design. Both problems are affected by conflicting factors, and some of these factors are the same, for instance improving feature modularization often improves components cohesion but it increases components coupling. However, the PLA design presents some specific characteristics that are not considered by existing search-based design works.

Considering the context described above, this paper is concerned with how to provide computational support for SPL architects in order to achieve well-designed PLAs relying on MOAs. So, we proposed a systematic and automated approach that uses search-based algorithms to optimize a PLA design named MOA4PLA (Multi-Objective Approach for Product-Line Architecture Design). This approach produces a set of potential solutions with the best trade-off between different and conflicting objectives, such as the SPL extensibility, feature modularization and basic design principles like coupling and cohesion. MOA4PLA has four main characteristics: (a) it encompasses a process to conduct the PLA design optimization through search-based algorithms; (b) it includes a metamodel to allow the PLA representation suitable for search-based algorithms; (c) it addresses the feature modularization by the application of a novel search operator; and (d) it introduces a multi-objective treatment to the PLA design problem including feature-driven metrics. In MOA4PLA the PLA design is represented in the detailed level using UML class diagrams.

The objective of this work is twofold: (a) to introduce MOA4PLA and (b) to provide empirical evidence that the approach improve the design of PLAs. An empirical study applying MOA4PLA to five PLA designs was carried out. The obtained results were analyzed taking into account the main characteristics of MOA4PLA. The results point out that the goal of MOA4PLA is achieved by obtaining better PLA designs than the original ones in terms of feature modularization and stability. Also, some solutions have less architectural elements candidates to manifest some architectural bad smell. So, the main contribution of the work is to introduce an automated approach that reduces the effort to design PLAs and improves the PLA feature modularization, consequently enhancing features reusability and evolvability.

This paper is organized as follows: Section 2 addresses related work. Section 3 introduces MOA4PLA and describes its process, PLA representation, search operators, used metrics and implementation aspects. Section 4 defines the empirical study, research questions and presents its threats to validity. The results are presented in Section 5. Section 6 discusses some findings and point research oportunities. Section 7 presents conclusions and future work.

2. RELATED WORK

There are few studies that employ SBSE techniques to support SPL Engineering. These works focus usually on automated product configuration and feature selection for either SPL development or SPL evolution [26, 17]. So their optimization goal is not the PLA design.

The subject of the most related existing works is on search-based software design [22]. Some of them apply SBSE techniques to object-oriented design. These works focus on the architectural design outset by addressing two problems: the application of design patterns to software design [24] and the class responsibility assignment [3, 30]. Most of them use MOAs and coupling and cohesion metrics [3, 24, 30]. The search operators move methods and attributes from one class to another and also add new classes [3, 30].

Design transformations based on search operators are applied in some studies directly to software design models [30], such as class diagrams. Thus, the possible variants of the design form the search space. Some studies perform a genotype-phenotype mapping where search operators are applied to integer vectors [3] or supergene representations [24]. PLA representation at class diagram level was not used in related work. So, in a previous work [8] we analyzed how to adapt each representation used in the literature to the PLA design context. Based on this analysis, we conclude that the direct object-oriented representation adopted in [30] is more suitable than others. It is easier to extend it in order to represent variabilities and features. So, we proposed a novel representation (described in Section 3.1) that comprises SPL needs aiming at PLA design optimization.

Existing works successfully optimize software design. However, they have not been applied to optimize the PLA design. Also, they do not use search operators for feature modularization neither specific SPL metrics. Furthermore, the used search operators do not include specific characteristics of PLA. In this sense, it is necessary to define a novel optimization approach specific to SPL context and needs, which uses different and more appropriate quality metrics and operators. So, in the next section we introduce MOA4PLA, a search-based approach to optimize PLA design.

3. MOA4PLA

MOA4PLA is independent of both the method adopted for mapping features in the architectural elements and the method used to represent variabilities into PLA. To optimize a PLA design using MOA4PLA, the architect needs both to choose which metrics should be used in the evaluation model. He/she also needs to establish the constraints to be applied to the solutions. MOA4PLA encompasses a process to support the automated optimization of PLA design as depicted in Figure 1. MOA4PLA is a generic approach that may be instantiated by using different search algorithms and metrics. According to Harman, Mansouri and Zhang [16], two key ingredients are necessary to tackle a problem using search algorithms: 1) a representation of the problem: that allows symbolic manipulation; and 2) an objective function: defined in terms of the representation to evaluate the quality of the solutions. The first ingredient is generated in the activity named Construction of the PLA Representation. The second one is defined in the activity Definition of the Evaluation Model. Then, in the third activity search algorithms are applied to solve the problem whereas in the last activity, the output of MOA4PLA is presented to the architect. These activities are explained below.

Construction of the PLA Representation: the input for this activity is the PLA design modeled in a UML class diagram containing the SPL variabilities. In order to properly apply the feature-driven search operator (Section 3.2), it is expected that each architectural element of this diagram is

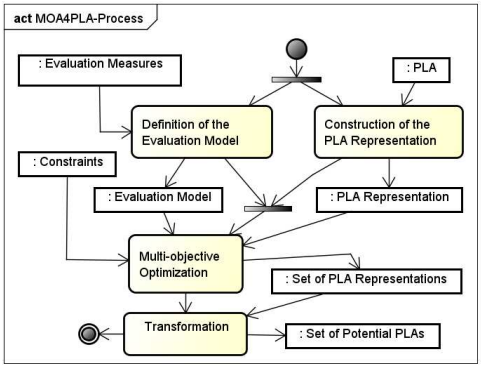


Figure 1: MOA4PLA Process

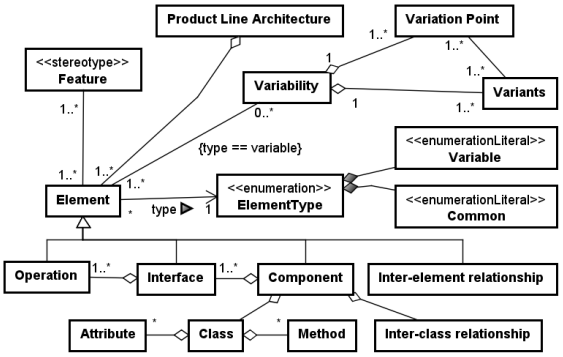


Figure 2: Metamodel of a PLA

associated to the feature(s) that it realizes. The output of this activity is a representation of the PLA design according to the metamodel presented in Section 3.1, containing architectural elements, their inter-relationships, the variabilities, variation points and variants.

Definition of the Evaluation Model: the metrics to be used in the objective function are selected. Conventional metrics to evaluate basic design principles can be used, such as coupling, cohesion and size. However, MOA4PLA also aims to optimize SPL specific factors such as feature modularization [2, 25, 14] and SPL extensibility [20]. A high extensibility level indicates that a greater number of products can be generated from the PLA, which leads to maximizing the market share. Given that a feature can be considered a concern in a SPL, concern-driven measures [25, 14] support the analysis of feature-based cohesion, feature scattering and interlacing between features in a PLA design. Component cohesion can also be measured by cohesion metrics specific for SPL [2]. These metrics give modularity indicators and modular PLAs are more stable, reusable and maintainable.

Multi-Objective Optimization: the PLA representation obtained in the first activity is optimized according to the constraints provided by the architect. Each obtained alternative PLA is evaluated following the evaluation model which contains the measures defined in the previous activity. A set of PLA representations is generated as output which are the solutions with the best trade-off between the objectives. Different search-based algorithms can be used in this activity, such as Multi-Objective Evolutionary Algorithms (MOEAs), Ant Colony and Particle Swarm Optimization.

Transformation: each PLA representation, from the set obtained in the Multi-Objective Optimization, is converted to a class diagram containing the PLA design. So, the architect must select one PLA design to be adopted according to the his/her priorities or goals.

Following sections present the PLA representation, the search operators used to improve the design, the metrics that can be used in the objective function and some implementation aspects for MOA4PLA.

3.1 PLA Representation

The PLA is represented by the metamodel shown in Figure 2. This metamodel is an extension of that one proposed in [4]. It is represented by a class diagram which contains the elements of a PLA associated with the respective description of variabilities. This allows the handling of the architectural elements by the search operators.

A PLA contains architectural elements such as components, interfaces, operations and their inter-relationships. Each architectural element is associated with feature(s) by using UML stereotypes. Each element can be common to all SPL products or variable, being present only in some product(s). Variable elements are associated to variabilities that have variation points and their variants.

Figure 3 depicts an example of a PLA for the SPL Arcade Game Maker (AGM) [27] represented by this metamodel. In this PLA design there are five variabilities detailed in UML notes attached to the classes that realize the variabilities. Variation points, alternative variants, optional and mandatory features are shown by the stereotypes $variationPoint, alternative\_OR, optional$ and $mandatory$, respectively. Features, such as configuration, ranking, movement, play, save, are also shown through stereotypes. Each architectural component was represented by a package of classes that compound it. Due to lack of space, attributes and operations were omitted.

3.2 Search operators

Specific search operators are necessary when a direct representation is used, such as the representation adopted in MOA4PLA. Search operators for software design can be used in MOA4PLA. They are: MoveMethod [3, 30], MoveAttribute [3, 30], AddClass [3, 30], MoveOperation [7] and AddComponent [7]. AddClass moves a method or an attribute to a new class. The addition of classes is necessary since optimal class assignments may require additional classes that were not identified in the design outset. The class that receives the moved attribute/method becomes client of the original class of the element. MoveOperation moves operations between interfaces. AddComponent creates a new interface to a new component, and then moves an operation to this interface. When an operation is moved, the component that receives the moved operation becomes client of the original interface of the operation.

In addition, we have introduced a search operator driven to PLA feature modularization named Feature-driven Operator. This operator aims at modularizing a feature tangled with others in a component, considering that a feature usually is solved by a group of architectural elements. Algorithm 1 presents the pseudocode of the feature-driven operator. It receives as input a PLA design A and returns its improved design as output. It selects an arbitrary component (c_x), and, if c_x have architectural elements (eg. interfaces, classes, operations, methods and attributes) associated with

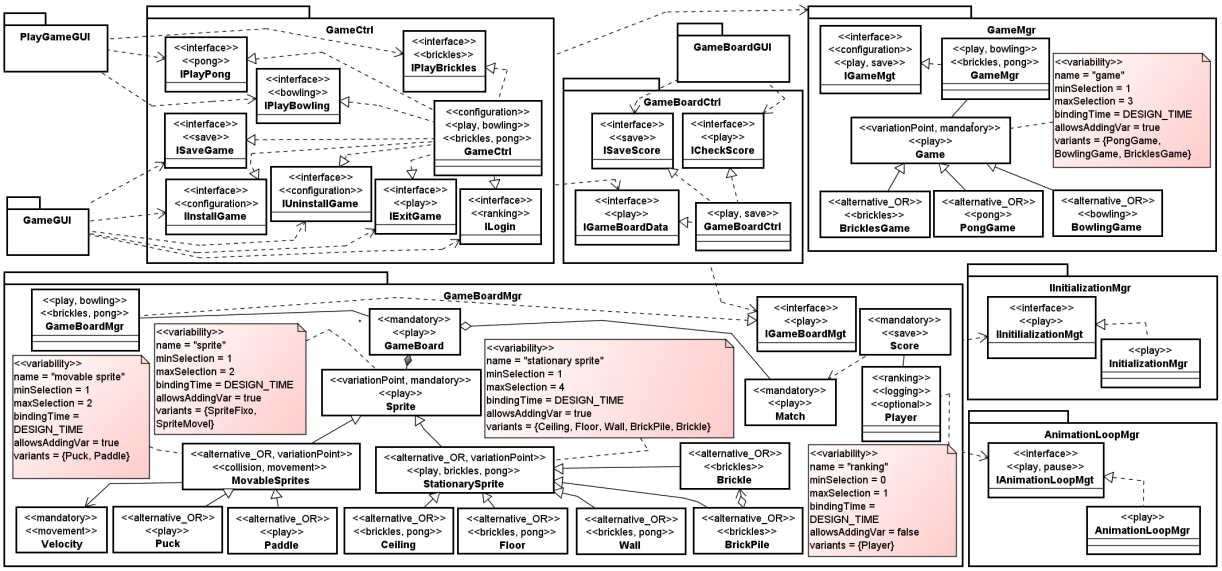


Figure 3: Original PLA AGM-v2

different features, an arbitrary feature (f_x) is selected to be modularized to the component c_z. c_z is chosen according to the size of the set c_f which contains all components of A associated with f_x. When the size is equal to 0, a new component c_z is created. When the size is 1, c_z receives the single component of c_f. Or, if the size is higher than 1, c_z receives a randomly selected component of c_f. The architectural elements of c_x associated with f_x are moved to c_z. c_z becomes client of the original components from the moved architectural elements.

Algorithm 1: Pseudocode of the Proposed Operator

```
begin
    C ← A.getAllComponents()
    E ← set of architectural elements of A
    c_x ← a randomly selected component from C
    F_c ← c_x.getAllFeatures()
    if F_c.size()>1 then
        f_x ← a randomly selected feature from F
        c_f ← A.getAllComponentsAssociatedWithFeature(f_x)
        if c_f.size==0 then
            c_z ← A.createComponent()
            c_z.addFeature(f_x)
        else
            if c_f.size==1 then
                c_z ← c_f.get(0)
            else
                c_z ← a randomly selected component from c_f
        for each e ∈ E | (e ≠ c_z or e ∉ c_z) do
            if e.getFeatures.size()==1 and
            e.isAssociatedWithFeature(f_x) then
                Move e to c_z
                A.addRelationship (c_z, original component of e)
        end
    return A;
end
```

The application of this operator helps to obtain PLA designs which have less scattered and less tangled features, thus, it improves the feature cohesion of the architectural components. In this way, the operator also contributes to avoid the presence of the architectural bad smell Scattered Functionality [15].

The application of the feature-driven operator may be illustrated in the PLA design of Figure 3. The operations were not shown, however, the feature configuration is spread over operations of five system interfaces where it is tangled with other four features. Nevertheless, two of those interfaces are associated only with the feature configuration: *IInstallGame* and *IUninstallGame*. By the application of the operator, operations associated with this feature would be modularized into one of these interfaces.

3.3 Evaluation Model

The evaluation model takes into account conventional and specific metrics according to the choice of the architect. Metrics to evaluate basic design principles are called here Conventional Metrics (CM). Table 1 presents some existing CM that can be used in MOA4PLA. The cohesion measure H indicates to the designers the components which do not have strongly-related elements. This means that such components cohesion are weak. The size metric NumOps reveals reusability aspects of the PLA because small interfaces are, in general, easier to reuse. The other metrics refer to coupling between architectural elements.

The Feature-driven Metrics (FM) [14, 25] are used to evaluate the changes caused by the feature-driven operator. They evaluate the degree of modularization of an architecture in terms of the features being realized. In addition to the negative impact on PLA reusability and maintainability, inaccurate feature modularization can lead to a wide range of design flaws, ranging from feature tangling and scattering to specific code smells, such as feature envy or god class [13].

Table 2 presents the FM suite that enables MOAs to evaluate the feature modularization in PLA design evolved by the proposed search operator. The metrics CDAC, CDAI and CDAO consider that a feature scattered on a high number of elements has negative impact on modularity. The feature interaction, measured by the metrics CIBC, OOBC and IIBC, happens by the presence of different features in a same architectural element. This interaction happens in three levels: component, interface and operation. Lack of feature-based cohesion (LCC) indicates that a component that addresses many features is not stable as a modification in any of the associated features may impact the others.

Table 1: Conventional Metrics Suite (CM) [32]

Attribute	Metric	Definition
Cohesion	Relational Cohesion (H)	Average number of internal relationships per class in a component.
Coupling	Dependency of Packages (DepPack)	Number of packages on which classes and interfaces of this component depend.
	ClassDependencyIn (CDepIn)	Number of elements that depend on this class.
	ClassDependencyOut (CDepOut)	Number of elements on which this class depends.
	DependencyIn (DepIn)	Number of UML dependencies where the package is the supplier.
	DependencyOut (DepOut)	Number of UML dependencies where the package is the client.
Size	Number of Operations by Interface (NumOps)	Number of operations in the interface.

Table 2: Feature-driven Metrics Suite (FM) [25, 14]

Attribute	Metric	Definition
Feature Scattering	Feature Diffusion over Architectural Components(CDAC)	Number of architectural components which contributes to the realization of a certain feature
	Feature Diffusion over Architectural Interfaces(CDAI)	Number of interfaces in the system architecture which contributes to the realization of a certain feature
	Feature Diffusion over Architectural Operations(CDAO)	Number of operations in the system architecture which contributes to the realization of a certain feature
Feature Interaction	Component-level Interlacing Between Features(CIBC)	Number of features with which the assessed feature share at least a component
	Interface-level Interlacing Between Features(IIBC)	Number of features with which the assessed feature share at least an interface
	Operation-level Overlapping Between Features(OOBC)	Number of features with which the assessed feature share at least an operation
Feature-based Cohesion	Lack of Feature-based Cohesion(LCC)	Number of features addressed by the assessed component

The evaluation model is extensible. So, other metrics can be used. They are not present here due to space limitation.

3.4 Implementation Aspects

The OPLA-Tool is an extension of the framework jMetal [12] implemented to validate MOA4PLA. Figure 4 depicts the main classes created by OPLA-Tool (highlighted in gray). OPLA-Tool adds a specialization to jMetal problem, named $OPLA$. $OPLA$ uses $conventionalMetrics$ and $feature\text{-}drivenMetrics$ in the optimization of variables of the type $Architecture$. The class $ArchitectureSolutionType$ was added to facilitate the manipulation of architectures. The input to the OPLA-Tool is an XMI file containing a PLA design modeled in a UML class diagram. The PLA representation is instantiated to an object of the class $Architecture$ from this XMI file.

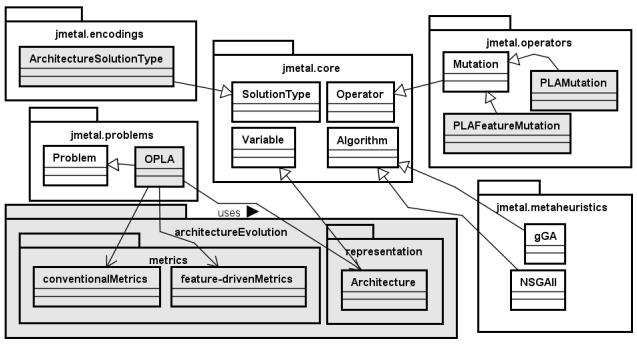


Figure 4: jMetal and OPLA-Tool class diagram

As mentioned before, different search algorithms can be used to implement the multi-objective optimization of MOA4PLA. Following related work [3, 24, 30], we decide to start the evaluation of MOA4PLA using evolutionary algorithms. MOEAs are particularly suitable for the PLA design problem since they evolve simultaneously a population of potential solutions to the problem obtaining a set of solutions to approximate the Pareto front in a single run of the algorithm.

The search operators were implemented as mutation operators. $PLAMutation$ and $PLAFeatureMutation$ (see Figure 4) are mutation search operators specific for PLA design. $PLAMutation$ contains the operators used in related work [3, 30, 7], called conventional operators. $PLAFeatureMutation$ implements the Feature-driven proposed operator (Section 3.2). Experiments with MOA4PLA need to use the OPLA problem in addition to either $PLAMutation$ or $PLAFeatureMutation$.

The search process applies some constraints to ensure that each generated solution is consistent as follow: (i) classes and their subclasses cannot be separated; (ii) the moved element must maintain the original associated feature even if it was moved to an element associated to a different feature; and, (iii) components, interfaces and classes cannot be empty. The two first constraints were implemented into the mutation operators. We decided to repair the solutions that do not satisfy the last constraint instead of excluding them from the population. By reparing a solution we maintain both satisfactory potential designs and the population diversity. At this moment, the implementation of the mutation operators does not apply changes on classes that participate in generalizations.

4. EMPIRICAL STUDY DEFINITION

We have undertaken an empirical study to evaluate the main characteristics of MOA4PLA. The study involved five PLA designs. It aimed to answer three research questions as follow:

RQ1: Is the multi-objective treatment to the PLA design problem given by MOA4PLA adequate? This question aims to evaluate the multi-objective treatment given by MOA4PLA to the referred problem compared to a single objective one in terms of the solutions found and the metrics values.

RQ2: Is the feature-driven search operator effective to modularize features? To answer this question, we performed a qualitative analysis of the solutions obtained by the algorithms using the proposed feature-driven search operator against the solutions obtained by the same algorithms using only the conventional search operators.

RQ3: Are the metrics used in the evaluation model appropriate to evaluate the PLA designs used in the study? It is also important to analyse if the applied met-

Table 3: Fitness Function

$FM(pla) =$	$\sum \sum_{i=1}^{c} LCC \sum_{i=1}^{f} CDAC \sum_{i=1}^{f} CDAI \sum_{i=1}^{f} CDAO \sum_{i=1}^{f} CIBC \sum_{i=1}^{f} IIBC \sum_{i=1}^{f} OOBC$
$CM(pla) =$	$\sum \sum_{i=1}^{c} DepIn \sum_{i=1}^{c} DepOut \sum_{i=1}^{cl} CDepIn \sum_{i=1}^{cl} CDepOut \frac{\sum_{i=1}^{c} DepPack}{c} \frac{\sum_{i=1}^{itf} NumOps}{itf} \frac{1}{\sum_{i=1}^{c} H}$
where	c is the number of components, itf is the number of interfaces, cl is the number of classes and f is the number of features of a design pla

rics are sensitive to evaluate the changes applied by search operators in the context of the used PLAs.

Aiming at supporting the analysis to answer these questions, we used the quality indicator the Euclidean Distance from an Ideal Solution (ED). It is used to find the closest solutions to the best objectives. It is based on Compromise Programming [5], a technique used to support decision making when a set of good solutions is available. An ideal solution has the minimum value of each objective of the Pareto front, considering a minimization problem. So, the achieved solution that has the lowest ED has the best trade-off between the objectives.

In addition, we decided to calculate the percentages of improving FM for each obtained solution. In this way, a fitness quantitative analysis of the solutions with the best trade-off, the best percentage and the worst percentage obtained by mono and multi-objective algorithms provide information to answer **RQ1**. A qualitative and ED quantitative analysis support answering **RQ2**. Finally, the answer of **RQ3** can be based on a qualitative analysis of the sensitiveness of metrics to evaluate the fitness of the solutions with the best and the worst percentages of FM improvement.

The next sections describe the fitness functions and the algorithms used in the study. Also, Section 4.3 presents the used PLAs and Section 4.4 discusses threats to validity.

4.1 Fitness Function

To evaluate the solutions, we used two functions, $CM(pla)$ and $FM(pla)$, which are respectively aggregations of the CM of Table 1 and of FM of Table 2. These functions are calculated according to Table 3. These two objectives aim at analysing if the obtained solutions (potential PLA designs) are likely to entail proper designs from the point of view of high cohesion, low coupling, reusability and feature-based modularity.

$FM(pla)$ consists on the sum of the sum of each feature-driven metric. FM were used to evaluate the feature modularization caused by the feature-driven operator. As they are co-related, when the value of one metric decreases, the values of the others decrease as well. $CM(pla)$ is compound by the sum of several CM. In this sum we are using the mean of DepPack for all components and the mean of NumOps for all interfaces. The metric H was counted inversely in the function because we are interested in maximizing the cohesion and minimize all other metrics.

4.2 Used search algorithms

We have implemented the most used MOEA NSGA-II (Non-dominated Sorting Genetic Algorithm) [10]. It is based on Genetic Algorithms (GA) with a strong elitism strategy. For each generation NSGA-II sorts the individuals, from parent and offspring populations, considering the non-dominance relation, creating several fronts. After the sorting, solutions with lower dominance are discarded. These fronts characterize the elitism strategy adopted by NSGA-II. This algorithm also uses a diversity operator (crowding distance) that sorts the individuals according to their distance from the neighbors of the border for each objective, in order to ensure greater spread of solutions.

Furthermore, in order to answer **RQ1** we also have implemented three alternative GAs [6]. As GAs are mono-objective and we are using two objectives, we decided to adopt an aggregation fitness function obtained through the weighted sum of CM and FM. We evaluated three different combinations of weights. To verify the empirical influence of each kind of metric in the PLA design we used a configuration to minimize only CM (identified here as GAC (GA with Conventional metrics)). In this configuration the weight of the FM was set with zero. The other configuration minimizes only FM (identified here as GAF (GA with Feature-driven metrics)). In this configuration the weight of the CM was set with zero. In the third configuration (GA), equal importance was given to both kind of metrics.

The parameters set up for NSGA-II and GAs are presented in Table 4. Some parameters were adjusted from values adopted in related work, such as the population size [24, 29] and the number of generations [29]. The mutation rate was empirically adjusted using NSGA-II with the conventional mutation. Four configurations were tested: {0.5, 0.7, 0.8, 1.0}. Solutions achieved with the rate 0.8 outperformed all solutions found by the other configurations.

Table 4: NSGA-IIs and GAs Parameters

Parameter	GAC	GAF	GA	NSGA-II
Population Size	100	100	100	100
Mutation Rate	0.8	0.8	0.8	0.8
Number of Generations	300	300	300	300
Number of Fitness Evaluations	30000	30000	30000	30000
Conventional Metrics Weight	1	0	0.5	-
Feature-driven Metrics Weight	0	1	0.5	-

Empirical studies [23, 31] observed that the crossover operator does not offer significant advantage in the search-based design when implemented in a random fashion. So, in this study, traditional crossover was not applied. We intend to propose a crossover operator specific for the representation of PLA design used in MOA4PLA in further work.

Aiming at answering **RQ2** we have also implemented GAs and NSGA-II using both only the conventional search operators and the conventional ones plus the feature-driven operator. So, the experiments named GA, GAC, GAF and NSGA-II have used only conventional operators. In the experiments named GA-FM, GAC-FM, GAF-FM and NSGA-II-FM the feature-driven mutation operator was added using the same parameters mentioned in Table 4.

Each experiment was executed 30 times for each PLA in order to know the behavior of each algorithm to solve the problem. The number of fitness evaluations was used as stop criterion for all algorithms. To analyze the results of the empirical study three different sets of solutions have been used. The first one is PF_{approx}. One set PF_{approx} for a PLA is obtained in each run of an algorithm. So, after 30 runs, 30 sets PF_{approx} were obtained. For GAs, PF_{approx} has only one

Table 5: Characteristics of the PLAs

PLA	Original Fitness (FM, CM)	#Components	#Interfaces	#Classes	#Variabilities	#Features
AGM-v1	(391.0, 31.82)	9	14	20	4	9
AGM-v2	(499.0, 35.18)	9	14	21	5	11
MM-v1	(608.0, 39.31)	11	16	13	7	13
MM-v2	(534.0, 33.48)	8	13	10	7	13
LPS-BET	(674.0, 176.84)	56	33	115	10	16

solution. The second set is PF_{known}. This set was obtained for each PLA through the union of the 30 sets PF_{approx}, removing dominated and repeated solutions. PF_{known} represents the best solutions found by each algorithm. The last one is PF_{true}. Since the actual PF_{true} is not known, in our study this set was obtained for each PLA through the union of the sets PF_{known}, removing dominated and repeated solutions. PF_{true} represents the best solutions known to the problem. This strategy to obtain the best solutions to a problem is recommended when the ideal set of solutions is not known [33].

4.3 PLAs used in the empirical study

Table 5 contains some information about the PLAs used in the empirical study, such as the number of components, classes, features, variabilities, etc. AGM [27] is a SPL created by the Software Engineering Institute (SEI) that encompasses three arcade games: Brickles, Bowling, and Pong. AGM-v1 was developed in [9] and AGM-v2 was obtained from AGM-v1 by adding two new features: ranking and logging. These features aims at maintaining information about both the best ranking for each game available in a AGM product and the users that achieve the rankings.

Mobile Media (MM) [9] is a SPL composed of features that handle music, videos, and photo for portable devices. It provides support for managing different types of media. MM-v1 was developed in [9] and MM-v2 is equivalent to the release R8 presented in [14]. MM-v1 and MM-v2 are equivalent in terms of features but MM-v2 has lower number of architectural elements. The main difference between them is the component MediaMgr that is less coupled and more cohesive. Furthermore, three features are less scattered improving the feature-based cohesion of the design.

LPS-BET [11] supports the bus city transport management. It offers features such as the use of an electronic card for transport payment; automatic toll gate opening; and unified travelling payment. It was conceived based on three existing transportation products of Brazilian cities.

These five PLA designs are component-based and follow the layered architectural style including manager components that provide interfaces to enable the communication between layers (GUI, system and business layers).

4.4 Threats to Validity

The main issue that we take into account as a risk to affect our study's conclusion validity is the size of the SPLs used in the sample (four PLAs are small). However, they allow us to realize that the goals of MOA4PLA could be achieved even for small SPLs. So, with greater SPLs the results could be observed as well. Furthermore, the sample can be increased during prospective replications of this study.

The construct validity regarding to the used metrics is guaranteed by their previous successful application in other studies, such as [9, 14, 25]. The features mapping to architectural elements influences on the feature modularization of the obtained solutions. We adopted the mapping performed by original designers of each PLA. Although different mappings could lead to different results, the benefits of MOA4PLA to improve feature modularization should not suffer the effects of such mapping. Finally, AGM, MM and LPS-BET are non-commercial SPL. In spite of this, these cases enable us to observed advantages and drawbacks of the proposed approach. Those observations can be analyzed in future studies with similar goals involving commercial SPLs.

5. EMPIRICAL STUDY RESULTS

Tables 6 and 7 present the fitness of the solutions found by GAs and NSGA-IIs respectively, following the format (FM, CM). The columns named Runtime also present the mean runtime (in seconds) used to obtain each PF_{approx} and the standard deviation (between parentheses). Runtime for each run of all algorithms is similar.

For GAs (Table 6), only one solution (the best) was returned after the 30 runs. On the other hand, for NSGA-II a set of non-dominated solutions was obtained. In this sense, the second column of Table 7 presents the cardinality of PF_{true}. The third and sixth columns present the cardinality of PF_{known} and, between parentheses, the number of solutions from PF_{known} that are included in PF_{true}. In both tables, solutions highlighted in bold form PF_{true}. The multi-objective algorithms NSGA-II and NSGA-II-FM achieved greater number of solutions in PF_{true} than the mono-objective GAs.

Considering the PLAs used in the study, a set of 109 solutions were found by the algorithms. From this set, only 4 solutions (3.6%) achieved higher (worse) values for one fitness function. So, 96% of the solutions achieved better values than the original fitness. This fact shows the effectiveness of MOA4PLA for achieving better PLA design than that ones supplied as input to it. The solution found by GAFFM and one solution found by NSGA-II-FM (476.0, 177.43) were two exceptions for LPS-BET. In both cases, the lowest values were achieved for FM compromising CM. Both solutions are dominated by solutions found by other algorithms. Thus, they are not in PF_{true}. Based on this results, the research questions are answered in the next sections.

5.1 Multi-objective treatment to PLA design

To answer **RQ1** it is interesting to discuss if the metrics used are in conflict justifying the use of MOEAs to solve the problem. For this end, we take the percentages of improvement in FM achieved by NSGA-IIs, presented in Table 8. The values in bold are the greatest percentages for FM and CM. The solutions found by NSGA-II and NSGA-II-FM with the best FM for AGM-v2, MM-v1 and LPS-BET achieved the worst value for CM. The same happens for the solution of MM-v1 with the worst FM: it has the lowest CM value. This fact attest the metrics used are in conflict.

Another point to be considered is the comparison between solutions found by GAs and NSGA-IIs (mono versus multi-

Table 6: Results of GAs

PLA	GA		GA-FM		GAC		GAC-FM		GAF		GAF-FM	
	Fitness	Runtime	Fitness	Runtime	Fitness	Runtime	Fitness	Runtime	Fitness	Runtime	Fitness	Runtime
AGM-v1	(186.0, 23.25)	115.24 (2.27)	(184.0, 21.72)	108.21 (1.89)	(491.0, 17.34)	103.88 (1.02)	(438.0, 17.34)	113.75 (10.07)	(184.0, 25.0)	129.14 (11.83)	(182.0, 28.16)	103.44 (2.64)
AGM-v2	**(237.0, 28.62)**	123.17 (3.67)	(225.0, 29.48)	120.10 (2.51)	(467.0, 20.75)	113.01 (0.33)	(476.0, 20.57)	117.08 (0.82)	**(236.0, 42.51)**	132.62 (3.75)	**(231.0, 32.64)**	118.83 (2.49)
MM-v1	(306.0, 31.11)	125.29 (3.47)	(225.0, 27.16)	113.75 (4.18)	(588.0, 29.17)	112.90 (0.28)	(540.0, 27.8)	124.60 (11.42)	(307.0, 31.11)	137.18 (9.91)	(221.0, 32.13)	116.81 (4.86)
MM-v2	(229.0, 26.83)	119.12 (3.07)	**(178.0, 20.76)**	106.19 (3.89)	(492.0, 23.52)	104.05 (0.26)	(443.0, 21.77)	109.84 (1.43)	(226.0, 33.83)	129.30 (4.27)	(181.0, 21.26)	107.66 (3.75)
LPS-BET	(629.0, 160.6)	578.07 (5.33)	**(475.0, 174.4)**	557.84 (12.11)	**(671.0, 147.1)**	589.76 (8.86)	**(660.0, 148.1)**	573.56 (11.53)	(625.0, 171.4)	595.38 (6.99)	(477.0, 208.0)	549.07 (14.71)

Table 7: Results of NSGA-II

PLA	# PF_{true}	NSGA-II			NSGA-II-FM		
		Number of Solutions	Runtime	Fitness	Number of Solutions	Runtime	Fitness
AGM-v1	8	7 (6)	103.27 (1.18)	**(298.0, 17.34)** (201.0, 21.47) **(261.0, 17.66) (247.0, 18.34) (212.0, 18.66) (209.0, 20.66) (267.0, 17.47)**	4 (2)	106.84 (1.59)	**(182.0, 20.75) (181.0, 21.25)** (332.0, 19.25) (304.0, 19.5)
AGM-v2	12	7 (4)	112.26 (1.31)	**(329.0, 18.62) (314.0, 19.62) (326.0, 19.57)** (276.0, 22.76) (301.0, 20.57) (277.0, 21.76) **(303.0, 19.76)**	7 (5)	118.94 (1.81)	**(244.0, 24.09) (253.0, 23.50)** (459.0, 19.72) **(262.0, 20.57) (261.0, 22.57)** (239.0, 29.50) **(243.0, 28.50)**
MM-v1	4	3 (3)	119.08 (1.56)	**(347.0, 24.9) (332.0, 25.11) (326.0, 26.11)**	2 (1)	125.73 (3.18)	**(219.0, 26.20)** (437.0, 26.08)
MM-v2	6	5 (0)	110.37 (1.9)	(236.0, 25.83) (249.0, 20.67) (238.0, 21.83) (246.0, 21.67) (339.0, 20.16)	6 (5)	112.58 (1.14)	**(276.0, 19.13) (204.0, 20.43) (258.0, 19.25) (243.0, 19.92) (236.0, 20.22)** (178.0, 21.29)
LPS-BET	31	8 (0)	580.47 (6.64)	(647.0, 154.8) (635.0, 159.6) (639.0, 157.6) (640.0, 156.8) (636.0, 158.6) (642.0, 156.6) (643.0, 155.8) (637.0, 157.8)	30 (28)	563.72 (9.61)	**(609.0, 154.1)(574.0, 156.1)(615.0, 153.1)(612.0, 153.4) (595.0, 155.1)(546.0, 157.4)(596.0, 154.4)(487.0, 168.2) (490.0, 166.4)(479.0, 173.8)**(476.0, 177.4) **(543.0, 158.8) (541.0, 159.6)(493.0, 164.6)(480.0, 173.8)(538.0, 159.8) (537.0, 160.6)(520.0, 160.8)(519.0, 161.6)(512.0, 161.8) (511.0, 162.6)586.0, 155.1)(553.0, 156.4)(483.0, 171.8) (542.0, 159.1) (551.0, 157.1)**(477.0, 174.4) **(498.0, 162.8)(482.0, 172.6)(481.0, 173.4)**

Table 8: Percentages

PLA	Original PLA	NSGA-II			NSGA-II-FM		
	(FM, CM)	The best FM	The best trade-off	The worst FM	The best FM	The best trade-off	The worst FM
AGM-v1	(391.0, 31.82)	(201.0, 21.47) 48.49%, 32.52%	(201.0, 21.47) 48.59%, 32.52%	(298.0, 17.34) 23.78%, **45.50**%	(181.0, 21.25) **53.70**%, 33.21%	(182.0, 20.75) 53.45%, 34.78%	(332.0, 19.25) 15.08%, 39.50%
AGM-v2	(499.0, 35.18)	(276.0, 22.76) 44.68%, 35.30%	(277.0, 21.76) 44.48%, 38.14%	(329.0, 18.62) 36.04%, **47,07**%	(239.0, 29.50) **52,10**%, 16.14%	(244.0, 24.09) 51.10%, 31.52%	(459.0, 19.72) 8.01%, 43.94%
MM-v1	(608.0, 39.31)	(326.0, 26.11) 46.38%, 33.57%	(326.0, 26.11) 46.38%, 33.57%	(347.0, 24.9) 42.92%, 33.57%	(219.0, 26.20) **63,98**%, 33.35%	(219.0, 26.20) 63.98%, 33.35%	(437.0, 26.08) 28.12%, **33.65**%
MM-v2	(534.0, 33.48)	(236.0, 25.83) 55.80%, 22.84%	(236.0, 25.83) 55.80%, 28.82%	(339.0, 20.16) 36,51%, 39.78%	(178.0, 21.29) **66.66**%, 36.40%	(178.0, 21.29) 66.66%, 36.40%	(276.0, 19.13) 48.31%, **42.86**%
LPS-BET	(674.0, 176.84)	(635.0, 159.65) 5.78%, 9.72%	(635.0, 159.65) 5.78%, 9.72%	(647.0, 154.88) 4.0%, 12.41%	(476.0, 177.43) **29.37**%, -0.33%	(490.0, 166.43) 27.20%, 5.88%	(615.0, 153.14) 8.75%, **13.40**%

objective algorithms). For AGM-v1 and MM-v1, all solutions achieved by GAs are dominated by solutions of NSGA-IIs. So, MOEAs achieve better solutions for the problem.

Regarding to LPS-BET, the solutions obtained by GA-FM, GAC and GAC-FM are non-dominated. But, they are far from the ideal solution. Tables 9 and 10 present the lowest ED achieved by solutions of GAs and NSGA-IIs. Table 10 also presents the ideal solution for each PLA. For LPS-BET, the solution with the lowest ED was obtained by NSGA-II-FM. Considering the solutions with the lowest values for FM from GAs and NSGA-IIs we have: from GAF-FM the solution with the fitness (477.0, 208.03) and from NSGA-II-FM the solution (476.0, 177.43). From this point of view, the multi-objective algorithm is the best again.

From this point of view, the solutions obtained for AGM-v2 and MM-v2 with the lowest FM values are presented in Table 11. For AGM-v2, solutions found by NSGA-II-FM have lower ED than solutions of GAF-FM and GA-FM. For MM-v2, GA-FM achieved the solution with the lowest ED. In spite of this, the solution of NSGA-II-FM is similar.

In general, it is possible to state that GA obtained better results than GAF and GAC. On regards to the employment of MOEAs to solve the problem, NSGA-II achieved better results than GA. Furthermore, a multi-objective algorithm generates, in a single run, a set of solutions with different trade-offs between the metrics, including the lowest ED. To obtain a similar set of solutions by using GA, it would be

Table 9: Euclidean Distances for GAs' solutions

PLA	GA	GA-FM	GAC	GAC-FM	GAF	GAF-FM
AGM-v1	7.74	5.31	310.00	257.00	8.22	10.87
AGM-v2	15.62	10.86	242.00	251.00	26.30	15.25
MM-v1	87.22	6.41	369.02	321.01	88.21	7.51
MM-v2	51.57	**1.62**	314.03	265.01	50.19	3.67
LPS-BET	154.59	27.28	196.0	185.00	151.95	60.91

Table 10: NSGA-II: the lowest Euclidean Distances

PLA	Ideal Solution	NSGA-II		NSGA-II-FM	
		ED	Fitness	ED	Fitness
AGM-v1	(181.0,17.34)	20.4	(201.0,21.47)	**3.5**	**(182.0,20.75)**
AGM-v2	(239.0,18.62)	38.1	(277.0,21.76)	**7.4**	**(244.0,24.09)**
MM-v1	(219.0,24.9)	107.0	(326.0,26.11)	**1.3**	**(219.0,26.20)**
MM-v2	(178.0,19.13)	58.3	(236.0,25.83)	2.1	(178.0,21.29)
LPS-BET	(475.0,147.14)	160.4	(635.0,159.65)	**24.2**	**(490.0,166.43)**

Table 11: Solutions with the lowest values for FM

PLA	GAF-FM/ED	GA-FM/ED	NSGAII-FM/ED
AGM-v2	(231.0, 32.6)/15.2	(225.0, 29.4)/10.8	(244.0, 24.0)/7.4
MM-v2	(181.0, 21.2)/3.6	(178.0, 20.7)/1.6	(178.0, 21.2)/2.1

necessary a lot of runs with different combinations of metrics weights. If we consider that the runtimes of GAs and NSGA-II are very similar, the computational cost to obtain a set of solutions with GA is huge (runtime * number of runs) in comparison with the NSGA-II runtime. So, multi-objective algorithms are adequate to solve the PLA design problem in the context of MOA4PLA.

5.2 Feature-driven search operator

In this section we analyse the algorithms performance using the feature-driven operator (**RQ2**). The algorithms that employ the proposed operator achieved the best results in terms of ED for almost all PLAs than their respective versions using only the conventional operators (Tables 9 and 10). There were only two cases in which GAF and GAC obtained a solution with lower ED than GAF-FM and GAC-FM: AGM-v1 and AGM-v2, respectively.

If we consider which algorithm achieved the solution with the lowest ED for each PLA, we can point NSGA-II-FM as the best for AGM-v1, AGM-v2, MM-v1 and LPS-BET. GA-FM was the best for MM-v2. So, in all cases, the proposed operator allows the algorithms to achieve PLAs with better feature modularization.

Figure 5 illustrates this point by showing some elements encompassed by the PLA design found for AGM-v2 with the best value for FM. The architectural elements that realize the features ranking and logging are presented in this picture. The complete PLA design is shown in a high level view in Figure 6. In the original PLA, ranking was tangled with 3 other features (logging, save and play) and it was scattered through 5 classes and interfaces from different packages. In the achieved PLA design (Figure 5), ranking is tangled only with logging. Ranking is modularized by $Component303554$ and realized by $ILogin$ and $Class71341$. Regarding to logging, it was associated together with ranking in two operations, so, it is still tangled with such feature. In spite of this, it is possible to see in Figure 5 that logging is now modularized in the class $Player$ and there are two operations associated with logging in $Interface243570$. Such interface and $Component302639$ are not associated with any feature. We infer that they were created by the conventional mutation operator AddComponent whose is not concerned about the feature assignment. However, that feature was not completely modularized into only one component, since it is associated with both a class of $GameBoardMgr$ ($Player$) and the referred interface of $Component302639$.

An analysis of the percentage of improvement in FM corroborates the evidence of the feature-driven operator allows better feature modularization. For instance, the solutions with the highest percentage (Table 8) were found by NSGA-II-FM (column The best FM) for all PLAs. Among the solutions with the best trade-off, NSGA-II-FM also provided higher percentages than NSGA-II. An interesting point to be highlighted is that even for solutions with the worst value for FM, the highest percentages of improvement in CM were achieved by NSGA-II-FM for MM-v1, MM-v2 and LPS-BET. This shows that, beyond its original purpose that is to modularize features, the feature-driven operator also provides PLAs with better stability values according CM.

5.3 Metrics used in the evaluation model

In **RQ3** we analyse if the metrics used in the evaluation model could evaluate the obtained PLA designs as expected.

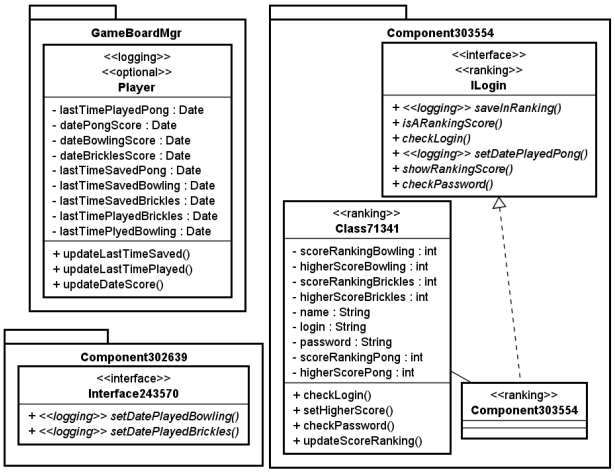


Figure 5: Features ranking and logging

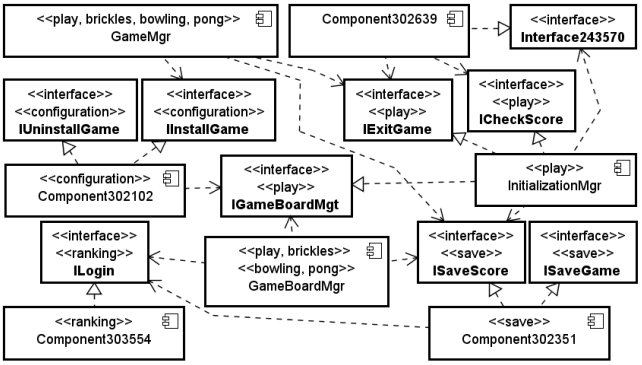


Figure 6: PLA with the best FM for AGM-v2

On regards to the FM, an analysis considering all PLAs points out that FM was sensitive to feature modularization. When we compare the PLA designs with the best values for FM against the designs with the worst FM, the formers have components with higher feature-based cohesion and greater number of modularized features, leading to lower numbers of tangled and scattered features. For instance, in the obtained solution with the worst FM for AGM-v2, logging is tangled with configuration, save, ranking and play whereas in the PLA of Figure 5 it was tangled only with ranking. Even in components associated with only one feature, operations in several interfaces are associated with other features. When considering the solution with the lowest ED, the number of features scattered is lower than in the worst FM and higher than in the best FM as expected. Similar situations happened for solutions with the best trade-off, the best FM and the worst FM for all other SPLs.

Another point to be considered is the influence of the conventional operators on the obtained solutions. As mentioned before, it was possible to realize the effects of the operator AddComponent. The effects of the operator MoveOperation were also perceived due to the presence of operations associated with some features in interfaces associated with other features. The higher the number of such situations in the design, the lower the value of FM. So, it is possible to state that FM were appropriate to measure feature modularization in MOA4PLA.

The conventional metrics CDepIn and CDepOut were not so appropriate because the most class relationships of the used PLAs are generalizations. Thus, the values of these metrics were not representative in the fitness since generalizations did not suffer the action of the search operators. Most of the achieved solutions have a lower number of interfaces, however the number of operations is still the same. Hence, the mean of NumOps used in CM fitness had higher values. This point is not a problem, but the value was not so sensitive in the fitness since only three solutions had worse value for CM. Those facts allows suggesting that for the PLAs used in the study different CM could be more suitable. With respect to the other used CM, it seems that they were appropriate to measure the stability of the design.

6. DISCUSSION

In addition to the points related to the research questions, in this section, we reason about the findings of the study and their implications.

It was not possible to see the effects of the application of conventional operators MoveMethod, MoveAttribute and AddClass. Solutions generated after the application of these operators did not survive over generations probably because the changes negatively impact FM. Still about search operators, other features could be modularized into a particular component in some solutions if the feature-driven operator had been applied a greater number of times.

The percentage of improvement in FM for LPS-BET were lower than the other PLAs. It is due to both the features are better modularized in the original PLA and there are many generalizations what restricts the application of search operators. Also, the used PLAs have generalizations where search operators were not applied. Without this limitation maybe better results would be obtained.

A drawback observed in the results is related to the layered style. Elements that were originally in different layers were mixed. Some system and GUI components were excluded in several obtained solutions. In some cases, operations with similar names from different layers were grouped into one component leading to a confusing piece of design. So, we have learned that search operators should not break rules defined by the architectural style adopted in the PLA.

If PLA designs with other characteristics were provided as input to MOA4PLA other results would be achieved. Also, other results would be achieved if other metrics or search operators were applied to the same PLA designs. In this sense, other metrics could be more appropriate to evaluate PLA designs with similar characteristics taking into account that some conventional metrics were not so sensitive to the changes performed in the obtained designs. We are planning to include elegance metrics [29] in the fitness since they allow evaluating the evenness of distribution of attributes and methods among classes; external couples between classes and the ratio of attributes to methods. We are also planning to include the SPL extensibility metric [20] in the next empirical study.

These observations and the results allow the identification of some research oportunities to be carried out. They include: (a) applying other metrics in the evaluation model, (b) applying mutation operators in any type of relationship, (c) investigating if a higher rate to apply the feature-driven search operator provides PLA designs with higher feature modularization, and (d) investigating ways to consider the architectural styles rules in the search operators.

A final observation is about the usefulness of the Pareto fronts from the point of view of the architect. From the set of the potential solutions he/she needs to select one of them. This choice can be based on one metric to be prioritized according to the SPL needs or organizational goals. Or he/she could choose the solution with the lowest ED because this solution has the best trade-off between the objectives used. If the architect wants to analyse the changes performed in the design, considering that he/she usually knows the design provided as input, it is possible to verify: (a) how the features are modularized into components by seeing the features associated with architectural elements; (b) what are the new components, interfaces and classes that were added since they have generic names, such as Component103 or Interface94. By the way, after choosing a PLA design, the architect needs to rename these new architectural elements to improve the names meaning.

7. CONCLUDING REMARKS

This paper introduced MOA4PLA, a systematic and automated approach to solve the PLA design problem using search-based algorithms. MOA4PLA includes a metamodel that allows the PLA representation and the direct transformation of the original PLA design by the algorithms. It treats the problem as multi-objective and includes conventional and feature-driven metrics to evaluate achieved solutions. A feature-driven search operator was also proposed.

The results have shown that PLA design can be treated as a multi-objective problem because the metrics used are in conflict. Due to this, a set of solutions with different compromise between the metrics was achieved by the algorithms.

From the answers for the three research questions, it is possible to conclude that the feature-driven operator and the feature-driven metrics allow MOA4PLA to generate PLA designs with lower scattered and lower tangled features that the original PLAs provided as input to the approach in the context of the PLAs used in the empirical study.

In future work we are planning to explore different number of objectives and other multi-objective algoritms to analyse which is the most suitable to the problem. We also intend to carry the identified research oportunities out in further studies, such as including SPL extensibility and elegance metrics in the evaluation model.

8. ACKNOWLEDGMENTS

Authors thank CNPq for financial support.

9. REFERENCES

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Trans. Softw. Engineering*, 39(5):658–683, May 2013.

[2] S. Apel and D. Beyer. Feature cohesion in software product lines: an exploratory study. In *Proc. of the ICSE'11*, ICSE '11, pages 421–430, New York, NY, USA, 2011. ACM.

[3] M. Bowman, L. C. Briand, and Y. Labiche. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic

algorithms. *IEEE Trans. Softw. Engineering*, 36(6):817–837, 2010.

[4] S. H. Chang, H. J. La, and S. D. Kim. Key issues and metrics for evaluating product line architectures. In *Proc. of the 18th Int. Conf. on Softw. Engineering & Knowledge Engineering (SEKE)*, pages 212–219, 2006.

[5] J. L. Cochrane and M. Zeleny. *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, 1973.

[6] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2 edition, 2007.

[7] T. E. Colanzi and S. R. Vergilio. Applying search based optimization to software product line architectures: Lessons learned. In *Proc. of the 4th Symposium on Search Based Software Engineering (SSBSE)*, volume 7515, pages 259–266, 2012.

[8] T. E. Colanzi and S. R. Vergilio. Representation of software product line architectures for search-based design. In *Proc. of the 1st International Workshop on Combining Modelling and Search Based Software Engineering, ICSE'2013*, pages 28–33, 2013.

[9] A. C. Contieri Junior *et al.* Extending UML components to develop software product-line architectures: Lessons learned. In *Proc. of the 5th European Conference on Software Architecture (ECSA)*, pages 130–138, 2011.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. on Evol. Computation*, 6(2):182 –197, Apr. 2002.

[11] P. M. Donegan and P. C. Masiero. Design issues in a component-based software product line. In *Proc. of Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 3–16, 2007.

[12] J. J. Durillo, A. J. Nebro, and E. Alba. The JMetal Framework for Multi-Objective Optimization: Design and Architecture. In *Proc. of the Congress on Evol. Computation (CEC)*, pages 4138–4325, 2010.

[13] E. Figueiredo, C. Sant'Anna, A. Garcia, and C. Lucena. Applying and evaluating concern-sensitive design heuristics. *Journal of Systems and Software*, 85(2):227 – 243, 2012.

[14] E. Figueiredo *et al.* Evolving software product lines with aspects: an empirical study on design stability. In *Proc. of ICSE'08*, pages 261–270, New York, NY, USA, 2008. ACM.

[15] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic. Toward a catalogue of architectural bad smells. In *Proc. of the 5th Int. Conf. on the Quality of Software Architectures (QoSA)*, pages 146–162, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, King's College London, April 2009.

[17] R. Karimpour and G. Ruhe. Bi-criteria genetic search for adding new features into an existing product line. In *Proc. of the 1st International Workshop on Combining Modelling and Search Based Software Engineering, ICSE'2013*, pages 34–38, 2013.

[18] F. v. d. Linden, F. Schmid, and E. Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[19] R. E. Lopez-Herrejon, D. Batory, and W. Cook. Evaluating support for features in advanced modularization technologies. In *Proc. of the 19th European Conference on Object-Oriented Programming (ECOOP)*, pages 169–194, Berlin, Heidelberg, 2005. Springer-Verlag.

[20] E. A. Oliveira Junior, I. M. d. S. Gimenes, and J. C. Maldonado. A meta-process to support trade-off analysis in software product line architecture. In *Proc. of SEKE 2011*, pages 687–692, 2011.

[21] V. Pareto. *Manuel D'Economie Politique*. Ams Press, Paris, 1927.

[22] O. Räihä. A survey on search-based software design. *Computer Science Review*, 4(4):203 – 249, 2010.

[23] O. Räihä, K. Koskimies, and E. Mäkinen. Empirical study on the effect of crossover in genetic software architecture synthesis. In *Proc. of the World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pages 619–625. IEEE, 2009.

[24] O. Räihä, K. Koskimies, and E. Mäkinen. Generating software architecture spectrum with multi-objective genetic algorithms. In *Proc. of 2011 Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pages 29–36, 2011.

[25] C. N. Sant'Anna. *On the Modularity of Aspect-Oriented Design: A Concern-Driven Measurement Approach*. PhD thesis, PUC-Rio, Brazil, 2008.

[26] A. S. Sayyad, T. Menzies, and H. Ammar. On the value of user preferences in search-based software engineering: a case study in software product lines. In *Proceedings of the ICSE'13*, pages 492–501, Piscataway, NJ, USA, 2013. IEEE Press.

[27] SEI. Arcade Game Maker pedagogical product line, 2013. http://www.sei.cmu.edu/productlines/ppl/.

[28] SEI. Product line hall of fame, 2013. http://www.splc.net/fame.html.

[29] C. Simons and I. Parmee. Elegant object-oriented software design via interactive, evolutionary computation. *IEEE Trans. on Systems, Man, and Cybernetics*, 42(6):1797 –1805, nov. 2012.

[30] C. Simons, I. Parmee, and R. Gwynllyw. Interactive, evolutionary search in upstream object-oriented class design. *IEEE Trans. Softw. Engineering*, 36(6):798 –816, nov.-dec. 2010.

[31] C. L. Simons. *Interactive Evolutionary Computing in Early Lifecycle Software Engineering Design*. PhD thesis, University of the West of England, UK, 2011.

[32] J. Wüst. SDMetrics, 2013. Available at http://www.sdmetrics.com/.

[33] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. Evol. Computation*, 7:117–132, 2003.

APÊNDICE D

ARTIGO A SER PUBLICADO NO COMPSAC 2014

A Feature-Driven Crossover Operator for Product Line Architecture Design Optimization

Thelma Elita Colanzi[1,2], Silvia Regina Vergilio[1]
[1]*Computer Science Department Federal University of Paraná (UFPR)*
CP: 19081, CEP: 19031-970, Curitiba, Brazil
[2]*Computer Science Departament of State University of Maringá (UEM)*
CEP: 87.020-900, Maringá, Brazil
Email: thelma@din.uem.br, silvia@inf.ufpr.br

***Abstract*—The Product Line Architecture (PLA) design is a multi-objective optimization problem that can be properly solved in the Search Based Software Engineering (SBSE) field. However, the PLA design has specific characteristics. For example, the PLA is designed in terms of features and a highly modular PLA is necessary to enable the growth of a software product line. However, existing search based design approaches do not consider such needs. To overcome this limitation, this paper introduces a feature-driven crossover operator that aims at improving feature modularization. The proposed operator was applied in an empirical study using the multi-objective evolutionary algorithm named NSGAII. In comparison with another version of NSGAII that uses only mutation operators, the feature-driven crossover version found a greater diversity of solutions (potential PLA designs), with higher feature-based cohesion, and less feature scattering and tangling.**

***Keywords*-Product Line Architecture Design; Multi-objective Evolutionary Algorithms; Crossover Operator.**

I. INTRODUCTION

Software Product Line (SPL) approach focuses on the development of specific products within a well-defined domain by leveraging their commonalities and managing their variabilities in a systematic way. Organizations have adopted SPL approach in order to shift from the reuse of individual components to the large-scale reuse of a product-line architecture (PLA) [20]. PLA is the SPL main core asset since it entails a design that is common to all the products derived from the SPL. Such design includes components to realize common and variable features.

Architects generally reason about a SPL in terms of features, which are prominent functional and non-functional characteristics of the products. In this way, features have crucial importance in the SPL approach. Furthermore, a highly modular PLA enables the growth of a SPL. However, features are rarely modularized. Modularizing features is difficult because feature-specific code often cuts across class boundaries. Even so, it is natural to consider modularizing features as a way to modularize programs [21]. Thus, the higher feature modularization the lower functionality scattering in PLA design. Feature modularization also affects other modularity topics, such as cohesion, coupling, crosscutting and tangling [11]. Metrics for PLA design evaluation provide indicators about modularity topics [2], [16], [26]. Nevertheless, architects need to analyse a lot of metric values. So, finding the best trade-off among the used metrics to define PLA design becomes a people-intensive task. Furthermore, some metrics can be in conflict and architects need to choose which metric should be prioritized.

In this context, to recognize a good design can be easy for architects, but it is difficult to obtain. Search based software engineering (SBSE) techniques have been proven to be effective in automatically discover near-optimal solutions to such kind of problems [1], [18], [24]. The problem of PLA design can be properly solved as an optimization problem in the SBSE field if (i) adequate representations for the PLA and for the generated solutions were formulated, and (ii) assuming that the quality of a PLA design can be effectively measured. Optimization problems with two or more objective functions are called multi-objective. In such problems, the objectives to be optimized are usually in conflict, which means that they do not have a single solution. In this way, the goal is to find a good trade-off of solutions representing a possible compromise among the objectives.

Due to the lack of studies that uses SBSE techniques for improving PLA design, we applied existing approaches on search-based software design for PLA design [8]. Such approaches [4], [23], [29] optimize software design through Multi-Objective Evolutionary Algorithms (MOEAs) [6]. The results pointed that for PLA design context it is necessary to use SPL specific metrics and search operators should be both driven to features and sensitive to PLA design.

To this end, we introduced a systematic and automated approach that uses search-based algorithms to optimize PLA design [7]. This approach produces a set of solutions with the best trade-off between different and conflicting objectives, such as coupling, cohesion and feature modularization. These solutions are named non-dominated and form the Pareto front [22]. One solution is non-dominated if it is better in one objective and it is not worse in any other one.

Our approach for PLA design optimization includes a direct PLA representation for the application of multi-

objective evolutionary algorithms based on genetic algorithms (GA). A GA is inspired by the theory of natural selection and genetic evolution [6]. From an initial population, genetic operators are applied consisting of selection, crossover and mutation. These operators evolve the population, generation by generation. Through the selection operator more copies of those individuals with the best values of the objective function are selected to be parent. So the best individuals (candidate solutions) will survive in the next population. The crossover operator combines parts of two parent solutions to create a new one. The mutation operator randomly modifies a solution. The descendent population created from the genetic operators replaces the parent population. The representation adopted in our approach [9] eases the implementation of specific genetic operators. In our experiments, mutation operators were applied to PLA design and they achieved satisfactory results.

However, an open research question of our previous studies is related to the benefits of crossover operator to PLA design. The essence of crossover operator is somehow to combine parts of two separate individuals (parents) to form offspring [6]. In this context, crossover can be viewed as a situation where two architects provide alternative designs for a PLA, and decide to merge their solutions, (hopefully) taking the best parts of both designs. Räihä *et al*. [25] studied the effect of crossover in genetic architecture synthesis. Results showed that evolution applying only mutation (asexual method) performed better than the traditional random crossover with mutation. However, mutation resulted in very homogenous populations and seemed to land on a local optimum very early. To deal with this problem, they propose a more sophisticated crossover [25], finding solutions with higher quality than the asexual method. Other studies also applied crossover specific for software design [17], [29]. Existing crossover operators could be adapted to the context of PLA design. But, they are not concerned with feature modularization what is important for PLA design. In addition, inaccurate feature modularization negatively impacts on PLA reusability and maintainability [15].

By analysing the abovementioned works, we observe that a specific crossover operator that considers feature modularization is a more valuable contribution for search based PLA design than to adapt existing crossover operators to the referred context. Considering this fact, in this paper, we propose a feature-driven crossover operator and study its benefits to PLA design optimization. The research question that guides our study is whether the feature-driven crossover provides benefits when optimizing PLA design using MOEAs. To answer this question, we implement two versions of the MOEA NSGA-II [12] including different combinations of genetic operators. The first version implements an asexual method (only mutation operators) and the second version uses both mutation and feature-driven crossover. Results point out the feature-driven crossover is beneficial for PLA design optimization since it improves the feature modularization of the obtained solutions.

This paper is structured in sections. Section II reviews works on crossover operators for search-based design. Section III presents our search-based approach for PLA design and Section IV introduces the feature-driven crossover. Section V describes the empirical study performed. Sections VI and VII contain quantitative and qualitative analyzes, respectively. Section VIII concludes the paper.

II. Related work

There are few studies that employ SBSE techniques to support SPL approach [19], [27], [30], but their optimization goal is not PLA design. Most optimization works on search-based software design [24] use MOEAs with coupling and cohesion metrics [4], [23], [29], however, without addressing PLA design. Next, we describe the most related of those works where the crossover is given a special role.

Harman and Hierons [17] propose a crossover operator for the clustering area. The crossover attempts to conserve building blocks by ensuring that at least one complete cluster from one of the parents is kept intact in the crossover process. This is done by directly copying a complete cluster from one parent in the beginning of the crossover parent.

In the study of Simons *et al*. [29], recombination is achieved by means of the trans-positional crossover (TPX). In TPX, two parent individuals are chosen at random from the population. Then attributes and methods are randomly chosen and swapped between the two based on their class position in the individuals. However, TPX can only occur where swapping an attribute or method to another class would not leave the class lacking attributes or methods.

Räihä *et al*. [25] propose a complementary crossover aiming at finding solutions with higher quality than the asexual method. The operator takes into account the strengths of different individuals in crossover. It seeks complementary parents in order to produce one offspring with the best parts of the two parents. For example, it selects one parent with good modifiability characteristics and the other with good efficiency characteristics, in the hope that the offspring could inherit, at least to some extent, both desirable quality attributes. In comparison to a random crossover with randomly chosen parents, experimental results suggest that the complementary crossover provides more versatility in the produced architectures and enable more complex solutions, leading to significantly better fitness averages [25].

It is possible to adapt the mentioned crossover operators for PLA design. However, none of them encompasses particularities of SPL approach. In addition, TPX [29] and complementary crossover [25] could lead to features scattered over the PLA design, decreasing feature modularization. The operator of Harman and Hierons [17] seems to cause less impact over feature modularization. Even though, it is not sensitive to feature mapping. To overcome these

limitations, we introduce, in Section IV, a feature-based crossover operator. Such operator is specific for a search based PLA design approach, described in next section.

III. Search Based Design of PLAs

In this section we describe our approach to optimize PLA design by using MOEAs. The idea and main steps of the approach were first introduced in [7]. Here, we describe fundamental aspects for implementation of such steps: the metamodel used for representing the potential solutions, mutation operators and a fitness function. We assume that the reader is familiar with the basics of GA.

Our approach includes the two key ingredients necessary to tackle a problem using search algorithms [18]: (i) a representation of the problem: that allows symbolic manipulation; and (ii) an objective function: defined in terms of the representation to evaluate the quality of the solutions. The first ingredient is explained in the next section. The second one is defined in Section III-C.

A. PLA Representation

We decided to start the PLA design optimization considering only the static structural aspect of the architecture, expressible as a UML (stereotyped) class diagram. So, a PLA design is represented by the metamodel shown in Figure 1 and detailed in [9]. It is represented by a class diagram which contains the elements of a PLA associated with the respective description of variabilities. This allows the handling of the architectural elements by the search operators. A PLA contains architectural elements such as components, interfaces, operations and their inter-relationships. Each architectural element is associated with feature(s) by using UML stereotypes. Each element can be common to all SPL products or variable, being present only in some product(s). Variable elements are associated to variabilities that have variation points and their variants.

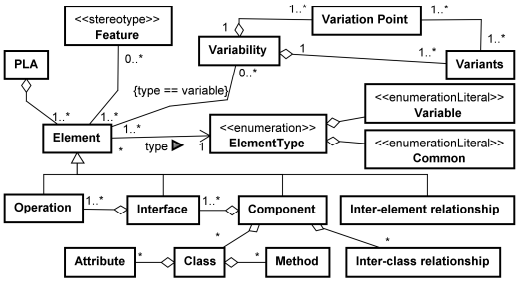


Figure 1. Metamodel of a PLA

A XMI file containing the original PLA design is the input to our approach. During the XMI processing to generate the PLA representation, each identified XMI element is instantiated as an object of the metamodel according to its type. So, the PLA representation will contain objects to represent the architectural elements, their relationships, variabilities and features associated with architectural elements.

The generation of the initial population follows the same procedure used in related work [25]. It consists on applying a randomly selected mutation operator (Section III-B) to the original PLA in order to obtain each new individual until completing the initial population.

B. Mutation Operators

Specific search operators are necessary when a direct representation is used. In our approach we used five mutation operators from related work. They are: MoveMethod [4], [29], MoveAttribute [4], [29], AddClass [4], [29], MoveOperation [8] and AddComponent [8]. AddClass moves a method or an attribute to a new class. MoveOperation moves operations between interfaces. AddComponent creates a new interface to a new component, and then moves an operation to this interface.

In addition, we have proposed a feature-driven mutation operator that focus on the PLA feature modularization [9]. This operator aims at modularizing a feature tangled with others in a component, considering that a feature usually is solved by a group of architectural elements. Features are associated with classes by using stereotypes. The operator selects an arbitrary component (c_x), and, if c_x has architectural elements assigned to different features, an arbitrary feature (f_x) is selected to be modularized in a new component (c_z). All architectural elements from c_x assigned to f_x are moved to c_z. c_z becomes client from the original components from the moved architectural elements. The application of this operator helps to obtain PLA designs which have less scattered and less tangled features, thus, it improves the feature cohesion of the architectural components.

We establish some constraints to ensure that each generated solution is consistent as follow: (i) classes and their subclasses cannot be separated; (ii) the moved element must maintain the original associated feature even if it was moved to an element associated to a different feature; and, (iii) components, interfaces and classes cannot be empty. The two first constraints were implemented into the mutation operators. We decided to repair the solutions that do not satisfy the last constraint instead of excluding them from the population. By reparing a solution we maintain both satisfactory potential designs and the population diversity.

C. Fitness Function

To evaluate the obtained solutions, the two fitness functions used in the experiments take into account conventional and SPL specific metrics. Metrics to evaluate basic design principles are called here Conventional Metrics (CM). Table I presents the CM used. The cohesion measure H indicates to the designers the components which do not have strongly-related elements. This means that in such components cohesion are weak. The size metric NumOps reveals reusability aspects of the PLA because small inter-

Table I
METRICS SUITES

Attribute	Metric	Definition
Conventional Metrics Suite (CM) [31]		
Cohesion	Relational Cohesion (H)	Average number of internal relationships per class in a component.
Coupling	Dependency of Packages (DepPack)	Number of packages on which classes and interfaces of this component depend.
	ClassDependencyIn (CDepIn)	Number of elements that depend on this class.
	ClassDependencyOut (CDepOut)	Number of elements on which this class depends.
	DependencyIn (DepIn)	Number of UML dependencies where the package is the supplier.
	DependencyOut (DepOut)	Number of UML dependencies where the package is the client.
Size	Number of Operations by Interface (NumOps)	Number of operations in the interface.
Feature-driven Metrics Suite (FM) [16], [26]		
Feature Scattering	Feature Diffusion over Architectural Components(CDAC)	Number of architectural components which contributes to the realization of a certain feature
	Feature Diffusion over Architectural Interfaces(CDAI)	Number of interfaces in the system architecture which contributes to the realization of a certain feature
	Feature Diffusion over Architectural Operations(CDAO)	Number of operations in the system architecture which contributes to the realization of a certain feature
Feature Interaction	Component-level Interlacing Between Features(CIBC)	Number of features with which the assessed feature share at least a component
	Interface-level Interlacing Between Features(IIBC)	Number of features with which the assessed feature share at least an interface
	Operation-level Overlapping Between Features(OOBC)	Number of features with which the assessed feature share at least an operation
Feature-based Cohesion	Lack of Feature-based Cohesion(LCC)	Number of features addressed by the assessed component

Table II
FITNESS FUNCTION

$FM(pla) =$	$\sum_{i=1}^{c} LCC + \sum_{i=1}^{f} CDAC + \sum_{i=1}^{f} CDAI + \sum_{i=1}^{f} CDAO + \sum_{i=1}^{f} CIBC + \sum_{i=1}^{f} IIBC + \sum_{i=1}^{f} OOBC$
$CM(pla) =$	$\sum_{i=1}^{c} DepIn + \sum_{i=1}^{c} DepOut + \sum_{i=1}^{cl} CDepIn + \sum_{i=1}^{cl} CDepOut + \frac{\sum_{i=1}^{c} DepPack}{c} + \frac{\sum_{i=1}^{itf} NumOps}{itf} + \frac{1}{\sum_{i=1}^{c} H}$

faces are, in general, easier to reuse. The other metrics refer to coupling between architectural elements.

Feature-driven Metrics (FM) (Table I) [16], [26] provide indicators about degree of modularization of a PLA design in terms of the features being realized. They measure feature scattering and feature interaction over operations, interfaces and components. They also measure the lack of feature-based cohesion. It indicates that a component that addresses many features is not stable as a modification in any of the associated features may impact the others. In addition to the negative impact on PLA reusability and maintainability, inaccurate feature modularization can lead to a wide range of design flaws, ranging from feature tangling and scattering to specific code smells, such as feature envy or god class [15].

The FM suite enables NSGAII to evaluate the feature modularization in PLA design evolved by the feature-driven mutation and crossover operators. The metrics CDAC, CDAI and CDAO consider that a feature scattered on a high number of elements has negative impact on modularity. The feature interaction, measured by the metrics CIBC, OOBC and IIBC, happens by the presence of different features in a same architectural element. This interaction happens in three levels: component, interface and operation. Lack of feature-based cohesion (LCC) indicates that a component that addresses many features is not stable as a modification in any of the associated features may impact the others.

To evaluate the solutions, we used two functions, $CM(pla)$ and $FM(pla)$, which are respectively aggregations of CM and FM (Table I). These functions are calculated according to the equations presented in Table II, where c is the number of components, itf is the number of interfaces, cl is the number of classes and f is the number of features of a design pla. These two objectives aim at analysing if the obtained solutions (PLA designs) are likely to entail proper designs from the point of view of high cohesion, low coupling, reusability and feature-based modularity.

$FM(pla)$ consists on the sum of the sum of each feature-driven metric. FM were used to evaluate the feature modularization caused by the feature-driven operator. As they are co-related, when the value of one metric decreases, the values of the others decrease as well. $CM(pla)$ is compound by the sum of several CM. In this sum we are using the mean of DepPack for all components and the mean of NumOps for all interfaces. The metric H was counted inversely in the function because we are interested in maximizing the cohesion and minimize all other metrics.

IV. FEATURE-DRIVEN CROSSOVER

Feature-driven crossover operator is concerned with maintaining and improving feature modularization of a PLA design. The proposed operator selects a feature at random and then creates children by swapping the architectural elements (classes, interfaces, operations, etc.) that realize the selected feature. In this way, we hypothesize that children might combine groups of architectural elements that better modularize some feature(s) inherited from their parents.

Figure 2 depicts a generic example of this operator. In the example, $Class1$, $Class2$, $Class3$ and $Class4$ are associated with the feature B in $Parent1$. Feature B was chosen to be swapped between the parents to generate children. So, classes associated with B are moved from $Parent2$ to $Child1$ as well as classes associated with B are moved from $Parent1$ to $Child2$. $Child1$ is completed with all other architectural elements from $Parent1$ and vice-versa for $Child2$. $Child1$ and $Child2$ are formed by white and gray classes, indicating their procedence.

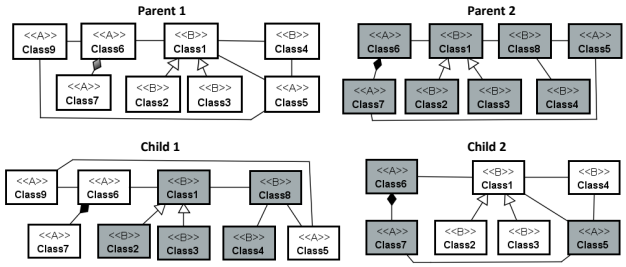


Figure 2. Crossover Operator

The pseudocode of the operator is presented in Algorithm 1. Feature-driven crossover randomly selects a feature (f_x) at line 4. From two parents ($Parent1$ and $Parent2$), two offsprings are generated in lines 7 and 11 ($Child1$ and $Child2$). Then, the elements associated with f_x are removed from $Child1$ and $Child2$ (lines 8 and 12). The next step is to add the elements (and their relationships) of $Parent2$ associated with f_x in $Child1$ (line 9) and the same for $Child2$ (line 13). After completing each child, relationships related to the SPL variabilities are updated (lines 10 and 15). Thus, it is possible to obtain children that better realize f_x.

Algorithm 1: Pseudocode of the Crossover Operator

```
1  begin
2      Input: Parent1, Parent2
3      F ← Parent1.getAllFeatures()
4      f_x ← a randomly selected feature from F
5      c_1 ← Parent1.getAllElementsAssociatedWithFeature(f_x)
6      c_2 ← Parent2.getAllElementsAssociatedWithFeature(f_x)
7      Child1 ← new Solution(Parent1)
8      Child1.removeElementsRealizingFeature(c_1, f_x)
9      Child1.addElementsRealizingFeature(c_2, f_x)
10     Child1.updateVariabilities()
11     Child2 ← new Solution(Parent2)
12     Child2.removeElementsRealizingFeature(c_2, f_x)
13     Child2.addElementsRealizingFeature(c_1, f_x)
14     Child1.updateVariabilities()
15     Output: Child1, Child2
16 end
```

Figure 3 illustrates the application of feature-driven crossover. The solutions presented are alternative design for the SPL Mobile Media [10]. Due to lack of space, only excerpts of solutions are presented and several details were omited. In $Parent1$, the feature $linkMedia$ is modularized in the system layer into the component $LinkMediaCtrl$. In $Parent2$, the feature $SMSTransfer$ is modularized in the system component $SMSTransferCtrl$. Given the selected feature $SMSTransfer$, after the application of the feature-driven crossover operator, two child solutions are generated: $Child1$ and $Child2$. In terms of fitness, $Child1$ is better than $Parent1$, $Parent2$ and $Child2$ because the component $MediaCtrl$ is more feature-driven cohesive and the feature $SMSTransfer$ is less tangled with other features in the PLA design. On the other hand, $Child2$ is worse than other three solutions in terms of modularization of the feature $SMSTransfer$ and the cohesion of $MediaCtrl$. $Child2$ probably will not survive in the next generation.

Feature-driven crossover operator aims at maintaining and improving feature modularization of a PLA design. So, its application generates solutions by swapping architectural elements associated with a randomly selected feature. Our hypothesis is that obtained solutions ($Child1$ and $Child2$) may combine architectural elements that better modularize some SPL features that were inherited from their parents. As crossover may be performed in several generations, the number of obtained childs with better feature modularization tends to improve over time. Parents, in turn, have often suffered mutations from feature-driven mutation operator in previous generations. In such cases, parents may contain well modularized features. So, the proposed feature-driven crossover complements feature-driven mutation.

V. Empirical Study

This section describes how the empirical study was performed: implementation aspects, quality indicators to evaluate the results, PLAs used, and threats to validity.

A. Implementation Aspects

To perform the study, we extended the jMetal framework [14] and implemented the most used MOEA NSGAII (Non-dominated Sorting Genetic Algorithm) [12]. The input is an XMI file containing a PLA design modeled in a UML class diagram. The PLA representation is instantiated from the XMI file given as input. Both input and output are XMI files containing the PLA design in order to ease the interchangeable use of the approach artifacts.

We executed two experiments to evaluate the proposed crossover: NSGAII-M and NSGAII-MC. NSGAII-M optimizes a PLA design through NSGAII using the mutation operators (Section III-B) according to the fitness function (Section III-C). None crossover is performed. On the other hand, in NSGAII-MC, NSGAII optimizes a PLA design using both the mutation operators and the feature-based crossover operator. At this moment, the implementation of the genetic operators does not move attributes and operations of classes involved in generalizations. Thus, the single change that may be applied to a generalization hierarchy is to move the entire hierarchy to another component.

We kicked off the parameters tuning using values adopted in related work [23], [29]. For instance, the generations number was set as 300 and the population size as 100. The mutation rate was empirically tuned as 0.9 since this rate

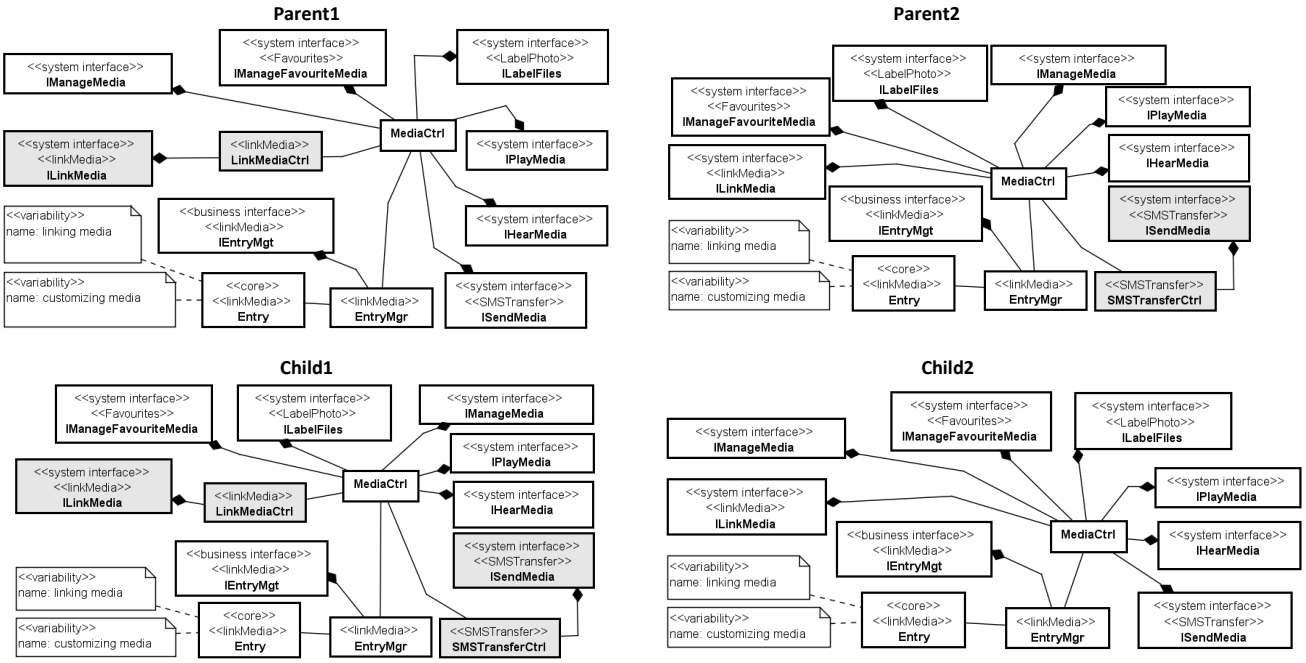


Figure 3. Feature-driven Crossover Operator: an Application Example

outperformed other configurations $(0.8, 1.0)$. The crossover rate was also empirically tuned for NSGAII-MC. The value 0.1 outperformed the other tested values: {0.2, 0.04}. It is important to highlight that in search-based design, mutation rates are high and crossover rates are low [25], [29].

Each experiment was executed 30 times for each PLA as recommended in [3], in order to reduce the possibility of the results were obtained by chance since search algorithms includes random variations. The number of fitness evaluations was used as stop criterion. To analyze the results of the empirical study three different sets of solutions were used. The first one is PF_{approx}. One set PF_{approx} for a PLA is obtained in each run of an algorithm. So, after 30 runs, 30 sets PF_{approx} were obtained. The second set is PF_{known}. This set was obtained for each PLA through the union of the 30 sets PF_{approx}, removing dominated and repeated solutions. PF_{known} represents the best solutions found by each algorithm. The last one is PF_{true}. Since the actual PF_{true} is not known, in our study this set was obtained for each PLA through the union of the sets PF_{known}, removing dominated and repeated solutions. PF_{true} represents the best solutions known to the problem. This strategy to obtain the best solutions to a problem is recommended when the ideal set of solutions is not known [32]. In addition to these three sets of solutions, two quality indicators were used to evaluate the results. They are described in the next section.

B. Quality Indicators

The most accepted indicator for performance assessment of multi-objective algorithms is the Hypervolume (HV) [32]. Comparing two PF_{approx} sets: whenever one PF_{approx} completely dominates another PF_{approx}, the HV of the former will be greater than the HV of the latter. The HV indicator calculates the volume in the region enclosed from the PF_{approx} to a reference point. A reference point is a point dominated by the solutions of all sets PF_{approx} found.

The second indicator, Euclidean Distance from the Ideal Solution (ED), is not a quality indicator, instead, it is used as a measure to help the decision maker in his/her final decision, i.e., from all solutions provided which one should be selected. ED is used to find the closest solution to the best objectives. An ideal solution has the minimum value of each objective, considering a minimization problem [5]. These minimum values are obtained from all PF_{true} solutions. In this sense, the solution with the lowest ED has the best trade-off between the objectives.

C. PLAs used in the empirical study

Table III contains some information about the PLAs used, such as the number of components, classes and features. This table also presents the original fitness of each PLA. Arcade Game Maker (AGM) [28] is a SPL created by the Software Engineering Institute (SEI) that encompasses three arcade games: Brickles, Bowling, and Pong. AGM-1 is presented in [10] and AGM-2 was obtained from AGM-1 by adding two new features: ranking and logging. AGM-3 and AGM-4 contain the same architectural elements of AGM-1 and AGM-2 respectively, but they have a different features mapping. This fact may lead to different results due to the feature-driven mutation and crossover operators.

Table III
CHARACTERISTICS OF THE PLAS

PLA	Original Fitness (FM, CM)	#Components	#Interfaces	#Classes	#Variabilities	#Features
AGM-1	(391.0, 31.82)	9	14	20	4	9
AGM-2	(499.0, 35.18)	9	14	21	5	11
AGM-3	(360.0, 31.82)	9	14	20	4	9
AGM-4	(462.0, 35.18)	9	14	21	5	11
LPS-BET	(674.0, 176.84)	56	33	115	10	16
MM-1	(608.0, 39.31)	11	16	13	7	13
MM-2	(534.0, 33.48)	8	13	10	7	13
MM-3	(434.0, 39.31)	11	16	13	7	12
MM-4	(430.0, 33.48)	8	13	10	7	12

Mobile Media (MM) [10] is a SPL composed of features that handle music, videos, and photo for portable devices. It provides support for managing different types of media. MM-1 was developed in [10] and MM-2 is equivalent to the release R8 presented in [16]. MM-1 and MM-2 are equivalent in terms of features but MM-2 has lower number of architectural elements, higher feature-based cohesion and one component is less coupled. MM-3 and MM-4 were obtained from MM-1 and MM-2, respectively, despite having different features mapping to the architectural elements.

LPS-BET [13] supports the bus city transport management. It offers features such as the use of an electronic card for transport payment; automatic toll gate opening; and unified travelling payment. It was conceived based on three existing transportation products of Brazilian cities.

The PLA designs are component-based and follow the layered architectural style including manager components that provide interfaces to enable the communication between layers (GUI, system and business layers).

D. Threats to Validity

We consider the PLAs size the main risk to affect our study conclusion validity. Only LPS-BET has medium size. However, they allow us to evaluate the performance of feature-driven crossover in different contexts of feature modularization even for small SPLs. So, with greater SPLs similar results could be observed as well. PLA designs used in the study are non-commercial. However, these cases enable us to observe the behaviour of the proposed operators.

Different feature mappings to the architectural elements could lead to different results. To mitigate this threat we adopted the mapping performed by original designers of each PLA. Although, the benefits of the genetic operators to improve feature modularization should not suffer the effects of such mapping. We adopted the same population size and the same number of generations independently of the PLA size. Different values for both parameters could imply in different, possibly better, convergence of NSGAII-MC for the largest PLA. This impact, as well as the impact of all threats can be analyzed in future studies.

VI. RESULTS AND ANALYSIS

This section contains the results obtained in the empirical study. From the results it is possible to analyse whether the feature-driven crossover operator is beneficial for search based PLA design. Table IV presents the number of solutions in PF_{true} for each PLA (second column). The third and sixth columns present the cardinality of PF_{known} of each experiment and, between parentheses, the number of solutions from PF_{known} that are included in PF_{true}. The fitness of the solutions are presented in the format (FM, CM). Solutions highlighted in bold form PF_{true}. The columns named Runtime present the mean runtime (in seconds) used to obtain each PF_{approx} and the standard deviation (between parentheses). Most times NSGAII-MC spend more time than NSGAII-M, but the difference is not so great (around 9%).

The PF_{true} sets of all PLAs are composed by solutions achieved by only one algorithm, ie. solutions found by one experiment dominates all solutions found by the other. In this point of view, NSGAII-MC has better performance than NSGAII-M. There are only two PLAs where the solutions of NSGAII-M form PF_{true}: LPS-BET and MM-1.

For AGM-1, AGM-2, AGM-3 and MM-4, NSGAII-M found lower number of solutions than NSGAII-MC. In these cases, populations of NSGAII-M are more homogenous than populations of NSGAII-MC. We consider homogeneity in terms of solutions spread in search space. Considering that the runtime of NSGAII-M is lower, it seems that it became stuck in a local optimum. As mentioned before, NSGAII-M had better performance for LPS-BET and MM-1. For LPS-BET, both experiments achieved heterogenous populations and similar number of solutions in their respective PF_{known}. LPS-BET is notoriously the largest PLA (Table III). Maybe due to this, NSGAII-M did not get trapped in a local optimum, what allowed it to obtain better solutions.

Table V presents the mean values of hypervolume considering the 30 runs of each experiment. The number between parentheses represents the standard deviation. Due to the stochastic nature of the algorithms, to perform a statistical comparison, Wilcoxon test was used at a 5% significance level [3]. The last column of the table presents the p-value of each test. Values presented in bold correspond to those where there is significant difference between the experiments. Regarding hypervolume, MM-3 is the single PLA for which there is not statistical difference between the experiments. NSGAII-M is the best for LPS-BET and MM-1 and NSGAII-MC is the best for the other six PLAs.

Table IV
RESULTS

PLA	# PF_{true}	NSGAII-M			NSGAII-MC		
		# PF_{known}	Runtime	Fitness	# PF_{known}	Runtime	Fitness
AGM-1	9	1 (0)	102.07 (2.73)	(180.0, 16.4)	9 (9)	110.16 (2.81)	**(144.0, 15.2) (137.0, 15.3) (359.0, 13.1) (339.0, 13.2) (131.0, 16.3) (165.0, 15.1) (191.0, 13.6) (220.0, 13.3) (186.0, 13.9)**
AGM-2	12	3 (0)	115.73 (3.55)	(268.0, 23.0) (273.0, 17.6) (270.0, 18.6)	12 (12)	123.62 (2.06)	**(165.0, 19.0) (163.0, 20.3) (164.0, 19.3) (332.0, 16.1) (312.0, 16.8) (187.0, 17.7) (435.0, 14.4) (376.0, 14.4) (326.0, 16.5) (370.0, 15.9) (372.0, 15.4) (220.0, 17.1)**
AGM-3	8	1 (0)	100.39 (3.55)	(169.0, 16.5)	8 (8)	111.81 (2.70)	**(198.0, 12.3) (169.0, 13.5) (174.0, 13.0) (101.0, 15.0) (131.0, 13.6) (195.0, 12.9) (120.0, 13.7) (109.0, 14.9)**
AGM-4	6	6 (0)	114.39 (1.28)	(247.0, 17.6) (241.0, 18.2) (232.0, 19.7) (234.0, 18.7) (230.0, 20.2) (226.0, 20.5)	6 (6)	124.27 (1.58)	**(299.0, 14.2) (175.0, 15.5) (283.0, 14.5) (160.0, 18.6) (277.0, 14.8) (170.0, 17.8)**
LPS-BET	34	34 (34)	565,02 (10.87)	**(757.0, 131.4) (575.0, 132.7) (556.0, 138.2) (565.0, 136.2) (560.0, 136.7) (569.0, 133.7) (566.0, 134.7) (528.0, 142.7) (522.0, 143.7) (550.0, 140.4) (496.0, 156.2) (791.0, 131.4) (497.0, 152.0) (507.0, 146.2) (498.0, 150.2) (567.0, 133.7) (549.0, 141.2) (519.0, 144.2) (555.0, 140.4) (523.0, 143.2) (667.0, 131.9) (529.0, 141.4) (512.0, 145.4) (518.0, 144.4) (503.0, 148.4) (541.0, 141.2) (506.0, 146.4) (505.0, 148.2) (511.0, 145.7) (517.0, 144.7) (585.0, 132.1) (513.0, 145.2) (848.0, 130.4) (657.0, 131.0)**	33 (0)	534,92 (11.89)	(499.0, 152.3) (508.0, 149.3) (509.0, 147.5) (502.0, 152.1) (536.0, 143.3) (533.0, 143.7) (546.0, 142.1) (506.0, 150.5) (731.0, 133.7) (609.0, 134.7) (514.0, 146.3) (585.0, 135.7) (563.0, 140.5) (579.0, 136.7) (521.0, 144.5) (520.0, 145.3) (515.0, 145.5) (531.0, 143.7) (562.0, 141.3) (560.0, 141.5) (542.0, 142.5) (537.0, 142.7) (538.0, 142.5) (651.0, 134.3) (605.0, 135.5) (630.0, 134.5) (649.0, 134.3) (798.0, 132.7) (667.0, 133.7) (785.0, 133.0) (937.0, 132.3) (569.0, 138.3) (880.0, 132.5)
MM-1	1	1 (1)	113.77 (1.44)	**(337.0, 25.0)**	2 (0)	135.06 (1.42)	(344.0, 25.6) (381.0, 25.3)
MM-2	2	2 (0)	105.44 (1.15)	(304.0, 21.1) (299.0, 22.1)	2 (2)	104.81 (1.17)	**(245.0, 19.1) (264.0, 14.7)**
MM-3	3	4 (0)	113.12 (0.46)	(378.0, 32.3) (391.0, 30.3) (366.0, 34.3) (368.0, 33.3)	3 (3)	133.80 (0.73)	**(356.0, 33.9) (362.0, 31.9) (376.0, 30.3)**
MM-4	8	4 (0)	106.98 (0.5)	(331.0, 27.9) (370.0, 24.5) (341.0, 25.9) (356.0, 24.9)	8 (8)	123.64 (0.62)	**(329.0, 25.8) (346.0, 24.4) (310.0, 27.8) (320.0, 25.8) (337.0, 24.8) (318.0, 27.3) (355.0, 24.3) (340.0, 24.8)**

Table V
MEAN OF HYPERVOLUME

PLA	NSGAII-M Mean (Std. Dev.)	NSGAII-MC Mean (Std. Dev.)	p-value
AGM-1	162611.1 (9590.6)	**186297.1** (14140.7)	9.54e-07
AGM-2	272010.2 (9158.0)	**298136.7** (21537.7)	3.82e-04
AGM-3	54854.9 (1781.6)	**61571.3** (6783.0)	1.17e-02
AGM-4	179225.6 (4954.2)	**202557.3** (11506.0)	7.61e-16
LPS-BET	**346436.9** (4374.0)	341895.4 (5568.2)	0.005302
MM-1	**237509.7** (19400.1)	221099.6 (27746.0)	0.04789
MM-2	455344.2 (26797.4)	**473216.0** (39243.6)	0.007301
MM-3	16528.3 (1392.3)	17467.3 (2276.8)	0.1153
MM-4	29446.2 (1489.9)	**31877.2** (4060.3)	0.03577

Table VI
THE LOWEST EUCLIDEAN DISTANCES

PLA	Ideal Solution	NSGAII-M ED	NSGAII-M Fitness	NSGAII-MC ED	NSGAII-MC Fitness
AGM-1	(131.0,13.1)	49,1	(180.0,16.3)	**3,2**	**(131.0,16.3)**
AGM-2	(163.0,14.3)	105,3	(268.0,23.0)	**5,0**	**(164.0,19.2)**
AGM-3	(101.0,12.3)	68,1	(169.0,16.5)	**2,6**	**(101.0,15.0)**
AGM-4	(160.0,14.2)	66,3	(226.0,20.5)	**4,4**	**(160.0,18.6)**
LPS-BET	(496.0,130.4)	**18,8**	**(506.0,146.4)**	21,4	(509.0,147.4)
MM-1	(337.0,25.0)	**0,0**	**(337.0,25.0)**	7,0	(344.0,25.5)
MM-2	(245.0,14.6)	54,5	(299.0,22.0)	**4,4**	**(245.0,19.0)**
MM-3	(356.0,30.2)	10,7	(366.0,34.3)	**3,5**	**(356.0,33.8)**
MM-4	(310.0,24.3)	21,3	(331.0,27.9)	**3,5**	**(320.0,25.8)**

The ED indicator corroborates the results of hypervolume for each PLA. Table VI shows the fitness of the ideal solution as well as the lowest ED and the fitness of the solution nearest to the ideal one of each experiment. There is a large distance between the solutions with the lowest ED found by the experiments for most PLA designs. The distances are more similar for PLAs where NSGAII-M was the best (LPS-BET and MM-1) and for MM-3 where there is not difference between both experiments in terms of hypervolume.

Results show that mutation operators are enough to optimize a PLA design. NSGAII-M achieves satisfactory solutions in terms of fitness. Values of FM and CM are much better than the original ones for all PLAs. There were only two solutions (6%) in PF_{true} of LPS-BET where the value of FM was worse than the original one.

From the solutions found by NSGAII-MC, five solutions (15%) of PF_{true} for LPS-BET have worse values for FM with respect to the original PLA. In spite of this, the application of feature-driven crossover operator in NSGAII-MC allows finding results even better than NSGAII-M. Populations are more heterogenous without decreasing quality of solutions. The crossover enables better exploitation of the search space achieving greater diversity of solutions.

VII. DISCUSSION

Beyond the quantitative analysis, it is important to analyse the quality of the achieved solutions under the designer point of view. To do this, we compared solutions with the lowest ED of each experiment for each PLA, since these solutions have the best trade-off between the objectives used. Here, we analyzed the solutions obtained, considering the

corresponding architecture, its architectural elements and properties measured by fitness function.

Solutions found by both experiments are similar considering: number of components, number of interfaces, numbers of classes and components associated with a single feature. The greatest difference between them refers to feature modularization. Some features are more tangled and scattered in solutions of NSGAII-M what leads to lower feature-based cohesion. And, worse feature modularization implies in higher coupling. This is why the fitness of NSGAII-MC solutions are better than of NSGAII-M solutions for conventional and feature-driven metrics (CM and FM).

To illustrate this, Figures 4 and 5 show excerpts of the solutions for MM-2 with the lowest ED. For both experiments it is possible to see that there are components to modularize the features $create/delete$ and $view/playMedia$ in the system layer (components with the suffix Ctrl). In the NSGAII-M solutions (Figure 4), $create/delete$ is tangled with the features SMS and $labelMedia$ whereas $view/playMedia$ is tangled with SMS and $linkMedia$ at operation level. In addition, SMS is scattered in two interfaces of different components. On the other hand, in the NSGAII-MC solution (Figure 5), SMS was modularized into $Component3679Ctrl$ increasing feature-based cohesion of the features $create/delete$ and $view/playMedia$. Operations associated with $linkMedia$ are not shown in the picture, but they were modularized into another component.

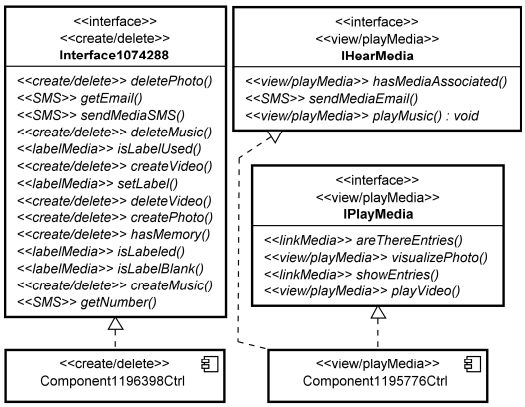


Figure 4. Partial Solution of NSGAII-M for MM-2

Both solutions are better than the original PLA. However, NSGAII-MC solution is even better. Also, there are other points in this solution (not showed here due to lack of space) in favour of NSGAII-MC. So, it is possible to state that NSGAII-MC obtains better results than NSGAII-M.

However, the feature-driven crossover has its weakness: the guarantee of consistency of the solutions generated is more difficult when applying feature-driven crossover. Some operations were lost in certain NSGAII-MC solutions. We hypothesize that this happens due to the exclusion of entire interfaces associated with the feature under modularization

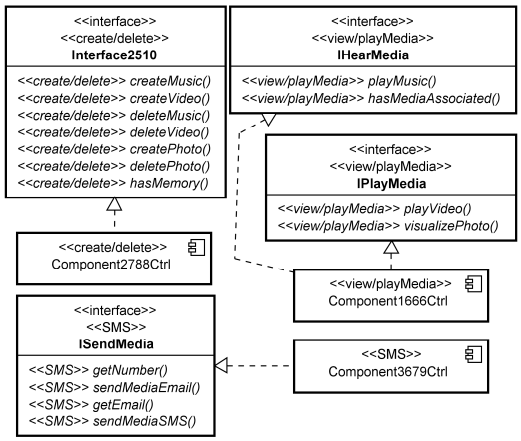


Figure 5. Partial Solution of NSGAII-MC for MM-2

(lines 8 and 12 of Algorithm 1). Then, the interfaces associated with the feature under modularization were added (lines 9 and 13), but they were better modularized in terms of features, i.e., the feature were less tangled leading to the lack of some operations associated with other features.

This suggests that some improvement is required to prevent cases like this. In our study, features are mapped to architectural elements in fine granularity, i.e., all architectural elements (components, interfaces, operations of interfaces, classes, attributes and methods) are associated with at least one feature. Nevertheless, features are usually mapped in coarse granularity at component, interface and class levels, what prevents such kind of occurence.

With respect to feature modularization, in some solutions of NSGAII-M and NSGAII-MC other features could be modularized into a particular component if the feature-driven mutation operator had been applied greater number of times.

VIII. Concluding Remarks

We introduced a feature-driven crossover operator that generates optimized designs from two different PLA designs by swapping groups of architectural elements realizing a particular feature. We implemented two versions of NSGAII: NSGAII-M using only mutation operators, and NSGAII-MC using mutation operators and the feature-driven crossover operator. We performed an empirical study applying NSGAII-M and NSGAII-MC for nine PLA designs aiming at comparing the obtained results.

We hypothesize that feature-driven crossover provides benefits to PLA design. The results of the conducted study corroborate to our hypothesis. Feature-driven crossover allows NSGAII find better solution in terms of fitness and higher diversity of solutions. A qualitative analysis showed the operator also provides solutions with higher quality in terms of feature-based cohesion and feature scattering and tangling.

As future work, we intend to improve the feature-driven crossover to settle its limitations mentioned at the end of Section VII. Furthermore, we will investigate if a higher application rate to the feature-driven mutation operator provides PLA designs with higher feature modularization. We also intend to ask the opinion of the original PLA designers about the quality of obtained solutions. Thus, it will be possible to improve the feature-driven crossover encompassing drawbacks pointed by the designers.

ACKNOWLEDGMENT

We would like to thank CNPq for financial support.

REFERENCES

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Trans. Softw. Engineering*, vol. 39, no. 5, pp. 658–683, May 2013. [Online]. Available: http://dx.doi.org/10.1109/TSE.2012.64

[2] S. Apel and D. Beyer, "Feature cohesion in software product lines: an exploratory study," in *ICSE'2011*, 2011, pp. 421–430.

[3] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *ICSE'11*, 2011, pp. 21–28.

[4] M. Bowman, L. C. Briand, and Y. Labiche, "Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms," *IEEE Trans. Softw. Engineering*, vol. 36, no. 6, pp. 817–837, 2010.

[5] J. L. Cochrane and M. Zeleny, *Multiple Criteria Decision Making*. Univers. of South Carolina Press, Columbia, 1973.

[6] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed. Springer-Verlag New York, Inc., 2007.

[7] T. E. Colanzi, "Search based design of software product lines architectures," in *ICSE'2012, Doctoral Symposium*, 2012, pp. 1507 –1510.

[8] T. E. Colanzi and S. R. Vergilio, "Applying search based optimization to software product line architectures: Lessons learned," in *SSBSE'12*, vol. 7515, 2012, pp. 259–266.

[9] ——, "Representation of software product line architectures for search-based design," in *CMSBSE-ICSE'2013*, 2013, pp. 28–33.

[10] A. C. Contieri Junior *et al.*, "Extending UML components to develop software product-line architectures: Lessons learned," in *Proc. of the 5th ECSA*, 2011, pp. 130–138.

[11] K. Czarnecki, "Variability in software: State of the art and future directions," in *Intl Conf. on Fundamental Approaches to Software Engineering*, ser. FASE'13, 2013, pp. 1–5.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182 –197, Apr. 2002.

[13] P. M. Donegan and P. C. Masiero, "Design issues in a component-based software product line," in *Proc. of Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, 2007, pp. 3–16.

[14] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[15] E. Figueiredo, C. Sant'Anna, A. Garcia, and C. Lucena, "Applying and evaluating concern-sensitive design heuristics," *Journal of Systems and Software*, vol. 85, no. 2, pp. 227 – 243, 2012.

[16] E. Figueiredo *et al.*, "Evolving software product lines with aspects: an empirical study on design stability," in *ICSE'2008*, 2008, pp. 261–270.

[17] M. Harman and R. Hierons, "A new representation and crossover operator for search-based optimization of software modularization," in *GECCO'2002*, 2002, pp. 1351–1358.

[18] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 1–61, Dec. 2012.

[19] R. Karimpour and G. Ruhe, "Bi-criteria genetic search for adding new features into an existing product line," in *CMSBSE- ICSE'2013*, 2013, pp. 34–38.

[20] F. v. d. Linden, F. Schmid, and E. Rommes, *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[21] R. E. Lopez-Herrejon, D. Batory, and W. Cook, "Evaluating support for features in advanced modularization technologies," in *Proc. of the 19th European Conference on Object-Oriented Programming (ECOOP)*, 2005, pp. 169–194.

[22] V. Pareto, *Manuel D'Economie Politique*. Paris: Ams Press, 1927.

[23] O. Räihä, K. Koskimies, and E. Mäkinen, "Generating software architecture spectrum with multi-objective genetic algorithms," in *World Congress on Nature and Biologically Inspired Computing*, 2011, pp. 29–36.

[24] O. Räihä, "A survey on search-based software design," *Computer Science Review*, vol. 4, no. 4, pp. 203 – 249, 2010.

[25] O. Räihä, K. Koskimies, and E. Mäkinen, "Complementary crossover for genetic software architecture synthesis." in *ISDA*, 2010, pp. 266–271.

[26] C. N. Sant'Anna, "On the modularity of aspect- oriented design: A concern-driven measurement approach," Ph.D. dissertation, PUC-Rio, Brazil, 2008.

[27] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: a case study in software product lines," in *ICSE'2013*, 2013, pp. 492–501.

[28] SEI. (2014) Arcade Game Maker pedagogical product line. Http://www.sei.cmu.edu/productlines/ppl/.

[29] C. Simons, I. Parmee, and R. Gwynllyw, "Interactive, evolutionary search in upstream object-oriented class design," *IEEE Trans. Softw. Engineering*, vol. 36, no. 6, pp. 798 –816, nov.-dec. 2010.

[30] S. Wang, S. Ali, and A. Gotlieb, "Minimizing test suites in software product lines using weight-based genetic algorithms," in *GECCO 2013*, 2013, pp. 1493–1500.

[31] J. Wüst. (2013) SDMetrics. Http://www.sdmetrics.com/.

[32] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Trans. Evol. Computation*, vol. 7, pp. 117–132, 2003.

APÊNDICE E

RESULTADOS DO ESTUDO 4

Neste apêndice são apresentados resultados alcançados quando da realização do Estudo 4 a respeito das configurações de funções de *fitness* para o modelo de avaliação.

O tempo médio de execução das trinta rodadas de cada experimento e o desvio padrão são apresentados na Tabela E.1. As três configurações de modelo de avaliação demandaram tempos de execução similares. Porém, na maior parte das vezes os experimentos NSGAII-M-FE e NSGAII-MC-FE demandaram menor tempo de execução que os demais experimentos. O desvio padrão desses experimentos também é menor, tendo em vista que encontraram um número menor de soluções e que a métrica de extensibilidade não se alterou ao longo do processo evolutivo.

Tabela E.1: Estudo 4 - Tempo de execução dos experimentos

PLA	**Tempo (seg)**	**Desvio Padrão**	**Tempo (seg)**	**Desvio Padrão**	**Tempo (seg)**	**Desvio Padrão**
	NSGAII-M-FE		**NSGAII-M-FC**		**NSGAII-M-FCE**	
AGM-1	89,92	0,72	102,59	1,06	100,32	1,05
AGM-2	98,12	0,41	101,93	0,55	105,86	1,01
AGM-3	90,54	0,41	103,66	0,70	101,45	1,03
AGM-4	99,01	0,45	112,22	0,75	109,93	0,88
LPS-BET	383,79	4,07	486,25	27,28	469,74	13,41
MM-1	106,62	0,90	111,00	0,91	107,91	1,06
MM-2	99,34	0,32	102,72	0,35	99,66	0,31
MM-3	108,62	0,31	114,25	1,15	110,09	0,98
MM-4	104,70	0,47	109,97	1,51	105,97	1,38
PLA	**NSGAII-MC-FE**		**NSGAII-MC-FC**		**NSGAII-MC-FCE**	
AGM-1	123,59	1,30	132,05	2,36	131,75	1,68
AGM-2	123,73	2,70	136,49	1,87	137,81	3,50
AGM-3	126,87	2,70	140,22	2,69	139,65	2,10
AGM-4	126,69	1,62	143,24	1,93	143,47	2,78
LPS-BET	749,50	12,96	523,18	25,85	527,82	38,01
MM-1	125,56	0,59	135,96	2,64	134,53	2,51
MM-2	115,71	0,63	123,91	0,90	123,92	1,09
MM-3	128,54	0,56	122,32	3,65	127,10	5,22
MM-4	123,10	0,82	122,89	2,49	125,03	2,40

Com relação aos experimentos NSGAII-M-FCE e NSGAII-MC-FCE, o fato de terem trabalhado com 3 objetivos não influenciou no tempo de execução do algoritmo, o que indica que o objetivo adicional não aumentou tanto a complexidade de solução do

problema. Outro ponto a destacar é que os experimentos que aplicam o operador de cruzamento demandam mais tempo que os experimentos que aplicam somente mutação.

As Tabelas E.2 e E.3 apresentam os conjuntos de soluções não dominadas retornados pelos experimentos após as trinta execuções (PF_{known}). Para cada PLA apresenta-se o número de soluções retornadas e o *fitness* de cada solução. Analisando somente os conjuntos PF_{known} de cada experimento não é possível determinar um comportamento padrão em relação ao número de soluções encontradas. O único destaque é que os experimentos NSGAII-M-FE e NSGAII-MC-FE retornaram somente uma solução para cada PLA.

Tabela E.2: Estudo 4 - Resultados do NSGAII-M

PLA	**NSGAII-M-FE**		**NSGAII-M-FC**		**NSGAII-M-FCE**	
	# PF_{known}	***Fitness*** (FM, Ext)	# PF_{known}	***Fitness*** (FM, CM)	# PF_{known}	***Fitness*** (FM, CM, Ext)
AGM-1	1	(275.0, 0.52)	2	(278.0, 16.66) (276.0, 16.72)	3	(277.0, 18.16, 0.52) (278.0, 18.11, 0.52) (280.0, 16.375, 0.52)
AGM-2	1	(321.0, 0.61)	2	(325.0, 19.44) (324.0, 19.46)	3	(328.0, 19.44, 0.61) (323.0, 19.46, 0.65) (323.0, 20.45, 0.61)
AGM-3	1	(196.0, 0.52)	1	(178.0, 15.80)	1	(187.0, 15.80, 0.52)
AGM-4	1	(243.0, 0.61)	2	(239.0, 18.04) (269.0, 17.48)	5	(243.0, 19.66, 0.61) (257.0, 19.45, 0.61) (324.0, 19.01, 0.61) (249.0, 19.47, 0.61) (277.0, 19.05, 0.61)
LPS-BET	1	(526.0, 1000.0)	21	(519.0, 144.87) (522.0, 141.90) (531.0, 141.03) (523.0, 141.25) (525.0, 141.12) (676.0, 125.06) (659.0, 126.06) (692.0, 124.06) (678.0, 124.40) (540.0, 133.73) (605.0, 126.06) (661.0, 125.90) (549.0, 130.93) (546.0, 131.73) (594.0, 126.15) (571.0, 128.06) (576.0, 127.06) (554.0, 129.88) (561.0, 129.25) (564.0, 129.06) (777.0, 122.73)	7	(553.0, 136.88, 1000.0) (554.0, 134.25, 1000.0) (566.0, 132.26, 1000.0) (560.0, 133.25, 1000.0) (550.0, 142.06, 1000.0) (528.0, 148.25, 1000.0) (535.0, 144.46, 1000.0)
MM-1	1	(396.0, 0.77)	2	(410.0, 25.52) (396.0, 25.53)	2	(390.0, 25.53, 0.77) (410.0, 25.52, 1.0)
MM-2	1	(365.0, 0.77)	2	(392.0, 22.74) (370.0, 22.75)	2	(392.0, 22.74, 0.77) (370.0, 22.75, 0.77)
MM-3	1	(362.0, 0.77)	14	(347.0, 32.94) (350.0, 32.19) (354.0, 31.19) (366.0, 29.80) (375.0, 29.18) (374.0, 29.78) (355.0, 30.47) (406.0, 28.55) (386.0, 29.14) (388.0, 28.71) (398.0, 28.56) (393.0, 28.59) (379.0, 29.16) (372.0, 29.79)	13	(376.0, 29.22, 0.77) (388.0, 28.71, 0.77) (359.0, 31.19, 0.77) (364.0, 30.48, 0.77) (371.0, 29.83, 0.77) (356.0, 31.94, 0.77) (410.0, 28.55, 0.77) (394.0, 28.63, 0.77) (402.0, 28.56, 0.77) (397.0, 28.59, 0.77) (385.0, 29.16, 0.77) (375.0, 29.81, 0.77) (380.0, 29.18, 0.77)
MM-4	1	(305.0, 0.77)	18	(384.0, 22.75) (361.0, 22.86) (316.0, 25.08) (321.0, 24.43) (368.0, 22.80) (376.0, 22.77) (309.0, 26.51) (319.0, 24.45) (314.0, 25.1) (310.0, 25.78) (308.0, 27.52) (352.0, 23.23) (327.0, 23.81) (335.0, 23.79) (336.0, 23.27) (322.0, 24.41) (318.0, 25.07) (344.0, 23.25)	14	(301.0, 26.80, 0.77) (368.0, 22.75, 0.77) (302.0, 25.78, 0.77) (313.0, 23.83, 0.77) (352.0, 22.80, 0.77) (329.0, 23.27, 0.77) (303.0, 25.1, 0.77) (320.0, 23.81, 0.77) (322.0, 23.32, 0.77) (345.0, 22.86, 0.77) (307.0, 24.45, 0.77) (308.0, 24.43, 0.77) (360.0, 22.77, 0.77) (337.0, 23.25, 0.77)

Tabela E.3: Estudo 4 - Resultados do NSGAII-MC

PLA	NSGAII-MC-FE		NSGAII-MC-FC		NSGAII-MC-FCE	
	# PF_{known}	***Fitness*** (FM, Ext)	# PF_{known}	***Fitness*** (FM, CM)	# PF_{known}	***Fitness*** (FM, CM, Ext)
AGM-1	1	(271.0,0.52)	3	(273.0,18.11)(1331.0,16.12) (280.0,16.66)	2	(276.0,16.72,0.52) (277.0,16.66,0.52)
AGM-2	1	(313.0,0.61)	1	(302.0,18.0)	8	(315.0,20.91,0.61)(334.0,19.43,0.61) (322.0,20.34,0.61)(333.0,19.4405,0.61) (321.0,20.34,0.61)(332.0,20.0,0.61) (331.0,20.01,0.61)(314.0,21.9102,0.61)
AGM-3	1	(196.0,0.52)	10	(201.0,16.75)(196.0, 16.80) (169.0,19.15)(188.0,17.25) (177.0,18.69)(175.0,18.77) (194.0,16.88)(252.0,15.75) (243.0,16.16)(246.0,16.14)	2	(198.0, 15.58, 0.52) (193.0,15.80,0.52)
AGM-4	1	(232.0,0.61)	4	(214.0,19.54)(213.0,22.54) (242.0,19.04)(261.0,18.48)	5	(216.0,21.54,0.61)(242.0,18.50,0.61) (230.0,19.41,0.61)(236.0,19.04,0.61) (224.0,19.95,0.61)
LPS-BET	1	(524.0,1000.0)	20	(367.0,83.49)(1133.0,45.54) (380.0,71.87)(696.0,50.54) (925.0,48.53)(839.0,50.28) (1075.0,47.53)(542.0,60.88) (474.0,61.09)(446.0,65.87) (403.0,68.88)(609.0,58.53) (640.0,57.28)(652.0,54.53) (871.0,49.37)(672.0,53.61) (469.0,62.88)(672.0,53.61) (678.0,51.53)(1079.0,47.53)	12	(2976.0,60.3026,1000.0)(924.0,69.46,1000.0) (472.0,70.2192,1000.0)(471.0,84.52,1000.0) (460.0,89.45,1000.0)(3561.0,60.12,1000.0) (453.0,92.45,1000.0)(1303.0,62.14,1000.0) (934.0,63.44,1000.0)(1093.0,62.6,1000.0) (1330.0,61.41,1000.0)(5095.0,58.57,1000.0)
MM-1	1	(407.0,0.77)	6	(385.0,27.40)(541.0,26.40) (1086.0,24.41)(548.0,25.20) (594.0,24.64)(588.0,25.19)	3	(356.0,26.27,0.77)(358.0,25.72,0.77) (383.0,21.29,0.54)
MM-2	1	(362.0,0.77)	4	(1084.0,21.75)(1065.0,22.22) (364.0,22.75)(386.0,22.74)	5	(338.0,22.58,0.77)(388.0,21.71,0.77) (1025.0,21.32,0.77)(1041.0,20.32,0.77) (1042.0,19.90,0.77)
MM-3	1	(362.0,0.77)	15	(175.0,24.52) (176.0,23.95)(169.0,25.54) (330.0,19.46)(313.0,19.98) (184.0,23.91)(283.0,20.22) (260.0,21.14)(186.0,23.50) (235.0,21.34)(187.0,22.91) (191.0,22.43)(270.0,20.59) (207.0,21.59)(203.0,22.05)	11	(136.0,24.25,1.74)(219.0,19.16,0.17) (191.0,20.94,1.74)(155.0,23.25,0.54) (201.0,19.80,0.17)(174.0,21.94,0.54) (137.0,23.62,1.7499)(172.0,22.51,0.54) (156.0,22.62,0.54)(153.0,23.51,1.74) (155.0,22.94,1.74)
MM-4	1	(305.0,0.77)	14	(256.0,20.03) (385.0,17.82) (352.0,18.03)(323.0, 18.41) (273.0,19.51)(279.0, 19.5) (294.0,18.94)(319.0, 18.44) (300.0,18.90)(298.0, 18.91) (239.0,20.43)(197.0, 22.22) (221.0,20.95)(207.0, 21.56)	16	(320.0,18.61,0.77)(299.0,18.92,0.77) (282.0,19.37,0.77) (269.0,19.91,0.77) (260.0,20.53,0.77) (449.0,19.66,0.44) (201.0,21.78,0.77) (284.0,22.94,0.44) (244.0,20.83,0.77) (374.0,19.86,0.44) (261.0,20.44,0.77) (285.0,22.26,0.44) (237.0,21.34,0.77) (223.0,21.36,0.77) (295.0,21.62,0.44) (305.0,20.03,0.44)