JOHNNY MAIKEO FERREIRA

TESTE DE LINHA DE PRODUTO DE SOFTWARE BASEADO EM MUTAÇÃO DO DIAGRAMA DE CARACTERÍSTICAS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Profa. Dra. Silvia Regina Vergílio.

CURITIBA

2012

JOHNNY MAIKEO FERREIRA

TESTE DE LINHA DE PRODUTO DE SOFTWARE BASEADO EM MUTAÇÃO DO DIAGRAMA DE CARACTERÍSTICAS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Profa. Dra. Silvia Regina Vergílio.

CURITIBA

2012

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

Sigla	Significado
AETG	Automatic Efficient Test Generator
AFS	Add Feature to Set Relation
CAS	Car Audio System
CDC	Create Depends Constraint
CEC	Create Excludes Constraint
DFL	Decrease Solitary Feature Lower Bound
DFU	Decrease Solitary Feature Upper Bound
DRL	Decrease Set Relation Lower Bound
DRU	Decrease Set Relation Upper Bound
FAST	Family-oriented abstraction, specification and translation
FDC	Change Feature from Depends Constraint
FIG	Feature Inclusion Graph
FM	Feature Model
FODA	Feature-Oriented Domain Analysis
FORM	Feature-Oriented Reuse Method
FMTS	Feature Mutation Test Suite
IFL	Increase Solitary Feature Lower Bound
IFU	Increase Solitary Feature Upper Bound
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRL	Increase Set Relation Lower Bound
IRU	Increase Set Relation Upper Bound
LPS	Linha de Produto de Software
OVM	Orthogonal Variability Model
RDC	Remove Depends Constraint
REC	Remove Excludes Constraint
RFS	Remove Feature from Set Relation
RSEB	Reuse-Driven Software Engineering Business
RSR	Remove Set Relation
SPL	Software Product Line
UML	Unified Modeling Language
VV&T	Validação, Verificação e Teste
XML	Extensible Markup Language

RESUMO

Uma linha de produto de software (LPS) pode ser definida como um conjunto de sistemas que compartilham um conjunto comum de características e que satisfazem os requisitos específicos de um determinado domínio. O desenvolvimento de software em linha de produto propõe soluções para os problemas causados pela rápida evolução e complexidade das aplicações. Metodologias de desenvolvimento de LPS têm como base alguns artefatos como, por exemplo, o modelo de características. Este modelo representa como as características são utilizadas durante a criação dos produtos, e também pode ser utilizado durante a atividade de teste, que se torna mais crítica e complexa quando comparada ao processo tradicional. O teste de todas as combinações de características, produtos, é impraticável devido à complexidade crescente das aplicações, e somente um subconjunto destes produtos pode ser testado. As técnicas existentes para seleção de produtos a partir do diagrama de características são geralmente baseadas em teste combinatorial, requerendo a combinação de pares de características do diagrama (teste *pair-wise*). Essas técnicas não consideram possíveis defeitos que estes diagramas podem conter. A aplicação de uma abordagem baseada em defeitos pode aumentar a probabilidade de encontrar defeitos e a confiança de que os produtos de uma LPS estão de acordo com os seus requisitos. Considerando esse fato, esse trabalho introduz uma abordagem baseada em teste de mutação para auxiliar na seleção de produtos para o teste de LPS. Para isso, são introduzidos operadores de mutação juntamente com um processo de teste e uma ferramenta de automatização. Por fim, são apresentados resultados experimentais, e uma comparação com o teste *pair-wise,* que demonstra que diferentes tipos de defeitos podem ser revelados pela abordagem introduzida.

ABSTRACT

A software product line (SPL) can be defined as a group of systems sharing a common set of features that fulfill requirements of a certain domain. Software product line development proposes solutions for problems caused by fast software evolution and growth. Some SPL development methodologies are based on artifacts, such as, the feature model. This model is widely used to represent how features are combined to create new products, and also can be used during the test phase, which is in general more complex and critical when compared to the traditional process. Testing all combinations of characteristics (products) is infeasible, in practice, due to the growing complexity of the applications, and only a subset of products can be usually tested. Existing techniques for products selection from feature diagrams are generally based on combinatorial testing, requiring pair-wise testing of feature interactions. They do not consider possible faults that can be present in the diagrams. The application of a fault-based approach can increase the probability of finding faults and the confidence that the SPL products match their requirements. Considering that, this work introduces a mutation based approach to help in the selection of products for feature testing of SPLs. Mutation operators are proposed, a testing process and an automated tool are introduced. In additional to this, experimental results are reported, and a comparison with *pair-wise* testing shows that other kind of faults can be revealed by the introduced approach.

CAPÍTULO 1 - INTRODUÇÃO

1.1 Contexto

Uma linha de produto de software (LPS) pode ser definida como um conjunto de sistemas que compartilham um conjunto comum de características gerenciadas que satisfazem os requisitos específicos de um seguimento de mercado ou missão, e que são desenvolvidos a partir de um núcleo e de forma planejada [35]. Características são atributos de um sistema de software que afetam diretamente os usuários finais. Um produto de uma LPS é obtido pela combinação das suas características. A abordagem de desenvolvimento em linha produto de software propõe soluções para os problemas causados pela rápida evolução e complexidade das aplicações de software, tais como funcionalidades desenvolvidas em lugares diferentes, mudanças rápidas repetidas em lugares diferentes, características semelhantes se comportando de maneira diferente em produtos diferentes e altos custos de manutenção [19].

Com um processo baseado na reutilização de componentes, arquiteturas e requisitos, a abordagem de desenvolvimento em LPS é dividida em duas etapas principais. A primeira, Engenharia de Domínio, consiste em desenvolver as características comuns a todos os produtos. A segunda, Engenharia de Aplicação, consiste no desenvolvimento dos produtos utilizando características comuns da etapa anterior [19].

Visando melhorias nos processos de desenvolvimento em linha de produto de software diversas metodologias foram propostas. Destas destacam-se, a FAST [47], que introduziu o conceito de divisão em etapas do processo de desenvolvimento. A FODA [22] que sugeriu uma análise de características dentro do domínio a partir de uma lista de variabilidades criadas, agrupando-as de maneira gráfica em um modelo de características. O FORM [23] que expandiu a metodologia FODA a partir da utilização de modelos de características durante o processo de engenharia de requisitos descrevendo como este pode ser utilizado para o desenvolvimento da arquitetura de domínio e dos componentes reutilizáveis.

Muitas das metodologias de desenvolvimento citadas utilizam o modelo de características para uma LPS. Este modelo é uma representação visual, em formato de árvore, das características, comuns e variantes, de uma linha de produto de software e de suas relações. Este modelo é utilizado para ilustrar como as

características são utilizadas para a montagem dos produtos, de forma que estes sejam diferenciados pelas características que o compõem [42].

Em todas essas metodologias a atividade de teste é essencial. O teste de LPS torna-se mais crítico e complexo comparado ao processo tradicional devido ao desenvolvimento de um produto ocorrer em duas etapas: o desenvolvimento dos componentes para reúso, na engenharia de domínio, e o desenvolvimento do produto a partir dos componentes reusáveis, na engenharia da aplicação [51]. Essa diferenciação do processo tradicional possibilitou o surgimento de novas estratégias de teste de software específicas para famílias de produto.

Entre as estratégias de teste de LPS destacam-se o teste de produto por produto [39], que consiste em testar todos os produtos criados de forma independente, indo na contramão do conceito de reutilização dentro de LPS. Teste incremental [18], nesta abordagem as características previamente testadas em um produto são apenas testadas por técnicas de regressão. E por fim, teste de instanciação de artefatos reutilizáveis [46], nesta abordagem os elementos comuns da LPS são testados apenas uma vez, e para cada produto criado apenas a parte variante é considerada em novos testes.

As dificuldades de teste de LPS encontram-se na completa integração e teste de sistema, na engenharia de domínio que geralmente não é possível devido ao fato de as variações não terem sido especificadas, e também, ao fato de que testar as diferentes combinações de componentes leva a um crescimento exponencial nas configurações de teste [30]. Ainda segundo McGregor [30], é difícil decidir o quanto se deve depender dos testes de domínio no teste de aplicação.

Apesar das dificuldades é observado um crescimento na adoção da abordagem de LPS pela indústria, e como consequência, um aumento na demanda e no interesse em teste de LPS. Esse crescimento é comprovado pelo grande número de pesquisas relacionadas ao tema [12, 17, 28, 42, 45], mencionando publicações referentes a aspectos gerais ou específicos do teste de LPS. Todos estes estudos enfatizam o teste de variabilidades e de características de fundamental importância. Uma questão importante a ser levantada refere-se a como assegurar que produtos gerados a partir de um diagrama de características de uma LPS estão de acordo com os requisitos.

Para responder a essa questão, idealmente, todos os produtos deveriam ser validados. Entretanto, infelizmente, o espaço de possíveis combinações de produtos em muitos casos é enorme e o teste exaustivo de todas as combinações é impraticável [10]. O aumento no tamanho e na complexidade das aplicações pode tornar o teste de todas as funcionalidades, na prática, inviável. Isto é similar ao que ocorre no teste de aplicações tradicionais no qual é impraticável realizar o teste de

todos os caminhos de um programa, e no qual critérios de teste são usados para selecionar os melhores caminhos e lidar com essa limitação. No contexto de LPS, este problema ocorre durante a seleção de um subconjunto de produtos para teste. De acordo com Oster et al [36], um critério de teste de LPS deveria cobrir a maior quantidade possível de interações entre características diferentes, aumentando a possibilidade de encontrar defeitos.

Com essa motivação alguns trabalhos surgiram e propõem critérios de teste para seleção de produtos a serem testados [10, 27, 30, 37], a maioria deles baseados em teste *pair-wise*. A ideia é selecionar produtos para cobrir todos os pares de interação entre características. Outra linha de investigação [6] propõe o uso de teste de caminhos básicos geralmente aplicado no teste estrutural de programas.

Entretanto, observa-se que nenhuma das propostas considera os defeitos que podem estar presentes no diagrama de características para selecionar os produtos, ou seja, os critérios propostos não são baseados em defeitos.

1.2 Justificativa

Considerando o contexto acima observam-se as seguintes motivações que justificam o presente trabalho:

- O diagrama de características é um artefato principal no desenvolvimento da LPS e bastante utilizado para a geração de dados de teste, entretanto sujeito a defeitos.
- O teste de todas as combinações de características, produtos, é impraticável, e somente um subconjunto destes produtos pode ser testado.
- As técnicas de teste existentes que consideram o diagrama de características geram requisitos de teste e medidas de cobertura considerando combinações (teste *pair-wise*) de produtos obtidas no diagrama e não consideram os possíveis defeitos no diagrama de características.
- A aplicação de critérios de teste baseados em defeitos, tal como o critério Análise de Mutantes, no diagrama de características, além de fornecer uma medida de análise da atividade de teste, pode levar a um aumento da eficácia em termos do número de defeitos encontrados.

1.3 Objetivo

Este trabalho tem como objetivo geral propor uma abordagem para realizar o teste de mutação em diagramas de características para LPS. Para isso têm-se os seguintes objetivos específicos: (i) identificar classes de defeitos associadas ao diagrama, que serviram de base para a introdução de um conjunto de operadores de mutação. Estes operadores descrevem defeitos que podem estar presentes no diagrama de características devido a enganos cometidos pelos desenvolvedores; (ii) propor um processo de teste para aplicação dos operadores, para que os mutantes possam ser exercitados e para o cálculo do escore de mutação; (iii) implementar o processo em uma ferramenta de suporte; e, (iv) condução de experimentos de avaliação e comparação com outras abordagens existentes, no caso o teste combinatorial *pair-wise*.

1.4 Organização do Trabalho

No Capítulo 2 são introduzidos alguns conceitos relativos à linha de produto de software, suas particularidades, características, vantagens e desvantagens. Além disso, é descrita a notação para o diagrama de características de uma linha de produto de software utilizada no trabalho. O Capítulo 3 apresenta conceitos de teste de software assim com as três diferentes técnicas de teste, sendo estas o teste funcional, teste estrutural e o teste baseado em defeitos. Esta última recebe maior atenção uma vez que é o foco deste trabalho. No Capítulo 4, o enfoque é dado ao teste de linha de produto de software. Este capítulo apresenta uma breve descrição sobre os trabalhos relacionados. O Capítulo 5 apresenta a proposta do trabalho desenvolvido, descrevendo inicialmente as classes de defeitos para o modelo de características, os operadores de mutação, como esses operadores são aplicados, como é obtido o escore de mutação, e é introduzido um processo de teste. O Capítulo 6 apresenta a ferramenta desenvolvida para auxiliar a aplicação do processo de teste, descrevendo seus aspectos de implementação. O Capítulo 7 apresenta um experimento de avaliação da abordagem de teste proposta utilizando a ferramenta de auxílio e os resultados para quatro LPS assim como uma comparação com os resultados obtidos pela avaliação de produtos gerados utilizando o algoritmo AETG para o teste *pair-wise*. Por fim, no Capítulo 8 é feita uma conclusão do trabalho proposto e são apresentadas propostas de trabalhos futuros.

Este trabalho contém também três apêndices. O Apêndice A, contém as opções de comandos da aplicação e exemplos de execução da ferramenta implementada. O Apêndice B contém os diagramas de características para as LPS avaliadas no Capítulo 7. E o Apêndice C, apresenta os conjuntos de *pair-wises* de variabilidades dos diagramas de características avaliados.

CAPÍTULO 2 – LINHA DE PRODUTO DE SOFTWARE

Uma linha de produto de software (LPS) pode ser definida como um conjunto de sistemas que compartilham um conjunto de características comuns e gerenciadas. Tais características são desenvolvidas, de forma planejada, a partir de um núcleo de artefatos previamente definidos e atendem aos requisitos específicos de um seguimento de mercado [35]. Características são atributos de um sistema de software que afetam diretamente os usuários finais e costumam ser representadas no modelo de características. Um produto de uma LPS é obtido pela combinação das suas características.

A engenharia de LPS é um processo que tem por finalidade fornecer componentes reutilizáveis, os quais podem ser utilizados para desenvolver novas aplicações dentro do mesmo domínio. O propósito é reduzir o tempo e os custos de produção e aumentar a qualidade do software por reutilizar elementos que já foram testados e validados [19].

Segundo Clements e Northrop [7] as três atividades básicas e necessárias para o desenvolvimento de uma LPS são:

- Desenvolvimento do núcleo de artefatos (*core assets*);
- Desenvolvimento do produto;
- Gerenciamento da linha de produto de software.

Essas três atividades estão diretamente relacionadas de tal forma que a alteração em uma delas pode acarretar em impacto nas demais.

O processo geral de uma linha de produto de software é baseado na reutilização de componentes, arquitetura e requisitos, e é composto por duas etapas. A primeira denominada Engenharia de Domínio, ou desenvolvimento para o reúso, que consiste em desenvolver um núcleo de artefatos através da análise, projeto e implementação do domínio. A segunda etapa, chamada de Engenharia de Aplicação, ou desenvolvimento com reúso, consiste no desenvolvimento dos produtos finais, utilizando o núcleo de artefatos e os requisitos específicos dos clientes. Esta fase por sua vez é composta por três processos: (1) requisitos de aplicação; (2) projeto de aplicação; e, (3) codificação [34].

Em detalhes, o estudo do domínio fornece um conjunto de requisitos de referência, os quais podem ser utilizados para definir os requisitos de uma aplicação e explicitar a necessidade de integrar novos requisitos. A arquitetura de referência, definida pelo projeto do domínio é utilizada para desenvolver e estruturar as

aplicações. Por fim, componentes reusáveis são utilizados para codificar as aplicações.

Durante o desenvolvimento do núcleo de artefatos, conexões bidirecionais devem ser estabelecidas entre os requisitos de referência, a arquitetura de referência e os componentes reutilizáveis para facilitar o gerenciamento de mudanças e atualizações na linha de produto de software [19]:

- Engenharia de Domínio: A engenharia de domínio é responsável pelo desenvolvimento do núcleo de artefatos da LPS. Esta fase é chamada de processo de desenvolvimento para o reúso e segue o modelo clássico de desenvolvimento de software. Em termos de processo, o desenvolvimento do núcleo de artefatos utiliza diversas entradas, como por exemplo, restrições de produto, de produção e estratégias de produção. Esta pode também fazer uso de diversos componentes pré-existentes, como, estilos, padrões, *frameworks*, e tem como saídas o escopo da LPS, o núcleo de artefatos e o plano de produção.
- Engenharia de Aplicação: A engenharia de aplicação é responsável pelo desenvolvimento dos produtos. Esta fase é chamada de processo de desenvolvimento com reúso e também segue o modelo de desenvolvimento de software em cascata, reutilizando o núcleo de artefatos previamente definidos na engenharia de domínio. O desenvolvimento dos produtos finais é baseado no núcleo de artefatos e nos requisitos especificados pelo cliente.

Em teoria uma LPS pode apresentar diversas vantagens, em [19] são listadas algumas:

- Melhor entendimento do domínio;
- Facilidade em treinar pessoal;
- Alta qualidade dos produtos;
- Confiança do cliente;
- Reaproveitamento de componentes e requisitos;
- Melhor análise dos requisitos;
- Melhor controle de qualidade;
- Redução de custos de produção e manutenção.

Em [21], algumas estatísticas concretas da indústria são exibidas para ilustrar as melhorias provenientes da adoção de LPS:

- A Nokia aumentou a capacidade de quatro para entre 25 e 30 modelos de telefone por ano, depois da adoção da abordagem da LPS;

- A Cummins, Inc., reduziu o tempo de produção de software para motores a diesel de um ano para aproximadamente uma semana;
- A Motorola observou um aumento em 400% na produtividade em uma das suas famílias de produtos;
- A HP reportou uma redução em sete vezes no tempo de entrega de seus produtos e um aumento em seis vezes na produtividade em uma de suas linhas de sistemas para impressoras.

Apesar de comprovadas pela indústria as vantagens que uma LPS pode trazer, alguns pontos devem ser considerados previamente à criação e adoção de uma LPS. Heymans e Trigaux [19] ressaltam que uma linha de produto de software requer grandes investimentos, riscos e impactos organizacionais e gerenciais. Por exemplo, a adoção de uma LPS requer uma mudança no foco gerencial, uma vez que os resultados serão a longo prazo.

Essa mudança organizacional de desenvolvimento, da visão de um único produto para a abordagem em LPS, pode, já de início, gerar uma mentalidade de resistência a mudanças. Outro problema deve-se a falta de engenheiros com uma visão global da arquitetura de linha de produto de software, falta de diretrizes, técnicas e ferramentas para representar, desenvolver e validar uma LPS.

De fato o desenvolvimento baseado em linha de produto de software está cercado por diversos riscos técnicos e organizacionais que são ressaltados quando comparados ao modelo tradicional de desenvolvimento. A abordagem em LPS diferencia-se da tradicional principalmente pelo prazo de retorno de investimento, uma vez que seus benefícios não são imediatos. Como apresentado na Figura 2.1 deve ser levado em conta também durante a análise do domínio a quantidade de produtos que devem ser desenvolvidos para que sejam visíveis as vantagens reais de uma LPS [19].

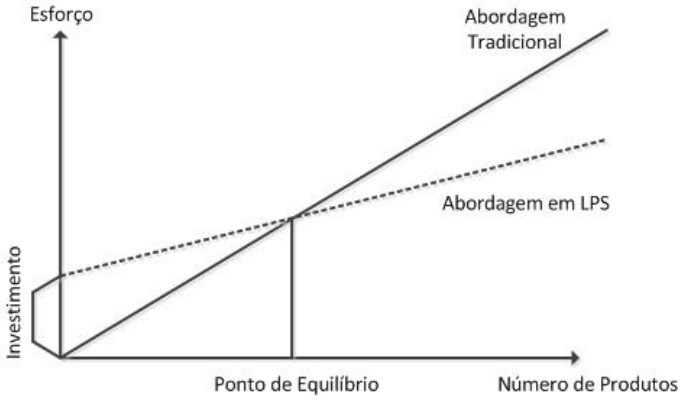

Figura 2.1 Desenvolvimento em Linha de Produto de Software (adaptado de [19]).

2.1 Variabilidade

Dentro de uma LPS as variabilidades ocupam um papel central, uma vez que são elas que mostram como uma linha de produto de software possibilita e facilita a diferenciação entre os seus produtos. De fato, a variabilidade está presente em todos os ciclos de desenvolvimento, desde a análise de requisitos à definição da arquitetura, componentes, codificação e testes [26].

O conceito de variabilidade está diretamente relacionado com outro conceito de fundamental importância dentro de uma LPS, o conceito de elementos comuns. Elementos comuns consistem em uma lista estruturada de suposições que são verdadeiras para cada um dos membros da família de produtos. Segundo [26] variabilidade consiste em uma suposição de como membros de uma família são diferenciados dos outros. Ou seja, variabilidades especificam particularidades de um sistema que correspondem às expectativas do cliente.

Usualmente variabilidade é descrita em pontos de variação e variantes. Um ponto de variação determina uma variabilidade e suas ligações descrevendo diversas variantes. Cada variante corresponde a uma alternativa projetada para realizar uma determinada variabilidade. Em outras palavras, um ponto de variação determina pontos de inserção de variantes e determinam as características da variabilidade [19].

Dificuldades podem ocorrer ao gerenciar variabilidade em LPS porque ambos, produto e família de produtos podem sofrer alterações. Produtos podem evoluir durante o tempo como no modelo clássico de desenvolvimento através de novos *releases*. Entretanto, podem evoluir seguindo as escolhas das variantes no produto.

Bachmann e Bass [2] sugerem uma classificação para as variabilidades de acordo com alguns critérios:

- Variabilidade em função: significa que uma função particular pode existir em alguns produtos e não em outros;
- Variabilidade em informação: significa que uma estrutura particular de informação pode variar de um produto para outro;
- Variabilidade em fluxo de controle: significa que um determinado padrão de interação pode variar de um produto para outro;
- Variabilidade em tecnologia: está relacionada com a plataforma que pode variar resultando em produtos similares em função, mas diferente em aspectos técnicos;
- Variabilidade em metas de qualidade de produto: significa que as metas de qualidade, tais como segurança, desempenho ou capacidade de sofrer alterações podem variar de um produto para outro.

2.2 O Modelo de Características

Como mencionado na introdução muitas metodologias de LPS incluem o modelo de características (FM). Este modelo define combinações válidas de características em um domínio. O modelo de características é visualmente representado por estruturas semelhantes a uma árvore nas quais os nós representam características e as arestas ilustram o relacionamento entre elas. Tal modelo é utilizado para ilustrar como as características são utilizadas para a construção de produtos, sendo estes caracterizados pelas características que contêm [42].

Introduzido inicialmente em 1990 como parte do método de desenvolvimento FODA como um meio de representar características comuns e variabilidades em famílias de sistemas o modelo de característica sofreu uma série de extensões como tentativas de torná-lo mais sucinto e natural [41]. Entretanto, existe um consenso de que ao menos um modelo de características deve ser capaz de representar os seguintes relacionamentos entre características:

- Mandatórias: Se uma característica filha é mandatória, esta deve ser incluída em todos os produtos em que a característica pai aparecer;
- Opcionais: Se uma característica filha é definida como opcional, esta pode opcionalmente ser incluída em produtos em que a característica pai aparecer;
- Alternativas: Um conjunto de características é definido como alternativo se apenas uma das características pode ser selecionada quando a característica pai é parte do produto;
- Relação ou: Uma relação entre um conjunto de características é definido como uma relação ou se uma ou mais características podem ser incluídas quando a característica pai é parte do produto.

Note que uma característica filha somente pode aparecer em um produto se este contém a característica pai. A característica raiz deve ser parte de todos os produtos dentro da LPS. Além dos relacionamentos de hierarquia entre características, o modelo deve poder também representar relacionamentos de restrição entre características através da árvore (relações diferentes da de parentesco) [41]. Essas restrições são normalmente na forma de:

- Incluir: Se uma característica A inclui uma característica B, a inclusão da característica A em um produto implica na adição da característica B no mesmo produto;
- Excluir: Se uma característica A excluí uma característica B, ambas as características não podem ser parte de um mesmo produto.

A semântica de um modelo de características é o conjunto de características que o modelo permite. A abordagem mais comum é utilizar lógica matemática para capturar a semântica do diagrama de características. Cada uma das características é representada por uma variável booleana e a semântica é capturada por lógica proposicional. A avaliação que satisfaz tal fórmula corresponde às configurações de características permitidas pelo diagrama [41].

A Figura 2.1 representa um meta-modelo, segundo [3], para um diagrama de características. Pela representação, observa-se que um modelo de características é composto por quatro elementos principais, características, relações, restrições e cardinalidades, sendo estas especializadas de acordo com os elementos existentes em um diagrama. Desta forma, a característica pode ser uma raiz (*root*), uma característica solitária (*solitary*) ou agrupada (*grouped*), além disso, características possuem os atributos nome e domínio. Relações podem ser binárias (*binary*) ou relações agrupadas (*set*) e, por fim, restrições podem ser do tipo: incluir (*depends*); ou, excluir (*excludes*). Também pelo diagrama, tem-se que cardinalidades são aplicadas a características solitárias ou em relações agrupadas. E que relações binárias são formadas por características solitárias enquanto relações agrupadas possuem características agrupadas. Entretanto, uma característica solitária pode ser composta por relações binárias e por agrupadas.

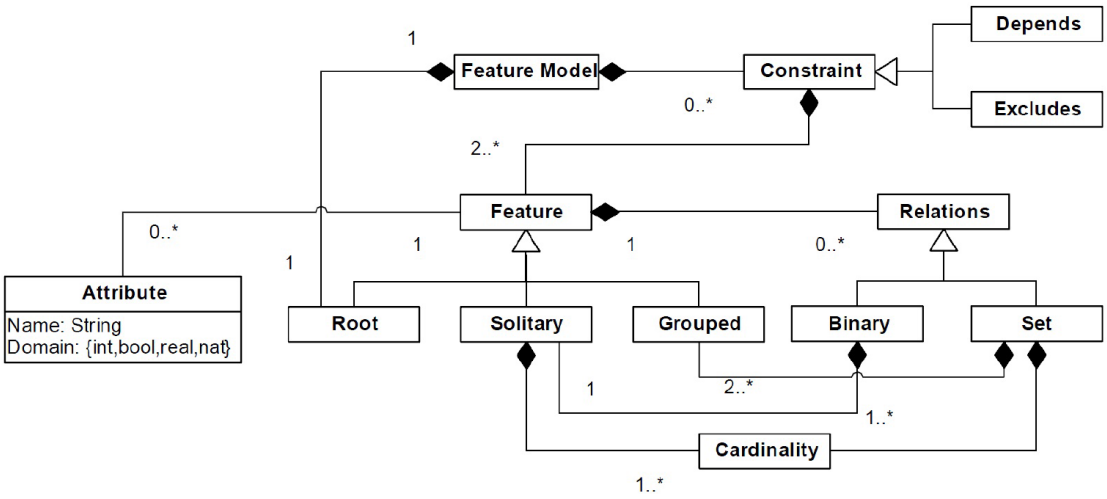

Figura 2.2 Meta modelo de um diagrama de características [3].

2.4 Um exemplo de Linha de Produto de Software

Em [46] é apresentada uma linha de produto de software para controle de Sistema de Som Automotivo. Uma versão modificada[1] dessa LPS é apresentada na Figura 2.3. Esta versão será utilizada como exemplo ao longo deste trabalho.

Um sistema de som automotivo contém diversas possibilidades de características, muitas das quais são comuns a todas as variações de produtos. Em geral, os controles básicos são comuns a todos os rádios: Um sistema de áudio automotivo deve prover uma opção de mudar seu modo, por exemplo, alternar entre rádio e o tocador de CD. Ele deve possibilitar que o usuário selecione os canais ou títulos, altere o volume, procure novos canais ou novos títulos. Dependendo do modo em uso e dos eventos recebidos os controles básicos como avançar ou voltar podem ter diferentes funções.

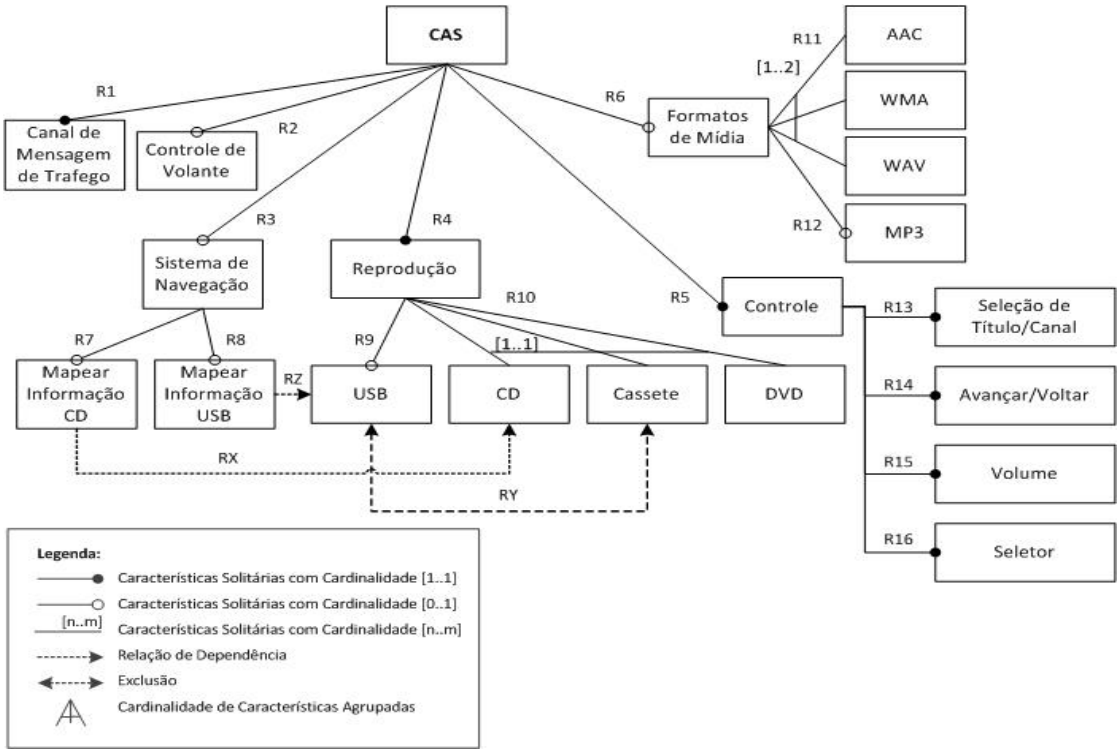

Figura 2.3 Modelo de características de Sistema de Som Automotivo (adaptado de [46]).

[1] As modificações introduzidas permitem a representação de todos os relacionamentos possíveis em um diagrama de características e possuem caráter ilustrativo, e não necessariamente representando uma realidade. As alterações são: Adição de uma relação de exclusão entre as características cassete e USB pertencentes ao ponto de variação Reprodução; Adição de uma característica DVD ao ponto de variação Reprodução e pertencente a escolha alternativa entre CD e Cassete; Adição de um novo ponto de variação Formatos de Mídia; Adição das variantes AAC, WMA, WAV e MP3 ao ponto de variação Formatos de Mídia; Adição de uma relação ou entre as características AAC, WMA e WAV.

Além disso, a escolha de tocar CD pode também influenciar seu comportamento. Canal de Mensagem de Tráfego é uma característica comum a todos os sistemas de som automotivos. Enquanto que, a disponibilidade da porta USB, do Controle de Volante e do Sistema de Navegação são características opcionais. Dependendo da mídia em reprodução, a atualização das informações do mapa do sistema de navegação pode ser opcional.

Na LPS exemplo podem ser observadas todas as possíveis relações entre elementos de um diagrama de características, tais como: (1) Obrigatórias ou características comuns, representadas pela relação entre CAS e Canal de Mensagem de Tráfego; (2) Opcionais representadas por Rádio e Controle de Volante na qual Controle de Volante não necessariamente precisa estar contido em todos os produtos que contiverem a característica CAS; (3) Relação OU, representada pela relação das características AAC, WMA e WAV com Formatos de Mídia, nesta relação a quantidade de características que devem estar presentes em um produto deve respeitar os limites estabelecidos, ou seja, no mínimo zero e no máximo duas destas características; (4) Escolha alternativa, representada pela relação entre CD, Cassete e DVD com a característica Reprodução, nesta relação apenas umas destas características deve acompanhar a característica Reprodução em um produto; (5) Inclusão, representada pela relação entre Mapear informação CD e CD, que garante que sempre que a característica Mapear Informação CD estiver presente em um produto esta deve ser acompanhada da característica CD; e (6) Exclusão, representadas pela relação entre Cassete e DVD, que define que as duas características em questão nunca poderão estar presentes em um mesmo produto.

2.5 Análise Automatizada de Modelos de Características

Análises automatizadas do modelo de características permitem um processo auxiliado pelo computador para extração de informação do modelo. Desta informação, estratégias de marketing e decisões técnicas podem ser derivadas. Em [41] são exemplificadas algumas operações válidas dentro desta análise.

- Determinar se um modelo está vazio: Essa operação tem como entrada o modelo de característica e retorna um valor qualquer representando se modelo está vazio ou não;
- Descobrir se um produto é valido: Esta operação verifica quando um produto de entrada pertence ou não ao conjunto dos produtos representados por um determinado modelo de características;

- Obter todos os produtos: Esta operação toma como entrada um modelo de característica e devolve todos os produtos que podem ser representados pelo modelo;
- Calcular o número de produtos: Esta operação retorna o número de produtos representados por um dado modelo de características;
- Calcular o coeficiente de variabilidade: Esta operação recebe como entrada um modelo de características e retorna a razão entre o número de produtos e $2^n - 1$ onde n é o número de características do modelo;
- Calcular o coeficiente de elementos comuns: Esta operação recebe um modelo e uma característica como entrada e retorna um valor representando a proporção de produtos válidos nos quais a característica está presente.

Estas análises podem ser executadas utilizando diferentes abordagens, por exemplo, transformando o modelo de características em especificações de lógica proposicional, descrições lógicas ou por algoritmos de restrições. Diversas são as ferramentas comerciais e de código livre que implementam estas análises, como por exemplo, AHEAD Tool Suit[2], FAMA Framework[3], Feature Model Plug-in[4] e pure::variants[5].

O framework FAMA [4] contém conjunto de funções de verificação e validação de diagramas de características e produtos que devido a sua extensibilidade é facilmente acoplado a outras aplicações. Entre os comandos e operações disponíveis no FAMA vale ressaltar as seguintes, que serão utilizadas neste trabalho: (i) validação de um diagrama de características informando se um diagrama de característica é vazio ou não; (ii) validação de produtos, ou seja, a verificação se um determinado produto pertence ou não ao conjunto de produtos representados por um diagrama de características dado; (iii) geração de todos os produtos que podem ser representados por um diagrama de características; (iv) apresentação do número de produtos representados por um diagrama de características.

2.6 Considerações Finais

LPS é um conjunto de sistemas que compartilham um conjunto comum de características gerenciadas que satisfazem os requisitos específicos de um domínio, e que são desenvolvidos a partir de um núcleo de forma planejada. Entre as vantagens

[2] http://www.cs.utexas.edu/users/schwartz/ATS.html
[3] http://www.isa.us.es/fama/
[4] http://gp.uwaterloo.ca/fmp/
[5] http://www.pure-systems.com/

de uma LPS estão alta qualidade dos produtos, melhor análise de requisitos e principalmente redução de custos de produção e manutenção. Em contrapartida as desvantagens são os grandes investimentos e riscos iniciais. Uma LPS é composta de um núcleo de artefatos, que são características compartilhadas por todos os produtos, e por variantes.

Pode-se observar que a maioria das metodologias de desenvolvimento em LPS utiliza o diagrama de características, que pode ser considerado um artefato fundamental para LPS. Este artefato tem sido bastante utilizado na fase de teste de LPS, objeto de estudo dos próximos capítulos.

CAPÍTULO 3 - TESTE DE SOFTWARE

A construção de software definitivamente não é uma atividade simples e dependendo das características e dimensões do sistema a ser criado pode se tornar extremamente complexa. Desta forma, está sujeita a diversos e diferentes tipos de problemas que acabam resultando na obtenção de um produto final diferente do que se esperava [13]. Diversos são os fatores que podem contribuir para causar tais problemas, mas a maioria tem uma mesma origem: o erro humano. Como a maioria das atividades de engenharia, a construção de software depende basicamente da habilidade, da interpretação e da execução das pessoas que o constroem; desta forma, erros acabam surgindo, mesmo com a utilização de métodos e ferramentas de Engenharia de Software [15].

De forma que estes erros sejam descobertos antes de o software ser liberado para utilização, existe uma série de atividades chamadas de Validação, Verificação e Teste (VV&T), que têm por finalidade garantir que tanto o modelo pelo qual o software está sendo construído quanto o produto estejam em conformidade com a especificação. Define-se então a atividade de VV&T como um elemento crítico de garantia de qualidade de software representando a última revisão de especificações, projetos e códigos [13].

Segundo Pressman [38] o destaque crescente do software como elemento de sistemas e os “custos” envolvidos associados às falhas de software são forças propulsoras para que a atividade de teste seja cuidadosa e bem planejada, tornando comum encontrar organizações que gastem pelo menos 40% do esforço total do projeto em testes.

3.1 Conceitos da Atividade de Teste

Segundo Mayers [32], teste é o processo de avaliar um programa com a intenção de encontrar erros. Isso se deve ao fato de diferentemente do processo da engenharia de software, o teste de software poder ser visto como um processo destrutivo uma vez que o engenheiro deve criar uma série de casos de teste com a intenção de encontrar problemas no sistema. Fazendo com que o engenheiro supere o conflito de interesse quando os erros são descobertos, uma vez que este foi quem projetou o sistema.

Myers [32] aponta também uma série de outros aspectos relativos ao teste software:

- A atividade de teste é o processo de executar um programa com a intenção de descobrir erros;
- Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto;
- Um teste bem sucedido é aquele que revela um erro ainda não descoberto.

Os objetivos apontados em [12] implicam em uma mudança drástica de ponto de vista. Eles contrariam a ideia comumente defendida de que um teste bem sucedido é aquele em que nenhum erro é encontrado. Assim casos de teste que descubram de forma sistemática diferentes classes de erros com o menor esforço e o menor tempo devem ser projetados. Como objetivo secundário, a atividade de teste de software demonstrará que as funcionalidades do software aparentemente estão trabalhando de acordo com as especificações e que os requisitos aparentemente foram cumpridos. Além é claro, de servir como uma boa indicação de confiabilidade de software e alguma indicação de qualidade do sistema como um todo [38]. Entretanto, existe algo que a atividade de teste não pode garantir: A atividade de teste não pode mostrar a ausência de defeitos, ela só pode mostrar que defeitos estão presentes.

Em [20] são apresentadas definições adotadas na literatura para os principais conceitos relacionados à atividade de teste:

- Defeito (do inglês *fault*): passo, processo ou definição de dados incorretos;
- Engano (*mistake*): ação humana capaz de produzir um defeito;
- Erro (*error*): estado inconsistente ou inesperado de um programa durante a sua execução;
- Falha (*failure*): produto de um resultado incorreto produzido pela execução do programa com relação ao resultado esperado.

Além dos conceitos citados, alguns outros termos relacionados aos dados utilizados durante a atividade de teste se tornam importantes. Domínio de entrada de um programa *P* é definido como o conjunto de todos os valores possíveis que podem ser utilizados para executar *P*. Da mesma forma, dado de teste é definido como um dos elementos que pertence ao domínio de entrada do programa. E caso de teste é um par formado pelo dado de teste e o resultado esperado para execução do programa para aquele dado de teste. Por fim, conjunto de teste é o conjunto de todos os casos de teste utilizados durante determinada atividade de teste [13].

3.2 Fases da Atividade de Teste

Devido à diversidade dos fatores que podem colaborar para a ocorrência de erros, a atividade de teste se torna extremamente complexa. Desta forma ela é dividida em fases com objetivos distintos. São fases da atividade de teste: teste de unidade; teste de integração; e o teste de sistema [13].

O teste de unidade tem como foco as menores unidades de um programa, por exemplo, funções, procedimentos, métodos ou classes. Tem como objetivo identificar os erros diretamente relacionados a algoritmos incorretos ou mal interpretados, estruturas de dados incorretas, ou simples erros de programação [13].

O teste de integração deve ser executado após a finalização do teste de unidade, enfatizando a construção da estrutura do sistema. À medida que as diversas partes do software são colocadas para trabalhar juntas, se torna necessário verificar se a interação entre elas funciona de maneira adequada [13].

Por fim, o teste de sistema tem por finalidade verificar se todas as funcionalidades foram implementadas corretamente de acordo com a especificação do sistema. Aspectos de correção, completude e coerência devem ser explorados, bem como os requisitos não funcionais, tais como, segurança, desempenho e robustez. De forma geral, a maioria das organizações prefere utilizar uma equipe independente para realizar o teste de sistema [13].

Independentemente da técnica de teste, existem etapas bem definidas para a execução da atividade: 1) Planejamento do teste; 2) Projeto de casos de teste; 3) Execução dos casos de teste; e 4) Análise dos resultados obtidos.

3.3 Técnicas e Critérios

Segundo [13] o que distingue essencialmente as técnicas de teste é a fonte utilizada para definir os requisitos da atividade de teste. Além disso, cada um dos critérios de teste procura explorar determinados tipos de defeito, estabelecendo requisitos de teste para os quais valores específicos do domínio de entrada do programa devem ser definidos com o intuito de exercitá-los.

O Teste Funcional é uma técnica amplamente utilizada para projetar casos de teste na qual o programa é considerado uma caixa preta, desta forma, para testá-lo são fornecidas entradas e avaliadas as saídas verificando se estas estão em conformidade com os objetivos especificados. A principal característica desta técnica é

desconsiderar os detalhes de implementação avaliando o software segundo o ponto de vista do usuário e suas funcionalidades [13].

O Teste Estrutural estabelece os requisitos de teste com base na estrutura interna do programa, requerendo a execução de suas partes ou componentes elementares. Os caminhos lógicos do software são testados, fornecendo-se casos de teste que põem à prova tanto conjuntos específicos de condições e/ou laços bem como pares de definições e usos de variáveis. De modo geral, os critérios pertencentes à técnica estrutural são classificados com base na complexidade, no fluxo de controle e no fluxo de dados [13].

A técnica estrutural apresenta algumas limitações e desvantagens decorrentes das limitações inerentes à atividade de teste. Tais aspectos introduzem diversos problemas na automatização do processo de validação de software, por exemplo, é indecidível se uma determinada sequência de comandos de um programa é executável ou não. Se todos os caminhos que cobrem um elemento requerido por um determinado critério estrutural forem não executáveis, o elemento será também não executável e o critério não poderá ser completamente satisfeito. Outra limitação é a existência de caminhos ausentes. Se o programa não implementa uma determinada função, não existirá um caminho que corresponda à função dada, consequentemente, nenhum dado de teste será requerido para exercitá-la;

O Teste Baseado em Defeitos utiliza tipos de defeitos conhecidos para gerar requisitos de teste. A motivação desta abordagem está nos possíveis erros que programadores e projetistas podem cometer durante o processo de desenvolvimento do software. Dentre os principais critérios desta técnica destacam-se a Semeadura de Erros e Análise de Mutantes nos quais os casos gerados são específicos para demonstrar a presença ou ausência de defeitos.

Proposto em [14] o critério Análise de Mutantes baseia-se na hipótese do programador competente, assumindo que programadores competentes escrevem códigos corretos ou próximos de estarem corretos. A partir desta hipótese, pode-se afirmar que defeitos são introduzidos nos programas através de desvios sintáticos, que mudam a semântica do programa levando-o possivelmente a um comportamento incorreto [38]. O critério Análise de Mutantes tem por objetivo identificar estes defeitos sintáticos mais comuns, para identificar erros nos programas.

Outra hipótese proposta em [14], refere-se ao efeito de acoplamento, a qual afirma que defeitos complexos são compostos por defeitos menores e mais simples. Desta forma, um conjunto de casos de teste que seja capaz de identificar um defeito simples também é capaz de revelar defeitos mais complexos.

Para realizar o teste de um programa P utilizando o critério Análise de Mutantes, o testador deve inicialmente aplicar operadores de mutação a P. Estes operadores alteram o programa P original gerando um novo programa P' mutante, que difere de P apenas pela adição de uma única modificação. O programa P original e os diversos programas P' mutantes devem ser executados com o mesmo conjunto de casos de teste T. Os resultados obtidos são analisados e os programas mutantes são agrupados em três conjuntos:

- Mutantes Mortos: para algum caso de teste t pertencente a T o resultado do programa original P e do programa P' são diferentes;
- Mutantes Vivos: para todos os casos de teste de T o programa mutante P' apresenta o mesmo resultado do programa P, então, deve ser analisado para verificar se o mutante é equivalente a P, ou se o conjunto de casos de T precisa ser melhorado para que seja possível matar P';
- Mutantes Equivalentes: não existe caso de teste que diferencie o programa P do programa mutante P', para todos e quaisquer casos de teste os programas apresentam sempre o mesmo resultado.

Após a execução dos mutantes, verifica-se a adequação dos casos de teste através do escore de mutação (em). O escore de mutação é uma medida de avaliação de quanto o conjunto de testes aproxima-se da adequação. Este possui um valor dentro do intervalo $[0..1]$. Quanto mais próximo de 1, mais adequado é o conjunto de testes. O escore pode ser obtido por:

$$em(P,T) = \ (MM(P,T))/(M(P) - ME(P))$$

Onde:

$MM(P,T)$: número de mutantes mortos.

$M(P)$: número de mutantes gerados.

$ME(P)$: número de mutantes equivalentes.

Uma das maiores desvantagens do critério Análise de Mutantes deve-se a seu custo de execução. Uma vez que é gerada uma grande quantidade de programas mutantes e todos os programas devem ser executados com o conjunto de teste, até para programas pequenos este critério demanda um grande tempo computacional para execução. Outra limitação é a existência de mutantes equivalentes. Determinar se dois programas computam a mesma função, ou seja, se são equivalentes, é uma questão indecidível e esta atividade deve ser conduzida manualmente pelos testadores e isto pode acarretar em custo adicional.

3.4 Testes Metamórficos

Um oráculo em teste de software é um procedimento pelo qual testadores podem decidir se a saída de um programa está correta. Em algumas situações, o oráculo pode não estar disponível ou ser muito difícil de ser aplicado e a predição manual e a comparação de resultados é uma tarefa custosa em termos de tempo e propensa a erros. Esta limitação é referenciada em teste de software como o problema do oráculo [50].

O teste metamórfico foi proposto em [8] como uma maneira de endereçar o problema do oráculo. A ideia por trás desta técnica é gerar novos casos de teste baseados na informação de teste existente. A saída esperada de novos casos de teste pode ser conferida pela utilização das chamadas relações metamórficas, isto é, relações conhecidas entre duas ou mais entradas e suas saídas esperadas. Desta forma, uma das vantagens desta técnica está em facilitar a automatização do oráculo.

Em [42] é apresentado um exemplo de teste metamórfico: Considere um programa que calcula a função cosseno $(\cos(x))$. Suponha que o programa produza a saída -0.3999 quando executado com a entrada $x = 42\,radianos$. Uma propriedade importante da função cosseno é $cos(x) = cos(-x)$. Utilizando esta propriedade como uma relação metamórfica, pode-se projetar um novo caso de teste com $x = -42$. Assumindo que a saída do programa para esta entrada seja 0.4235. Pode-se facilmente observar ao comparar ambas as saídas que o programa não está correto e assim automatizar o oráculo.

Testes metamórficos têm se mostrado eficazes em diversos domínios de testes, incluindo programas numéricos, teoria dos grafos ou aplicações orientadas a serviços [42].

3.5 Teste Combinatorial

Teste combinatorial baseia-se na teoria de que defeitos podem ser revelados pela interação de parâmetros do software sendo testado, utilizando um *array* de cobertura de casos de teste gerados através de um mecanismo de amostragem [33]. Experimentos sugerem que alguns defeitos são acionados apenas por combinações de parâmetros não usuais, e em geral, pela interação entre apenas dois [25], o que explica a popularidade de técnicas como o *pair-wise*.

No teste de *pair-wise* todas as possibilidades de pares de combinações de valores de parâmetros são cobertas por pelo menos um caso de teste [25]. Embora o

pair-wise não seja exaustivo, é uma técnica útil visto que esta pode ajudar a identificar defeitos potenciais simples, ocasionados pela interação entre parâmetros, com um conjunto relativamente pequeno de casos de teste [24].

De acordo com o exemplo apresentado em [24], suponha que se deseje testar se uma nova aplicação funciona perfeitamente em computadores que utilizam sistema operacional Windows ou Linux, processadores Intel ou AMD e protocolos IPv4 ou IPv6. Estas configurações resultariam em um total de $2 \times 2 \times 2 = 8$ possibilidades de combinações, entretanto, como apresentado na Tabela 3.1, apenas 4 casos de teste são necessários para testar cada uma das combinações de um componente com outro pelo menos uma vez.

Tabela 3.1 Configurações do teste *pair-wise*

Caso de Teste	Sistema Operacional	CPU	Protocolo
1	Windows	Intel	IPv4
2	Windows	AMD	IPv6
3	Linux	Intel	IPv6
4	Linux	AMD	IPv4

Muito embora os resultados apresentados no exemplo demonstrem uma redução, não muito expressiva, de 8 para 4 casos de teste, os resultados se tornam-se mais animadores quando sistemas maiores são considerados. Como por exemplo, considere um software de automatização de manufatura que contenha 20 controles, cada um com 10 possibilidades de configurações, resultando em um total de 10^{20} combinações, pela utilização de *pair-wise*, esse valor seria reduzido para apenas 180 casos de teste [24].

3.6 Testes Baseado em Modelos

O teste baseado em modelos contrasta-se com o teste baseado em programa. Neste tipo de teste, casos de teste são gerados considerando modelos de especificação da aplicação [16]. Portanto um modelo precisa estar disponível e poder ser de certa forma “executado” a fim de que uma saída seja produzida e possa-se avaliar o seu comportamento com relação ao esperado.

Portanto, um dos maiores desafios do teste em modelos é contar com uma especificação precisa e com modelos mais formalmente especificados [13]. Mas

diversas vantagens podem ser observadas neste tipo de teste, pois como a antecipação do processo de teste uma vez que o modelo pode ser construído durante a definição dos requisitos. De forma semelhante ao teste de programas, critérios de teste funcional, estrutural e baseado em defeitos podem ser utilizados.

Modelos de comportamento tais como máquinas de estado finito têm sido alvo de diversos estudos [13], além destes destacam-se outros modelos, tais como diagramas de casos de uso, diagrama de classes, diagrama de interação entre outros [43].

3.7 Comparação Entre Critérios de Teste

Estudos reportados na literatura [49] mostram que a comparação entre critérios de teste tem sido realizada considerado três fatores principais: (i) a eficácia está diretamente relacionada ao número de defeitos que um critério consegue revelar; (ii) o *strenght*, que mede a dificuldade de satisfação de um critério de teste dado que um outro critério foi satisfeito; e (iii) custo, calculado a partir da quantidade de casos de teste necessários para satisfazer um critério.

3.8 Considerações Finais

A atividade de teste possui muitas limitações para sua completa automatização, entretanto reduzir estas limitações e os custos dessa atividade são ainda desafios de pesquisa. Diversas são as técnicas criadas a fim de se fornecer informações sobre qualidade de um software. Cada técnica de teste possui uma finalidade específica e é mais eficaz em revelar certos tipos de defeitos, por isto elas são consideradas complementares. Outro desafio é a proposição destes critérios para novos contextos de desenvolvimento de software, tais como o desenvolvimento em LPS, tema abordado no próximo capítulo.

CAPÍTULO 4 – TESTE DE LINHA DE PRODUTO DE SOFTWARE

De acordo com [12], alguns estudos classificam os trabalhos de teste de LPS considerando as duas fases primárias na engenharia de linha de produtos de software: engenharia de domínio; e, a engenharia de aplicação. Na engenharia de domínio algumas atividades de teste estão relacionadas ao desenvolvimento de casos de teste e execução de testes com o objetivo de avaliar a qualidade do núcleo de artefatos, que mais tarde serão instanciados na fase de engenharia de aplicação. As duas principais atividades incluem o desenvolvimento de artefatos de teste que possam ser reutilizados de forma eficiente durante a engenharia de aplicação e a aplicação de testes no núcleo de artefatos criados durante a engenharia de domínio. Durante a engenharia de domínio dois tipos de teste podem ser aplicados: (1) teste de unidade, que se concentra em testar as menores unidades de implementação de software; e, o (2) teste de integração, que se preocupa em testar as relações e a integração entre as unidades do sistema.

Na engenharia de aplicação as atividades de teste estão relacionadas à seleção e instanciação de artefatos para construir casos de teste específicos. São alguns dos tipos de testes que podem ser executados nesta etapa: (1) teste de sistema, que tem por objetivo assegurar que o produto final está de acordo com o especificado; e o (2) teste de aceitação, que consiste na execução geralmente pelo cliente de um roteiro de teste no sistema.

Em seu trabalho, Engström e Runeson [17] apontam os principais desafios ao se testar uma LPS. Um destes desafios está relacionado com o grande número de testes requeridos. De forma a testar completamente uma LPS, todas as possibilidades de uso de cada componente do núcleo de artefatos e cada variabilidade deveriam ser testados. Entretanto o fato do número de variações de produtos crescer exponencialmente com o número de variantes torna o teste exaustivo inviável. Este desafio é o que mais se relaciona com o objetivo do presente trabalho e com o objetivo dos trabalhos relacionados existentes que visam o teste de variabilidades em uma LPS. O teste de variabilidades em uma LPS é apresentado na Sessão 4.2.

O segundo maior desafio, diretamente relacionado com o primeiro, ocorre em saber como balancear o esforço de teste entre os componentes reusáveis e as instâncias da LPS. Tendo assim como questão chave identificar quais componentes deveriam ser testados na engenharia de domínio, e quais deveriam ser testados na engenharia de aplicação.

Outro ponto importante está em como lidar com variabilidades, levantando a questão de como diferentes tipos de pontos de variações devem ser testados. Deste modo, torna-se importante a verificação da não existência de relações incorretas entre pontos de variação, ou seja, que características indesejadas não estejam presentes em produtos. Além disso, a completa integração e teste de sistema na engenharia de domínio geralmente não é possível uma vez que a implementação das variações ainda não está disponível [30].

Da forma semelhante ao teste seguindo o modelo tradicional de desenvolvimento de software, teste em LPS requer um planejamento cuidadoso e um processo bem definido [30]. Geralmente o teste de linha de produto de software é realizado de acordo com o modelo V [45], apresentado na Figura 4.1. Artefatos produzidos durante a engenharia de domínio e de aplicação são testados segundo os níveis do modelo V.

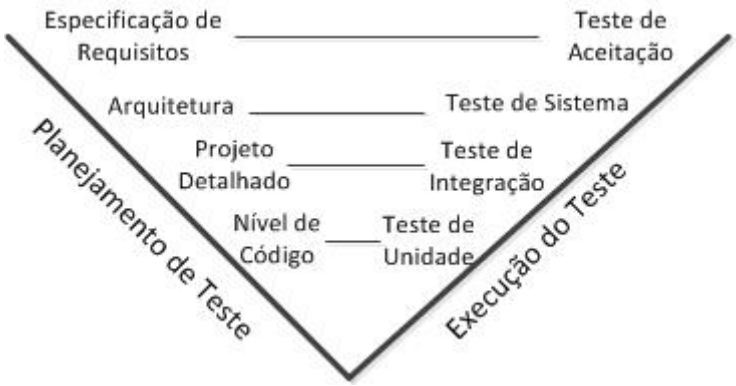

Figura 4.1 O Modelo V (adaptado de [10]).

O lado esquerdo do modelo V inclui os níveis de abstração que são comuns no software. Neste lado, testes são projetados durante o projeto e implementação do sistema. Os modelos de projeto podem também ser validados pela utilização de inspeções guiadas. O lado direito do modelo une as fases de teste aos níveis de abstração da implementação. Neste lado, os testes projetados são executados.

4.1 Estratégias de Teste de LPS

Tenvalinna *et al.* [45] descrevem algumas estratégias para teste de linha de produto de software. A primeira e a segunda estratégias são adaptações de estratégias de teste de sistemas orientados a objetos, enquanto que as duas últimas fora desenvolvidas especialmente para o contexto de LPS.

- Teste de Produto por Produto: Abordagens tradicionais de teste de software são aplicadas nesta estratégia, uma vez que os produtos são testados separadamente. Nesta abordagem as características de reutilização de componentes de LPS não são exploradas para reduzir os esforços da atividade de teste tornando-a extremamente custosa [43]. Entre as grandes vantagens da abordagem de produto por produto estão a qualidade do software obtido e a simplificação do gerenciamento dos testes [39];
- Teste Incremental de Família de Produtos: Nesta abordagem ciclos completos do modelo V são aplicados no primeiro produto a ser testado, e para os demais produtos, apenas técnicas de teste de regressão são aplicados. Ainda seguindo o modelo V, as características da LPS também podem ser testadas [45];
- Teste com Divisão de Responsabilidades: A abordagem de teste com divisão de responsabilidades consiste em dividir as atividades de teste do modelo V entre a engenharia de domínio e a engenharia de aplicação. Enquanto em nível de engenharia de domínio, o componente é testado em nível de código, no nível de aplicação, testes de integração, de sistema e aceitação são executados [45];
- Instanciação de Artefatos Reutilizáveis: Nesta abordagem utilizam-se características de uma LPS ao criar artefatos de teste o mais extensível possível em nível de engenharia de domínio, e em nível de engenharia de aplicação um processo de teste completo é instanciado de acordo com o modelo V [45].

4.2 Teste Baseado em Defeitos em LPS

A abordagem de teste de mutação em LPS foi inicialmente introduzida por Stephenson *et al.* [44], no qual dois problemas são apresentados ao se testar uma família de produto: (1) Quanto as características devem ser testadas antes de serem agrupadas em um produto? (2) como saber se um produto corresponde aos seus requisitos e como distinguir diferentes produtos de uma mesma família de produtos?

Para resolver o primeiro problema é proposta a realização de teste de unidade nos artefatos centrais, e apenas realizar teste em variantes quando estas estiverem incorporadas a produtos finalizados. Esta abordagem, segundo [44], deve ser mais prática e de menor custo considerando o potencial número de variantes.

Para o segundo problema, de determinar se um produto corresponde aos requisitos de uma LPS, Stephenson *et al.* [44] propõem a utilização do teste de mutação para identificar diferenças superficiais entre instancias de uma LPS. Para ilustrar, é apresentado o exemplo no qual existe o problema de decidir entre duas características, A e B, para compor um produto. Desta forma o conjunto de produtos desta LPS pode ser dividido entre dois grupos, o primeiro no qual a característica A está presente, e no outro a característica B. Então, os casos de teste devem ser gerados tais que os resultados dos testes sejam suficientes para mostrar a diferença entre os dois produtos. Por fim, ambas as saída podem ser comparadas com a saída esperada para decidir qual dos dois produtos atende aos requisitos especificados.

Em seu trabalho McGregor *et al.* [31], apresenta uma abordagem para auxiliar no problema de como encontrar a melhor solução para testar uma LPS. Para lidar com este problema, é apresentado um modelo de defeitos, que identifica quais defeitos são prováveis de estarem presentes em uma instância de uma linha de produto de software.

De modo geral, testadores utilizam modelos de defeitos para projetar casos de testes efetivos e eficientes, uma vez que são especificados para procurar por determinados problemas que podem estar presentes. Modelos de defeitos podem também auxiliar na determinação de quanto esforço de teste é suficiente para detectar classes de defeitos específicas. O modelo de McGregor *et al.* [31] é baseado nas principais características de uma LPS nas quais erros podem ser encontrados: características; mecanismos de variação; modelo; e, artefatos centrais. São exemplos de pontos nos quais os defeitos podem ser identificados:

- Defeitos de interface devido ao desenvolvimento distribuído;
- Defeitos em mecanismos de configuração utilizados, ocasionados pela enorme possibilidade de configurações diferentes de artefatos impedindo que todas as combinações sejam testadas;
- Defeitos no processo de tomada de decisão, uma vez que existem diversos pontos de variações e variantes para serem escolhidos;
- Artefatos que envolvem o escopo inteiro da família de produtos podem ser inconsistentes;
- Defeitos no planejamento, podendo este ser muito vago, muito detalhado, incompleto ou não estar claro.

Observa-se que os defeitos propostos por McGregor estão mais relacionados a problemas relativos ao desenvolvimento de software e não a possíveis defeitos presentes e relacionados especificamente às variabilidades descritas em um modelo tal como o FM.

A abordagem de teste de mutação foi utilizada para realizar o teste de produtos que já foram selecionados para o teste, mas não no contexto de teste de variabilidades e com o objetivo de seleção de produtos. Este tópico é tema da próxima seção.

4.3 Teste de Variabilidades

De acordo com [10] existe uma quantidade significativa de trabalho explorando características únicas de abstrações de arquitetura de software com o objetivo de validar sistemas, tais como, detectar incompatibilidade de componentes, planejar teste de integração e utilizar noções de modelo de cobertura de arquitetura de software para adequação de conjuntos de teste. Em contraste, existem relativamente poucos trabalhos que exploram as características únicas de abstrações de LPS, ou seja, a identificação de elementos comuns e variabilidade, para validação.

Como mencionado anteriormente, um dos grandes desafios para validar uma LPS deve-se à interação entre diferentes combinações de relações entre características. Ao utilizar uma LPS, desenvolvedores decidem a respeito das ligações entre características em seu sistema, o que irá definir uma instância da linha de produto de software. Neste processo, é necessário considerar quais combinações de ligações escolhidas aparecem em algum produto. Neste caso, pode-se ter alguma confiança de que as interações entre características foram exercitadas. Mas, caso não apareçam, não se pode garantir que defeitos não poderão aparecer nestas relações.

Idealmente, desenvolvedores de LPS deveriam validar todas as combinações de variabilidades possíveis, aumentando o nível de confiança da LPS. Mas, infelizmente, o espaço de combinações possíveis em uma LPS real pode ser enorme e uma consideração exaustiva destas combinações se tornaria inviável. Desta forma, o problema de garantir cobertura de teste de LPS resume-se então em gerar instâncias da LPS que melhor exercitem as combinações possíveis de ligações entre características.

A maior parte dos trabalhos voltados especificamente para o teste de variabilidades e para a seleção de produtos para teste está baseada no teste de combinação de características obtidas a partir de modelos tais como o modelo de características (FM) e o modelo de variabilidades ortogonal, OVM (*Orthogonal Variability Model*).

Dentre estes trabalhos destaca-se o de McGregor [30] que utiliza *othogonal arrays* e métodos de teste combinatorial para cobrir o espaço de variabilidades da

LPS. Cohen *et al.* [10] propuseram um trabalho que tem como base o OVM, *pair-wise testing* e vetores de cobertura para selecionar os produtos.

Utilizando como critério de cobertura a abordagem de *pair wise*, e vetores de cobertura é obtida à relação de instâncias a serem testadas não considerando restrições entre as relações. Por fim, o modelo relacional é utilizado para excluir combinações de produtos não válidos de acordo com a LPS.

Segundo [11] o teste *pair-wise* é uma técnica efetiva de geração de casos de teste baseado na observação de que a maioria dos defeitos são causados pela interação de pelo menos dois fatores. A geração de casos de teste *pair-wise* garante a cobertura de todas as combinações de duas características. Mesmo com um conjunto de casos de teste muito menor comparado ao teste exaustivo ainda se mostra eficaz em encontrar defeitos.

De forma semelhante, Perrouin *et al.* [37] utilizam teste *t-wise*, no qual são utilizadas combinações de t características. O objetivo é reduzir o número de casos de teste comparado-se à abordagem exaustiva, garantindo ainda uma cobertura razoável de uma LPS. É proposto um conjunto de ferramentas utilizando Alloy para gerar automaticamente os casos de teste e satisfazer o critério *t-wise* de cobertura através de estratégias de divisão e composição para lidar com aspectos de escalabilidade no teste *t-wise*.

Da mesma maneira Lamancha e Usaola [27] garantem a cobertura das características pelo método *pair wise*. Entretanto, ao invés de utilizar um mapeamento relacional de alguma representação do modelo de características, são criadas regras que representam os relacionamentos entre características para a construção de matrizes de pares de características. A partir então destas matrizes e utilizando uma modificação do algoritmo AETG [9] são gerados os produtos que satisfazem os requisitos de cobertura.

De forma simplificada, o AETG é um algoritmo combinatorial para geração de casos de teste que cobrem todas as combinações validas de n parâmetros. O tamanho de um conjunto de casos de teste gerados pelo AETG cresce de forma logarítmica com relação ao número de parâmetros de teste. Esta característica permite a testadores definirem modelos de teste utilizando um conjunto reduzido de casos de teste [9].

Diferentemente das abordagens baseadas em teste combinatorial, o trabalho de Cabral *et al.* [6] introduz uma proposta chamada FIG Basic Path, que gera, a partir de um OVM em teste, um grafo de dependências de características e requer a geração de produtos de tal modo que os caminhos básicos (linearmente independentes) deste grafo sejam executados.

O trabalho de Oster et al. [36] combina projeto de algoritmos de característica combinatorial para geração de características *pair-wise* com testes baseados em modelos para redução do tamanho de uma LPS necessária para a cobertura de interações entre características. Segura *et al*. [42] através da abordagem de teste metamórficos possibilita a geração automática de informações de teste para análise de modelo de características. Pela utilização da ferramenta FAMA para análise automática do diagrama de características em combinação com relações metamórficas propostas, diferentes produtos pertencentes à mesma LPS são gerados para teste a partir de um produto conhecido pertencente a uma LPS [42]. Evitando desta forma a necessidade de se confiar inteiramente em testadores para verificar quando a saída de uma análise está ou não correta.

4.4 Exemplo de Geração de Casos de Teste

Para ilustrar a aplicação de teste *pair-wise* utilizando o algoritmo AETG, considere a versão simplificada da LPS para sistema de som automotivo apresentada na Figura 4.2.

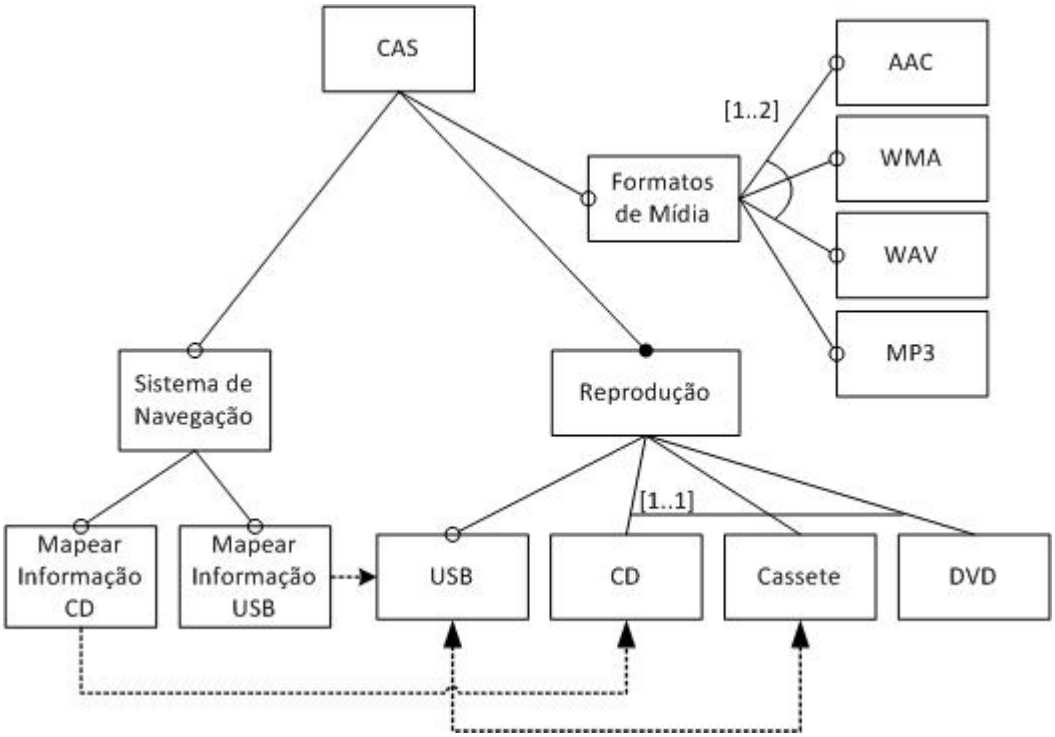

Figura 4.2 LPS de sistema de som automotivo simplificada.

Os conjuntos de combinações de características possíveis segundo o critério de teste *pair-wise*, gerados manualmente, para a LPS exemplo estão representados nas Tabelas 4.1 a 4.6. As combinações não possíveis segundo as restrições impostas pelas relações entre as características não são contempladas nas tabelas.

Tabela 4.1 Pares de combinações entre as sub características de Rádio e Sistemas de Navegação.

Radio - Sistemas de Navegação
Sistema de Navegação - Mapear Informação CD
Sistema de Navegação - Mapear Informação USB
Formatos de Mídia - Mapear Informação CD
Formatos de Mídia - Mapear Informação USB

Tabela 4.2 Pares de combinações entre as sub características de Rádio e Reprodução

Rádio - Reprodução
Sistemas de Navegação - USB
Sistemas de Navegação - CD
Sistemas de Navegação - Cassete
Sistemas de Navegação - DVD
Formatos de Mídia - USB
Formatos de Mídia - CD
Formatos de Mídia - Cassete
Formatos de Mídia - DVD

Tabela 4.3 Pares de combinações entre as sub características de Rádio e Formatos de Mídia.

Rádio-Formatos de Mídia
Sistemas de Navegação - AAC
Sistemas de Navegação - WMA
Sistemas de Navegação - WAV
Sistemas de Navegação - MP3
Formatos de Mídia - AAC
Formatos de Mídia - WMA
Formatos de Mídia - WAV
Formatos de Mídia - MP3

Tabela 4.4 Pares de combinações entre as sub características de Sistemas de Navegação e Reprodução

Sistemas de Navegação - Reprodução
Mapear Informação CD - CD
Mapear informação USB - USB

Tabela 4.5 Pares de combinações entre as sub características de Sistemas de Navegação e Formatos de Mídia.

Sistemas de Navegação-Formatos de Mídia
Mapear Informação CD - AAC
Mapear Informação CD - WMA
Mapear Informação CD - WAV
Mapear Informação CD - MP3
Mapear informação USB - AAC
Mapear informação USB - WMA
Mapear informação USB - WAV
Mapear informação USB - MP3

Tabela 4.6 Pares de combinações entre as sub características de Reprodução e Formatos de Mídia

Reprodução - Formatos de Mídia
USB - AAC
USB - WMA
USB - WAV
USB - MP3
CD - AAC
CD - WMA
CD - WAV
CD - MP3
Cassete - AAC
Cassete - WMA
Cassete - WAV
Cassete - MP3
DVD - AAC
DVD - WMA
DVD - WAV

Para a geração de casos de teste, o algoritmo AETG para geração de produtos implementado pela *Combianorial Tool*[6] foi utilizado. O resultado, considerando apenas variabilidades do diagrama de características, é apresentado na Tabela 4.7. Dentre os produtos gerados é possível observar produtos inválidos segundo a especificação da LPS, por exemplo, os produtos que contêm a característica Mapear Informação CD deveriam também conter a característica CD.

A partir do conjunto de casos de teste gerados pela técnica *pair-wise* é possível observar que estes não exercitariam alguns dos possíveis defeitos que podem estar contidos em um diagrama de características. Por exemplo, a LPS especifica, pela relação escolha alternativa, que um produto deve conter apenas uma das sub características da característica Reprodução pertencentes a essa relação. Entretanto se essa relação tiver sido adicionada erroneamente nenhum dos casos de teste gerados seria capaz de identificar o defeito, uma vez que cada um dos produtos contém apenas uma destas características.

Tabela 4.7 Resultado da execução do algoritmo AETG pela ferramenta *Combinatorial Tool*.

#	**Resultados**
1	Sistema de Navegação ,Mapear Informação CD ,USB ,AAC
2	Sistema de Navegação ,Mapear Informação CD ,USB ,WAV
3	Sistema de Navegação ,Mapear Informação CD ,CD ,AAC
4	Sistema de Navegação ,Mapear Informação CD ,CD ,MP3
5	Sistema de Navegação ,Mapear Informação CD ,Cassete ,WMA
6	Sistema de Navegação ,Mapear Informação CD ,Cassete ,WAV
7	Sistema de Navegação ,Mapear Informação CD ,Cassete ,MP3
8	Sistema de Navegação ,Mapear Informação CD ,DVD ,AAC
9	Sistema de Navegação ,Mapear Informação CD ,DVD ,WMA
10	Sistema de Navegação ,Mapear Informação CD ,DVD ,MP3
11	Sistema de Navegação ,Mapear Informação USB ,USB ,WMA
12	Sistema de Navegação ,Mapear Informação USB ,CD ,WAV
13	Formatos de Mídia ,Mapear Informação CD ,CD ,WMA
14	Formatos de Mídia ,Mapear Informação USB ,USB ,MP3
15	Formatos de Mídia ,Mapear Informação USB ,Cassete ,AAC
16	Formatos de Mídia ,Mapear Informação USB ,DVD ,WAV

[6] http://161.67.140.42/CombTestWeb/

4.5 Considerações Finais

Os trabalhos mencionados para o teste de variabilidade e seleção de produtos oferecem critérios de teste para cobrir a maior quantidade possível de interações entre características, uma vez que o teste da combinação de todas as características se torna impraticável. Entretanto, não é possível garantir que um subconjunto de produtos selecionado seja capaz de revelar todos os defeitos relacionados ao gerenciamento de características. De modo a aumentar a probabilidade de encontrar defeitos e a confiança de que os produtos gerados estejam de acordo com os requisitos de um diagrama de características, é introduzida no próximo capítulo uma abordagem baseada em defeitos utilizando teste de mutação. A abordagem apresentada pode ser utilizada de forma complementar a outras técnicas existentes descritas neste capítulo.

CAPÍTULO 5 – ABORDAGEM BASEADA EM MUTAÇÃO

Como mencionado no capítulo anterior, os trabalhos existentes para selecionar casos de teste, produtos, para uma LPS a partir do diagrama de características não consideram os defeitos que podem estar presentes nestes diagramas. Dado o fato de que o teste de mutação pode aumentar a probabilidade de se revelar defeitos e aumentar a confiança de que os produtos gerados atendem às especificações desejadas, este capítulo apresenta uma abordagem de geração e avaliação de conjuntos de casos de teste baseada em mutação do diagrama de características de uma Linha de Produto de Software. Para isso, são apresentados e classificados os possíveis defeitos identificados em um diagrama de características, são propostos operadores de mutação e por fim, um processo de teste.

5.1 Defeitos no Modelo de Características

Observando o teste baseado em defeitos e fundamentando-se em abordagens de teste de modelos, foram identificadas as seguintes classes de defeitos que podem estar presentes em um diagrama de características.

5.1.1 Atribuição Incorreta da Cardinalidade de uma Característica Solitária

Um defeito desta classe pode ocorrer quando a cardinalidade de uma característica solitária é definida incorretamente. Uma vez que cada uma das características solitárias aceitam diferentes intervalos de valores para cardinalidade, um possível defeito pode ocorrer se, por exemplo, uma característica definida como mandatória recebe cardinalidade com os valores 0 e 1 para os limites inferiores e superiores respectivamente. Neste caso a característica seria incorretamente definida como opcional.

5.1.2 Atribuição Incorreta de Elementos de uma Relação Agrupada

Essa classe de defeitos está diretamente relacionada aos elementos escolhidos para fazer parte de uma relação de agrupamento. Assume-se que uma das características agrupadas não deveria fazer parte de uma relação de agrupamento, ou que uma característica solitária deveria ser incluída na relação de agrupamento.

5.1.3 Existência de uma Relação de Agrupamento

Esta classe compreende defeitos associados a relações de agrupamento incorretamente criadas. Por exemplo, algumas características filhas pertencem a uma relação de agrupamento com uma determinada cardinalidade, entretanto, deveriam ser definidas como características solitárias.

5.1.4 Atribuição Incorreta da Cardinalidade de uma Característica Agrupada

Relações de agrupamento permitem que um dado número de características agrupadas façam parte de um produto criado. Uma vez que a cardinalidade é responsável por definir este número, um engano ao definir os valores mínimos e máximos pode resultar em diagramas que permitam produtos com menos ou mais características do que o especificado.

5.1.5 Restrições Incorretas

Esta classe de defeitos está associada às restrições de dependência e exclusão, e inclui os seguintes casos: (i) uma das características de uma restrição foi incorretamente selecionada; (ii) a restrição não deveria existir; e, (iii) uma restrição está ausente no modelo de características.

5.2 Definições do Meta Modelo de Características Adotado

Os operadores de mutação são definidos baseados em um modelo formal proposto a partir do meta-modelo de [41] apresentado no Capítulo 2. Um modelo de características é denotado por $FM = (F, C, R)$ onde:

- F é o conjunto de características de FM, que é composto pelos subconjuntos $Raiz$, S (características solitárias) e G (características agrupadas). Uma característica $f \in F$ pode ser: (i) $r \in Raiz$; (ii) $s_{min,max} \in S$ na qual a cardinalidade associada a característica solitária é representada por min e max, para o valor mínimo e máximo da cardinalidade respectivamente; e (iii) $g \in G$.
- C é o conjunto de restrições que podem ser do tipo D (restrições de dependência) ou E (restrições de exclusão). Os elementos destes conjuntos são denotados por $d(f_i, f_j) \in D$ e $e(f_i, f_j) \in E$.

- R é o conjunto de todas as relações de FM. R é composto por Bin (conjunto das relações binárias) e Set (conjunto das relações agrupadas). Uma relação binária $b \in Bin$ é composta por uma única característica solitária. Uma relação de agrupamento $st_{min,max}(g_1, \dots, g_n) \in Set$ é composta por $n > 1$ características, onde n é o número de elementos do conjunto. De forma similar a características solitárias, relações de agrupamento comportam uma ou mais cardinalidades, representadas por min e max, para os limites de valores inferiores e superiores respectivamente e com $min \leq max$ e $max \leq n$.

Características mandatórias e opcionais são representadas respectivamente por $b(s_{1,1})$ e $b(s_{0,1})$. Relações baseadas em cardinalidades (clonagem de características) são representadas por $b(s_{min,max}) \mid min > 1 \; e \; max \geq min$. Uma relação "OU" é definida por $st_{min,max}(g_1, \dots, g_n) \mid 0 \leq min \leq max, e \; max \leq n$. Uma escolha alternativa é dada por $st_{1,1}(g_1, \dots, g_n)$.

5.3 Operadores de Mutação

Os seguintes operadores de mutação foram identificados e agrupados de acordo com as classes de defeitos previamente descritas, como apresentadas na Tabela 5.1.

Tabela 5.1 Relação de operadores de mutação com as classes de defeitos.

Classe de Defeito	**Operador**	**Descrição**
Atribuição incorreta da cardinalidade de uma característica solitária	**DFL**	Decrementa o valor mínimo da cardinalidade de uma característica solitária
	IFL	Incrementa o valor mínimo da cardinalidade de uma característica solitária
	DFU	Decrementa o valor máximo da cardinalidade de uma característica solitária
	IFU	Incrementa o valor máximo da cardinalidade de uma característica solitária
Atribuição incorreta de elementos de uma relação agrupada	**AFS**	Adiciona uma característica solitária a uma relação de agrupamento
	RFS	Remove uma característica de uma relação de agrupamento
Existência de uma relação de agrupamento	**RSR**	Remove uma relação de agrupamento

Tabela 5.2 (Continuação) Relação de operadores de mutação com as classes de defeitos.

Classe de Defeito	Operador	Descrição
Atribuição incorreta da cardinalidade de uma característica agrupada	**DRL**	Decrementa o valor mínimo da cardinalidade de uma característica agrupada
	IRL	Incrementa a valor mínimo da cardinalidade de uma característica agrupada
	DRU	Decrementa o valor mínimo da cardinalidade de uma característica agrupada
	IRU	Incrementa a valor máximo da cardinalidade de uma característica agrupada
Restrições incorretas	**FDC**	Inverte as características de uma relação de dependência
	RDC	Remove uma restrição de dependência
	REC	Remove uma restrição de exclusão
	CDC	Cria uma restrição de dependência
	CEC	Cria uma restrição de exclusão

5.3.1 DFL

O operador DFL (*decrease solitary feature lower bound*) decrementa o valor mínimo da cardinalidade de uma característica solitária. Para características mandatórias, com cardinalidade mínima igual a 1, o resultado da aplicação deste operador seria equivalente a conversão desta característica para opcional. Com base no diagrama de características de exemplo da Figura 2.3, a Figura 5.1 representa a aplicação do operador DFL à característica solitária Canal de Mensagem de Tráfego (cardinalidade [1..1]). Após a aplicação do operador a característica é alterada de mandatória para opcional (cardinalidade [0..1]).

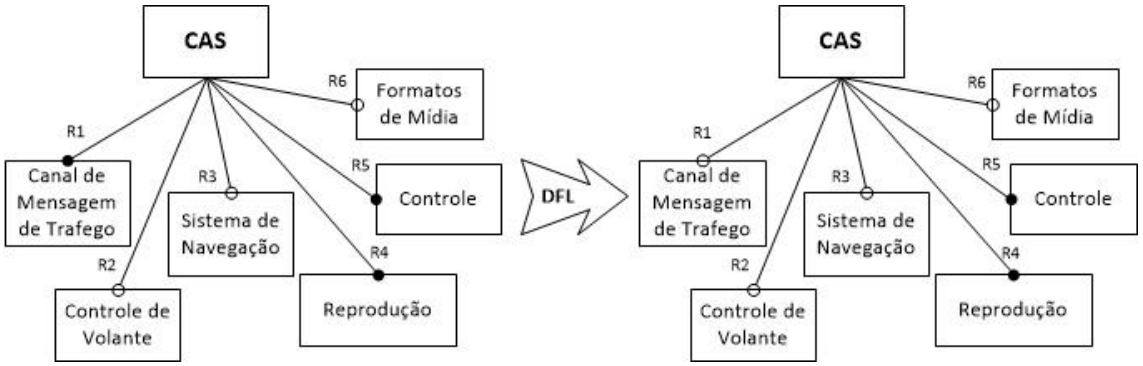

Figura 5.1 Exemplo de aplicação do operador DFL.

A função expressa pelo operador DFL pode ser descrita como:

$$DFL(s_{min,max}) = s_{min-1,max}\text{ , se } min > 0$$

5.3.2 IFL

O operador IFL (*increase solitary feature lower bound*) incrementa o valor mínimo da cardinalidade de uma característica solitária. Para características opcionais, com cardinalidade mínima igual a 0, o resultado da aplicação deste operador seria equivalente a conversão desta característica para mandatória. A Figura 5.2 apresenta um exemplo de aplicação do operador IFL à característica solitária Controle de Volante. Neste exemplo a característica Controle de Volante passou de opcional para mandatória.

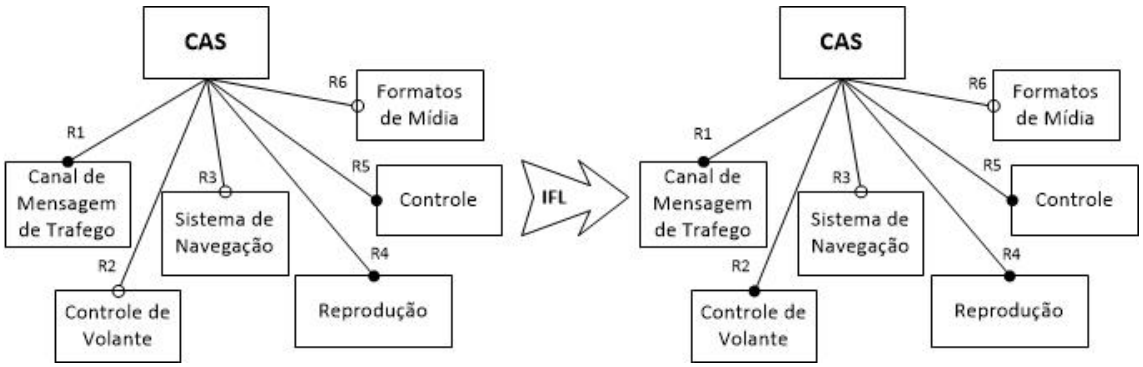

Figura 5.2 Exemplo de aplicação do operador IFL.

A função expressa pelo operador IFL pode ser descrita como:

$$IFL(s_{min,max}) = s_{min+1,max}\text{ , se } min < max$$

5.3.3 DFU

O operador DFU (*decrease solitary feature upper bound*) decrementa o valor máximo da cardinalidade de uma característica solitária. Por exemplo, ao aplicar o operador em uma característica solitária com cardinalidade [1..2], como resultado teríamos uma característica com cardinalidade [1..1]. A Figura 5.3 apresenta um exemplo de aplicação do operador DFU à característica clonada Canal de Mensagem de Tráfego de cardinalidade [1..2], originada de uma alteração do diagrama de características CAS [46] para ilustrar uma característica clonada. Neste exemplo, após a aplicação do operador, a característica Canal de Mensagem de Trafego, se torna

uma característica solitária de cardinalidade [1..1], ou seja, uma característica mandatória.

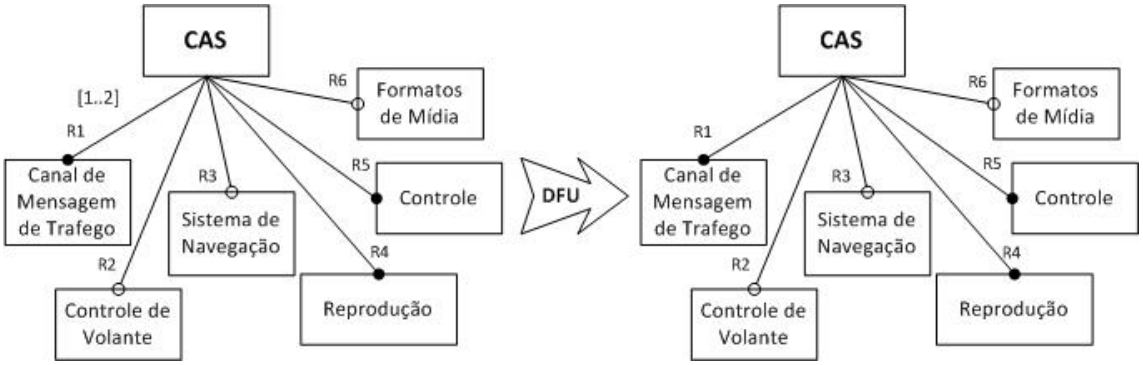

Figura 5.3 Exemplo de aplicação do operador DFU.

A função expressa pelo operador DFU pode ser descrita como:

$$DFU(s_{min,max}) = s_{min,max-1}\,, se\ min < max$$

5.3.4 IFU

O operador IFU (*increase solitary feature upper bound*) incrementa o valor máximo da cardinalidade de uma característica solitária. A Figura 5.4 apresenta um exemplo de aplicação do operador IFU à característica solitária Reprodução. Ao sofrer a mutação a cardinalidade da característica [1..1] é alterada para [1..2]. Desta forma o diagrama mutante possibilita a inserção de até duas características Reprodução dentro de um mesmo produto (clone).

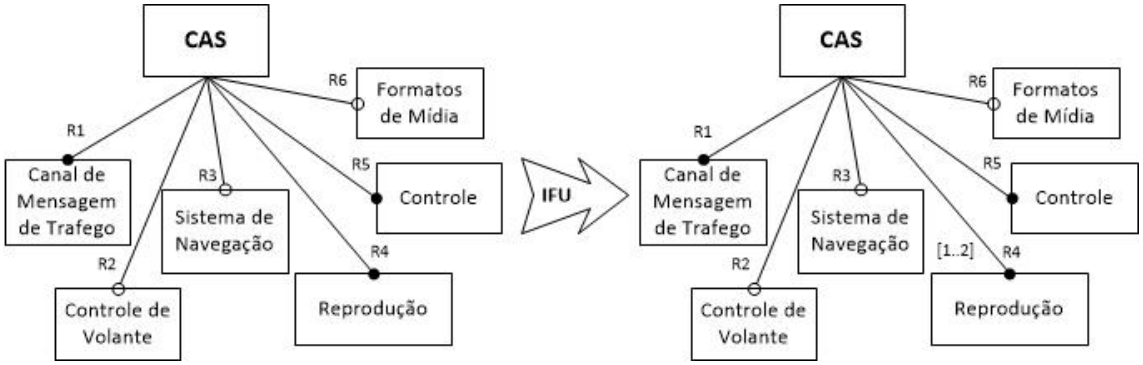

Figura 5.4 Exemplo de aplicação do operador IFU.

A função expressa pelo operador IFU pode ser descrita como:

$$IFU(s_{min,max}) = s_{min,max+1}$$

5.3.5 AFS

O operador AFS (*add feature to a set relation*) adiciona uma característica solitária a uma relação de agrupamento. A aplicação deste operador a uma característica resulta na conversão desta para uma característica agrupada. A Figura 5.5 apresenta um exemplo de aplicação do operador AFS. No exemplo a característica opcional MP3 foi adicionada à relação R11 entre a característica solitária Formatos de Mídia e as características agrupadas ACC, WMA e WAV.

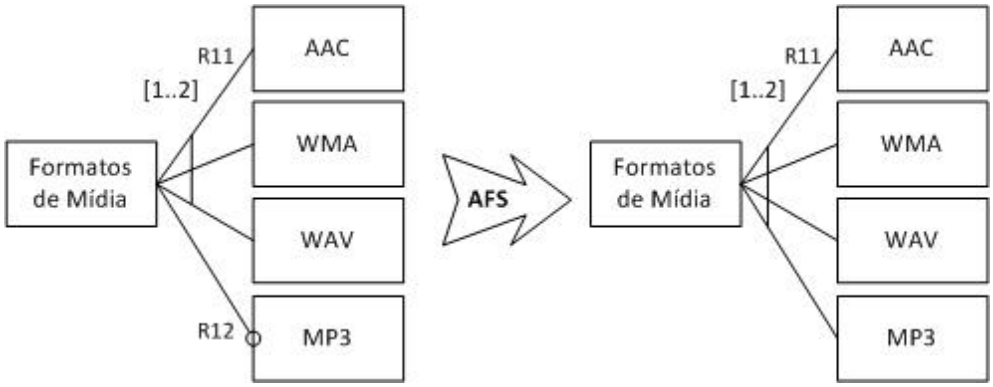

Figura 5.5 Exemplo de aplicação do operador AFS.

A função expressa pelo operador AFS pode ser descrita como:

$$AFS\left(st_{min,max}(f_1, \dots, f_{k-1}, f_{k+1}, f_n), b(f_{k_{min,max}})\right) = st_{min,max}(f_1, \dots, f_{k-1}, f_k, f_{k+1}, \dots, f_n)$$

5.3.6 RFS

O operador RFS (*remove feature from a set relation*) remove uma característica de uma relação de agrupamento. A aplicação deste operador a uma característica resulta na conversão desta para uma característica solitária. Como apresentado na Figura 5.6 o operador RFS removeu a característica agrupada WAV da relação R11, dando origem uma nova característica solitária WAV com cardinalidade [0..1] e a uma relação binária R17.

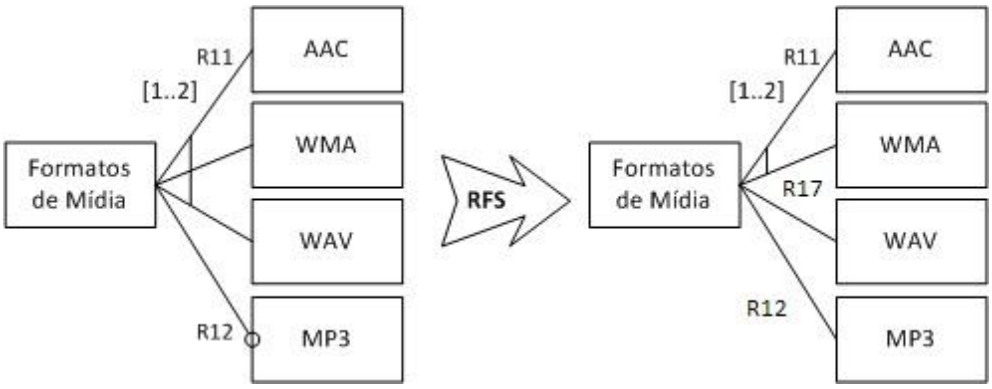

Figura 5.6 Exemplo de aplicação do operador RFS.

A função expressa pelo operador RFS pode ser descrita como:

$$RFS\left(st_{min,max}(f_1, \dots, f_{k-1}, f_k, f_{k+1} \dots, f_n)\right) = st_{min,max}(f_1, \dots, f_{k-1}, f_{k+1} \dots, f_n) \; e \; b(f_{k_{0,1}})$$

5.3.7 RSR

O operador RSR (*remove a set relation*) remove uma relação de agrupamento. A aplicação deste operador resulta na conversão de características agrupadas de uma relação de agrupamento em relações binárias e características solitárias opcionais, com cardinalidade mínima igual a 0. A Figura 5.7 apresenta um exemplo de aplicação do operador RSR. Neste exemplo, após a aplicação do operador a relação de agrupamento R11 foi removida, gerando três novas relações binárias R17, R18 e R19 relacionando as características solitárias AAC, WMA e WAV respectivamente, com cardinalidade [0..1], à característica solitária Formatos de Mídia.

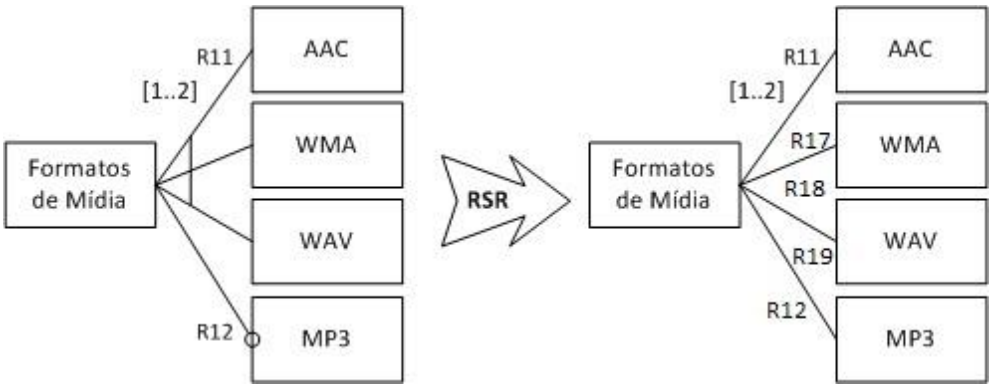

Figura 5.7 Exemplo de aplicação do operador RSR.

A função expressa pelo operador RSR pode ser descrita como:

$$RSR\left(st_{min,max}(f_1,\dots,f_n)\right) = b(f_{1_{0,1}}),\dots,b(f_{n_{0,1}})$$

5.3.8 DRL

O operador DRL (*decrease set relation lower bound*) decrementa o valor mínimo da cardinalidade de uma relação de agrupamento. Na Figura 5.8 é apresentado um exemplo de aplicação do operador DRL, na qual o valor mínimo da cardinalidade foi alterado, inicialmente uma relação [1..2] e após a aplicação [0..2], ou seja, essa relação de agrupamento agora permite produtos com nenhuma das características agrupadas.

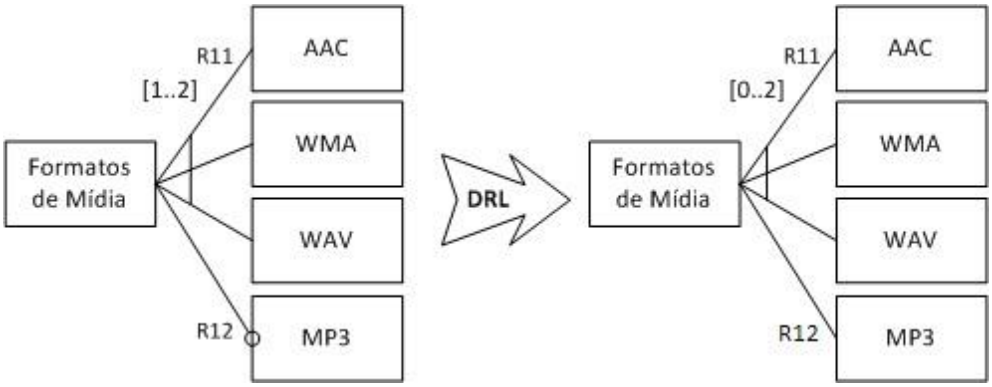

Figura 5.8 Exemplo de aplicação do operador DRL.

A função expressa pelo operador DRL pode ser descrita como:

$$DRL\left(st_{min,max}(f_1,\dots,f_n)\right) = st_{min-1,max}(f_1,\dots,f_n), se\ min > 0$$

5.3.9 IRL

O operador IRL (*increase set relation lower bound*) incrementa o valor mínimo da cardinalidade de uma relação de agrupamento. No exemplo da Figura 5.9, a aplicação do operador IRL aumentou o valor do limite mínimo da cardinalidade, de forma que após a sua aplicação, a relação agrupada R11 permite apenas duas características agrupadas em cada novo produto.

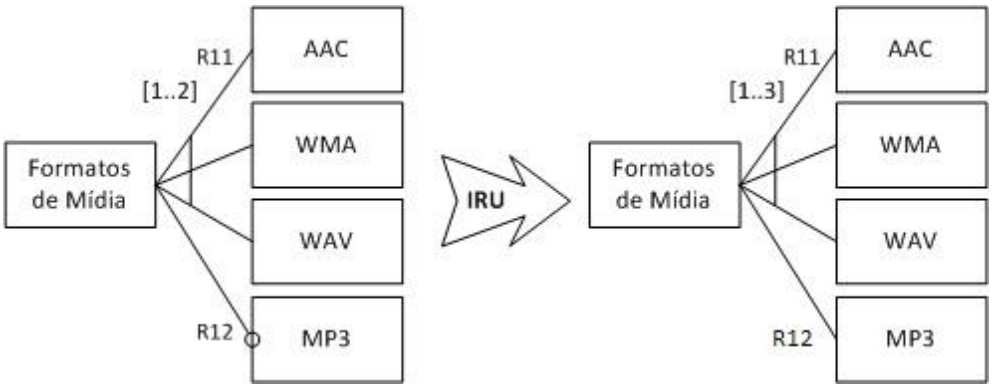

Figura 5.9 Exemplo de aplicação do operador IRL.

A função expressa pelo operador IRL pode ser descrita como:

$$IRL\left(st_{min,max}(f_1,\dots,f_n)\right) = st_{min+1,max}(f_1,\dots,f_n), se\ min < max$$

5.3.10 DRU

O operador DRU (*decrease set relation upper bound*) decrementa o valor máximo da cardinalidade de uma relação de agrupamento. O exemplo da Figura 5.10 ilustra a execução deste operador. Neste exemplo a execução deste operador sobre a relação R11, caracteriza em uma redução no valor máximo da cardinalidade da relação. Como resultado, R11 foi convertido de uma relação “OU” para uma escolha alternativa, na qual apenas uma das características pode estar presente em um produto.

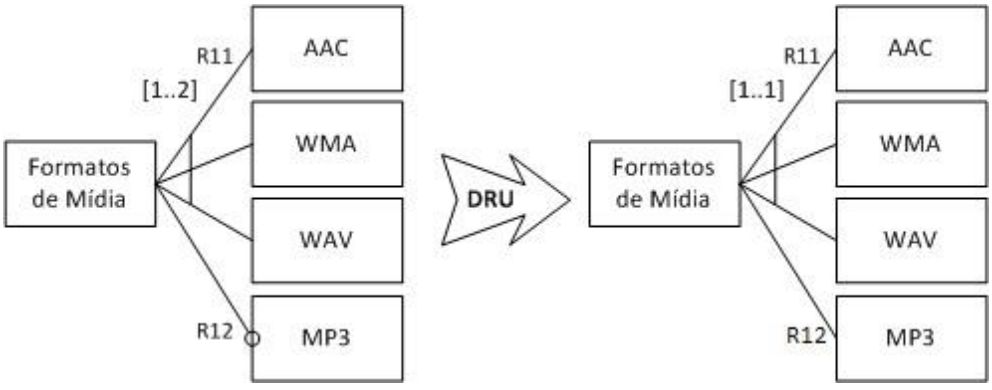

Figura 5.10 Exemplo de aplicação do operador DRU.

A função expressa pelo operador DRU pode ser descrita como:

$$DRU\left(st_{min,max}(f_1,\dots,f_n)\right) = st_{min,max-1}(f_1,\dots,f_n), se\ min < max$$

5.3.11 IRU

O operador IRU (*increase set relation upper bound*) incrementa o valor máximo da cardinalidade de uma relação de agrupamento. No exemplo da Figura 5.11 foi aplicado o operador IRU à relação R11, resultando na alteração do valor máximo da cardinalidade.

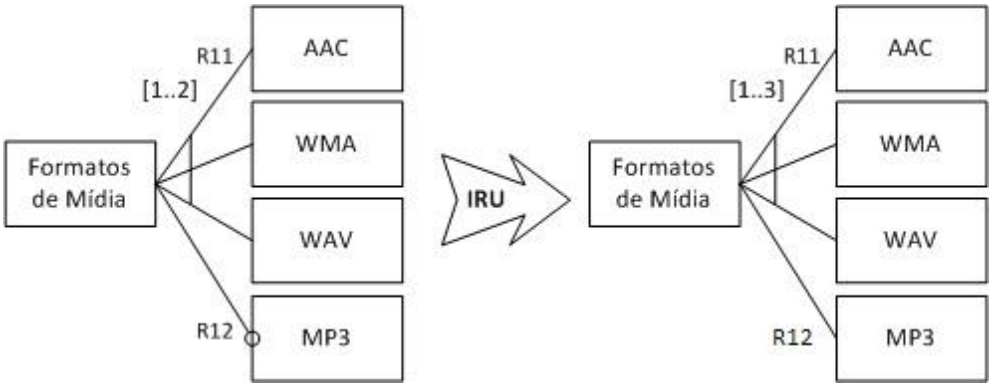

Figura 5.11 Exemplo de aplicação do operador IRU.

A função expressa pelo operador IRU pode ser descrita como:

$$DRU\left(st_{min,max}(f_1,\dots,f_n)\right) = st_{min,max+1}(f_1,\dots,f_n), se\ max < n$$

5.3.12 FDC

O operador FDC (*change feature of a depends constraint*) troca a posição das características de uma restrição de dependência, resultado em uma inversão na dependência entre as características envolvidas. No exemplo da Figura 5.12 a aplicação do operador FDC sobre a restrição RX resultou na inversão da restrição. Ou seja, após a execução, a característica Mapear Informação CD deverá estar contida em todos os produtos os quais contenham a característica CD.

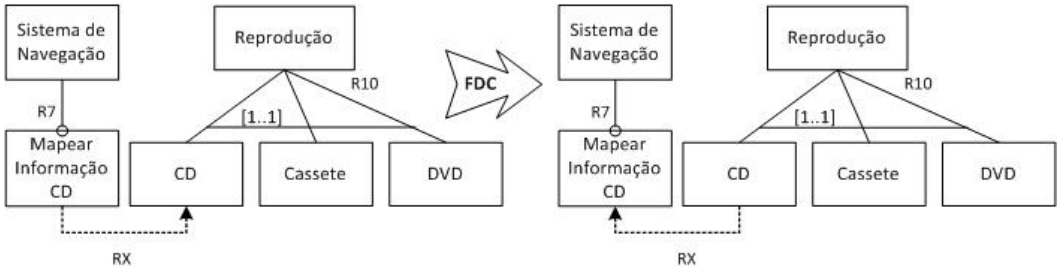

Figura 5.12 Exemplo de aplicação do operador FDC.

A função expressa pelo operador FDC pode ser descrita como:

$$FDC\big(d(f_a, f_b)\big) = d(f_b, f_a)$$

5.3.14 RDC

O operador RDC (*remove a depends constraint*) remove uma restrição de dependência. Como mostrado no exemplo da Figura 5.13, a aplicação do operador RDC sobre a restrição RX resultou na remoção desta restrição.

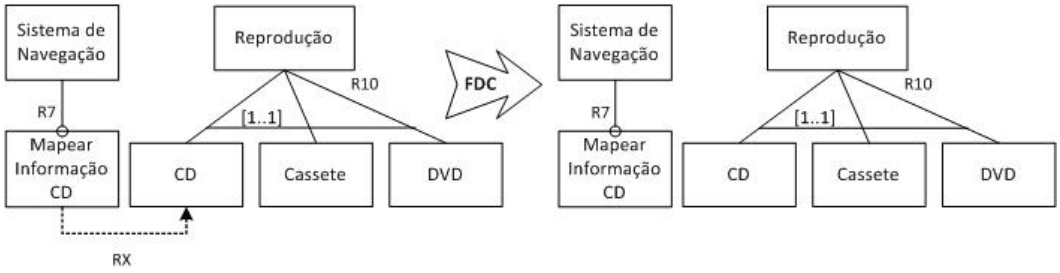

Figura 5.13 Exemplo de aplicação do operador RDC.

A função expressa pelo operador RDC pode ser descrita como:

$$RDC\big(d(f_a, f_b)\big) = D - \{d\}$$

5.3.15 REC

O operador REC (*remove a excludes constraint*) remove uma restrição de exclusão. Como mostra a Figura 5.14 a restrição de exclusão RY entre as características USB e Cassete foi removida após a execução do operador REC.

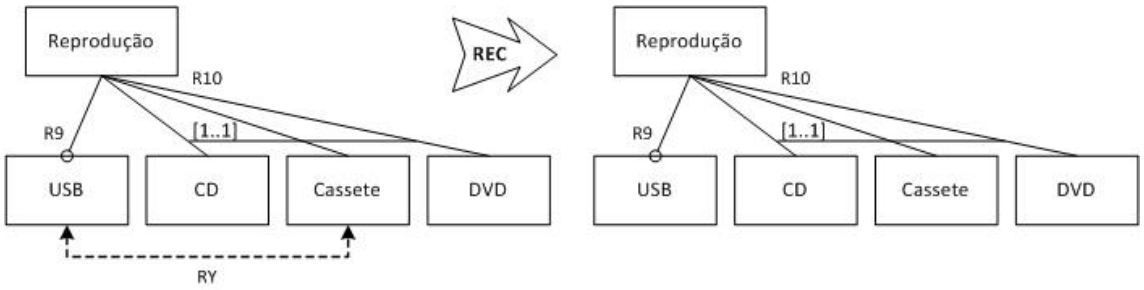

Figura 5.14 Exemplo de aplicação do operador REC.

A função expressa pelo operador REC pode ser descrita como:

$$REC\big(e(f_a, f_b)\big) = E - \{e\}$$

5.3.16 CDC

O operador CDC (*create a depends constraint*) cria uma nova restrição de dependência. Como mostra a Figura 5.15 a restrição RW foi criada entre as características Controle de Volante e MP3 após a execução do operador CDC.

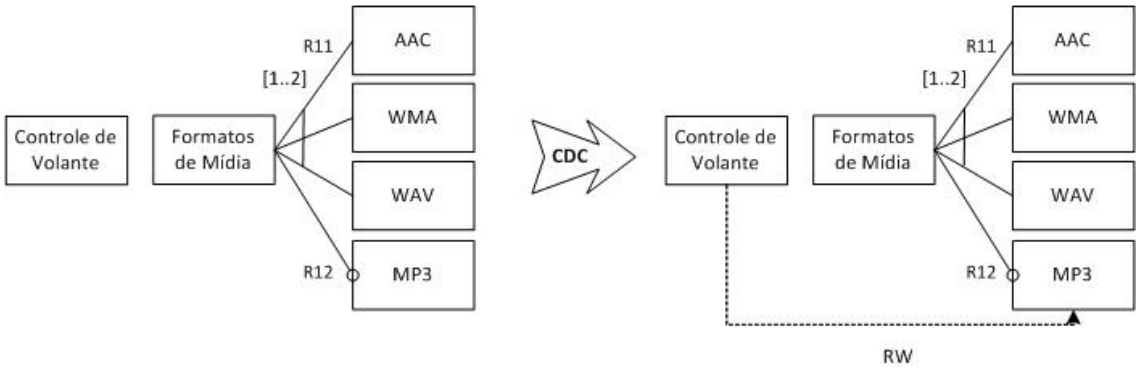

Figura 5.15 Exemplo de aplicação do operador CDC.

A função expressa pelo operador CDC pode ser descrita como:

$$CDC(f_1, \dots, f_n \in F)\big|f_k = s_{0,max} \in S \; or \; f_k = g \in G = d\big(f_i, f_j\big)\big|i, j \leq n,$$
$$i \neq j \; e \; c\big(f_i, f_j\big), c\big(f_i, f_j\big) \notin C$$

5.3.17 CEC

O operador CEC (*create a excludes constraint*) cria uma nova restrição de exclusão. Como mostra a Figura 5.16 a restrição de exclusão RW foi criada entre as características Controle de Volante e MP3 após a execução do operador CEC.

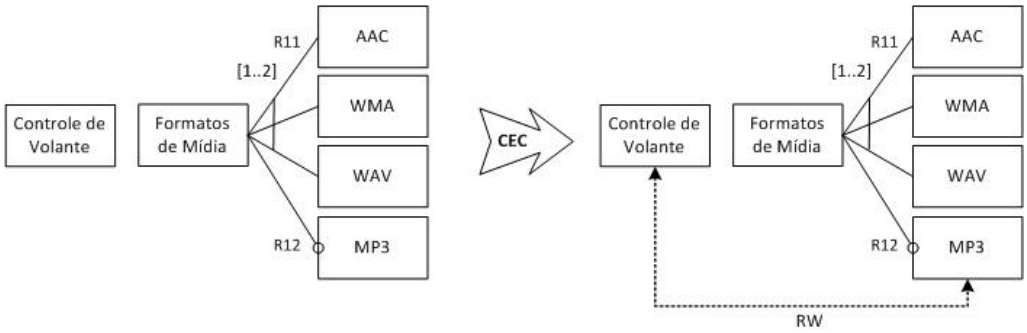

Figura 5.16 Exemplo de aplicação do operador CEC.

A função expressa pelo operador CEC pode ser descrita como:

$$CEC(f_1, \dots, f_n \in F)\big|f_k = s_{0,max} \in S \; or \; f_k = g \in G = e(f_i, f_j)\big|i, j \in [1, n],$$
$$i \neq j \; e \; c(f_i, f_j), c(f_i, f_j) \notin C$$

5.4 Processo de Teste

A Figura 5.17 representa o processo de análise de mutantes no diagrama de características. Os detalhes são discutidos a seguir.

O processo se inicia a partir de um diagrama de características válido fornecido. O processo é dividido em três etapas, geração de mutantes, a geração de casos de teste e a execução dos casos de teste. O processo de geração de mutantes aplica os operadores de mutação selecionados ao diagrama fornecido de acordo com uma porcentagem de aplicação definida pelo usuário tendo como saída um conjunto de diagramas mutantes.

A etapa de geração de casos de teste pode ser manual ou automática, no primeiro caso o usuário fornece os casos de teste que desejar com o intuito de matar os diagramas mutantes, e no segundo caso, a aplicação será responsável pela criação dos casos de teste a partir de características dos operadores e de um conjunto inicial de produtos. Esta etapa tem com saída um conjunto de casos de teste, ou seja, produtos válidos para o diagrama de características fornecido como entrada ou para um dos diagramas mutantes.

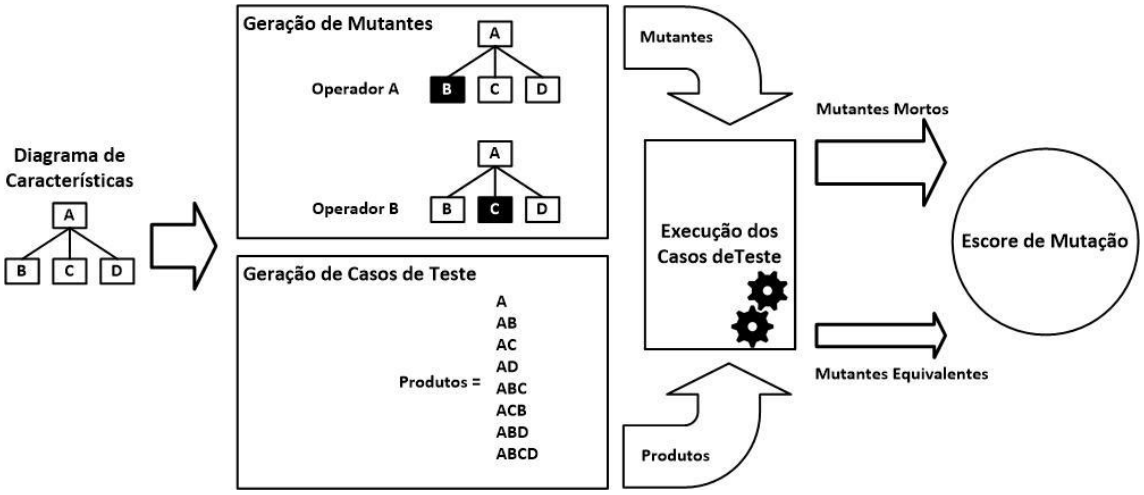

Figura 5.17 Aspectos de implementação da análise de mutantes no diagrama de características.

A partir dos diagramas mutantes e do conjunto de casos de teste a etapa de execução dos casos de teste é responsável por avaliar os produtos gerados utilizando os mutantes produzidos. Essa avaliação consiste em checar se os produtos criados são validos tanto para o diagrama de entrada quanto para o diagrama mutante. Caso o resultado de ambas as avaliações seja diferente, pode-se considerar que o caso de teste matou o diagrama mutante.

Cada um dos produtos do caso de teste é testado para todos os mutantes criados, visando a satisfazer o critério do teste de mutação com um conjunto reduzido de casos de teste. Caso o conjunto de casos de teste não satisfaça este critério a etapa de criação identifica os diagramas equivalentes e gera novos casos de teste visando maximizar o escore de mutação.

5.5 Exemplo de Aplicação

Considere o diagrama da Figura 5.18, resultante da aplicação do operador de mutação CEC no diagrama de características de CAS, na qual foi introduzida uma nova restrição de exclusão entre as características Controle de Volante e MP3.

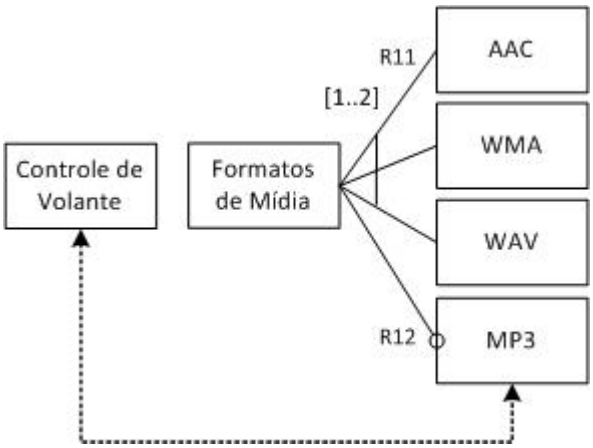

Figura 5.18 Exemplo de aplicação da proposta para o operador CEC.

Neste caso, para que um produto consiga matar o diagrama mutante é necessário que este obtenha valores do processo de validação diferenciados para o diagrama original e para o diagrama mutante. Tomando-se como exemplo o produto da Figura 5.19, que contém as características MP3 e Controle de Volante. Este produto é válido para o diagrama sendo testado, mas não é valido para o diagrama mutante, portanto é capaz de matar o mutante da Figura 5.18.

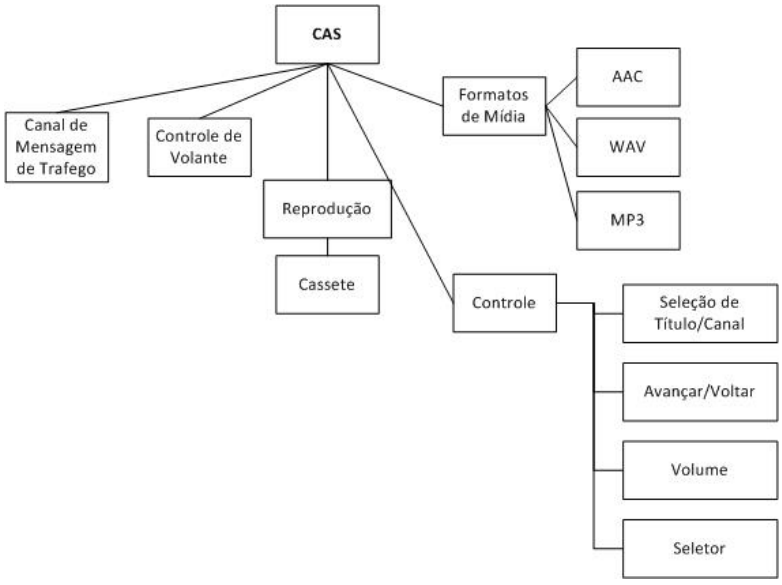

Figura 5.19 Exemplo de produto da LPS CAS.

5.6 Considerações Finais

A abordagem baseada em mutação proposta neste capítulo pode ser utilizada: (i) como um guia para a seleção de produtos para teste; e (ii) para avaliar a qualidade de um conjunto de teste. No primeiro caso, sem o auxílio de um conjunto inicial de

produtos, a tarefa consiste na geração e identificação de casos de teste que matem os mutantes até que um determinado escore de mutação seja atingido (idealmente $escore = 1$). No segundo, o escore de mutação é utilizado como uma forma de avaliação. Neste caso, o testador possui um conjunto de casos de teste T e precisa avaliar a qualidade deste conjunto. Da mesma forma, é possível utilizar a abordagem para comparar dois conjuntos de testes, baseados em seus escores de mutação.

Os usos apresentados acima não são exclusivos. Se um conjunto de teste T disponível não possui um escore adequado, é possível adequar tal conjunto pela adição de casos de teste. Esta utilização, com um conjunto inicial de casos de teste, possibilita uma redução no esforço para aplicação da abordagem de mutação. Desta forma, esta abordagem pode ser considerada complementar às abordagens existentes, como por exemplo, ao *pair-wise*. Ela melhora a eficácia em termos de defeitos revelados e oferece uma medida de cobertura para avaliar conjuntos de produtos.

Para implementar esta abordagem foi implementada uma ferramenta de suporte, descrita no próximo capítulo. Foi proposto um processo de identificação de diagramas equivalentes e através do uso de um algoritmo de otimização é possível uma redução do conjunto de casos de teste necessários para satisfazer o critério de teste de mutação. Este algoritmo e um exemplo de execução do processo são também apresentados no próximo capítulo.

CAPÍTULO 6 – ASPECTOS DE IMPLEMENTAÇÃO E VALIDAÇÃO

Para facilitar e automatizar a abordagem de teste apresentada no capítulo anterior, uma ferramenta de auxílio foi desenvolvida, *Feature Mutation Test Suite* (FMTS). A FMTS possibilita que, para um dado diagrama de características, seja possível realizar o processo de teste de mutação de características pela: (i) aplicação dos operadores de mutação e geração de diagramas mutantes; (ii) geração de conjuntos de produtos; (iii) avaliação do conjunto gerado. Além disso, a FMTS possibilita a avaliação de um conjunto de teste existente, a otimização deste conjunto e a identificação de diagramas equivalentes. Detalhes de implementação da FMTS são apresentados a seguir.

Após uma revisão criteriosa das diversas ferramentas disponíveis de auxílio no trabalho com diagramas de características, optou-se por utilizar o framework FAMA [4], uma vez que este, até a presente data de avaliação trata-se de uma ferramenta de código livre, desenvolvida em Java e que permite a extensão da sua biblioteca de funções para outras aplicações. Além disso, o FAMA apresenta o conjunto mais amplo de funções de avaliação de produtos e diagramas disponíveis. Desta forma, visando maior agilidade de implementação e facilidade na integração, optou-se pelo desenvolvimento da aplicação utilizando a linguagem Java.

A Figura 6.1 apresenta o diagrama de pacotes do FMTS. O pacote *FeatureModel* armazena as classes utilizadas para a modelagem do diagrama de características utilizado, e é apresentado em mais detalhes na Figura 6.2. O pacote *inOut*, é responsável pela leitura e escrita dos diagramas de características do disco em formato XML. O *testFramework*, é responsável pela geração de mutantes, geração de produtos e execução dos testes. Além disso, neste pacote são realizadas as validações de diagramas, geração de produtos iniciais, otimização do resultado, e o cálculo do escore de mutação. Por fim o pacote *userInterface*, implementa funções de interação entre a aplicação e o usuário.

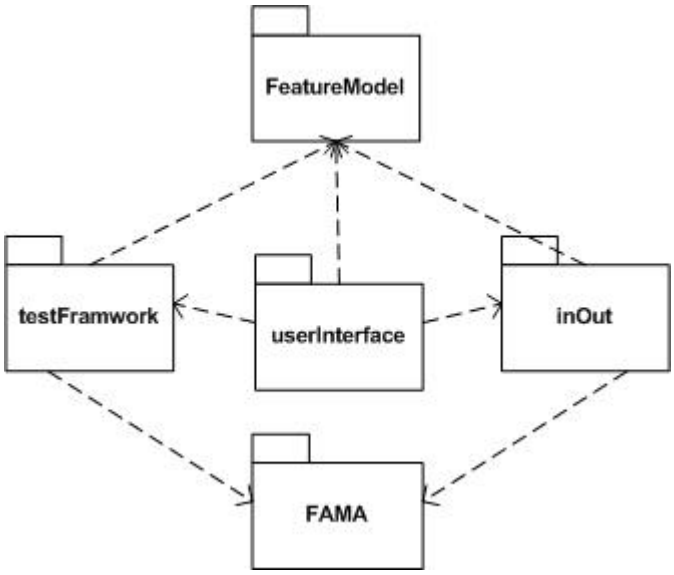

Figura 6.1 Diagrama de pacotes da ferramenta FMTS.

A interação com o FAMA acontece em dois momentos dentro do FMTS:

1. Dentro do pacote *inOut*, no qual ao realizar a leitura do diagrama de características, o mesmo é validado pelo FAMA, sendo a continuidade do processo de teste condicionado a validade do diagrama a ser testado;
2. Dentro do pacote *testFramework*, no qual o FAMA é utilizado para: validar os diagramas mutantes criados; gerar o conjunto inicial dos produtos válidos para o diagrama a ser testado, validar os casos de teste tanto para o diagrama a ser testado, quanto para o diagrama mutante.

Devido à simplicidade da aplicação a forma escolhida de interação do usuário com a aplicação é através de um console. Sendo que este, através de comandos pré-determinados possibilita ao usuário: realizar a leitura de um diagrama para a aplicação; gerar diagramas mutantes; gerar casos de teste; executar os casos de teste para os diagramas mutantes; calcular o escore de mutação; visualizar produtos e mutantes gerados; verificar diagramas equivalentes; e, refinar os resultados dos testes.

A Figura 6.2, apresenta o diagrama de classes associado ao pacote *FeatureModel*. A representação escolhida baseia-se no modelo de representação de diagramas de características descrito em [3]. Neste modelo, um diagrama de características é descrito pela composição de Restrições, Características e Cardinalidades, seguindo a formalização descrita no Capítulo 5.

A ferramenta desenvolvida é dividida em três módulos principais, o primeiro é composto de um gerador de diagramas mutantes, o segundo é responsável pela geração dos casos de teste, o terceiro é responsável pela execução dos casos de teste e pelo cálculo do escore de mutação. As funcionalidades e aspectos de implementação de cada módulo são descritos a seguir.

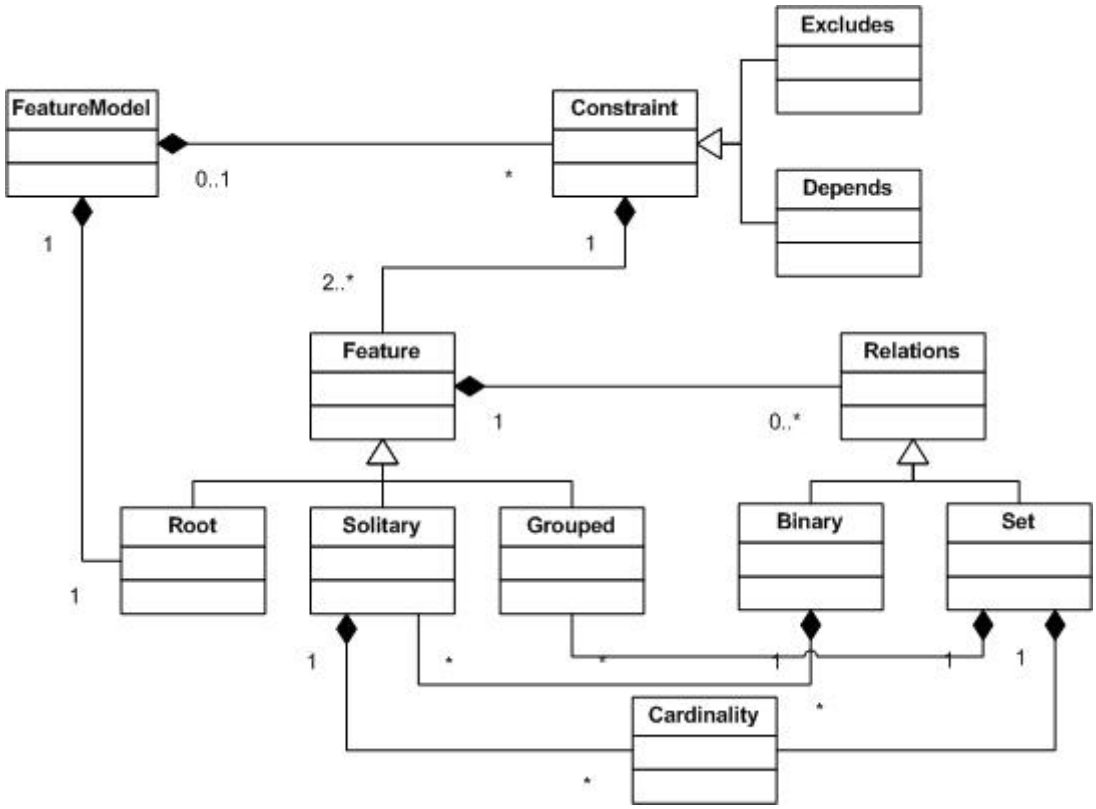

Figura 6.2 Diagrama de classes da ferramenta FMTS.

6.1 Geração de Mutantes

O módulo gerador de diagramas mutantes tem por objetivo, a partir da leitura de um diagrama de características de uma LPS fornecida, dos operadores de mutação que serão utilizados e da porcentagem de aplicação destes operadores nas relações entre as características do diagrama, a criação de novos diagramas mutantes. Os diagramas de entrada devem estar de acordo com o formato especificado pelo esquema XML fornecido junto ao FAMA.

A Figura 6.3 apresenta um arquivo XML para o diagrama de características da LPS de som automotivo mostrado ao longo dessa dissertação (Figura 2.3), definida

pelos elementos: (i) *feature*, para a característica raiz; (ii) *solitaryFeature*, para características solitárias; e (iii) *groupedFeature*, para características agrupadas. As relações são definidas por: (i) *binaryRelation*, para relações binárias; e, (ii) setRelation, para relações agrupadas. Cardinalidades são expressas por elementos *cardinality*, enquanto *excludes* e *requires* expressam restrições de exclusão e inclusão respectivamente.

De acordo com o modelo especificado, características e relações possuem o atributo nome (*name*). Cardinalidade possui valores máximos (*max*) e mínimos (*min*). Por fim, restrições devem apresentar os atributos, nome (*name*), característica (*feature*), e *excludes* ou *requires*, de acordo com a restrição, que irá definir qual característica deverá ser excluída ou incluída em um diagrama, respectivamente.

Para ilustrar a definição do arquivo XML, da Figura 6.3, têm-se:

- CAS é a característica raiz;
- O primeiro nível do diagrama é composto pelas características solitárias Canal de Tráfego de Mensagem, Controle de Volante, Sistema de Navegação, Reprodução, Controle e Formatos de Mídia;
- Canal de Mensagem de Tráfego e Reprodução são características solitárias obrigatórias;
- Controle de Volante e Formatos de Mídia são características opcionais;
- CD, Cassete e DVD são características agrupadas a característica Reprodução e com cardinalidade máxima e mínima igual a 1;
- RY é uma restrição de exclusão, na qual a característica USB exclui a característica Cassete;
- RX e RZ são restrições de inclusão, na quais, a característica Mapear Informação CD requer a característica CD e Mapear Informação USB requer a característica USB respectivamente.

Uma vez provido o diagrama de características no formato correto, e validado pela aplicação, o módulo de geração de mutantes fará a leitura deste arquivo e um novo objeto do tipo *FeatureModel*, contendo as informações relativas a LPS fornecida será criado.

```
<?xml version="1.0" encoding="UTF-8"?>
<feature-model>
  <feature name="CAS">
    <binaryRelation name="R1">
      <cardinality min="1" max="1" />
      <solitaryFeature name="Canal de Mensagem de Tráfego"></solitaryFeature>
    </binaryRelation>
    <binaryRelation name="R2">
      <cardinality min="0" max="1" />
      <solitaryFeature name="Controle de Volante"></solitaryFeature>
    </binaryRelation>
    <binaryRelation name="R3">
      <cardinality min="0" max="1" />
      <solitaryFeature name="Sistema de Navegação">
        <binaryRelation name="R7">
          <cardinality min="0" max="1" />
          <solitaryFeature name="Mapear Informação CD"></solitaryFeature>
        </binaryRelation>
        <binaryRelation name="R8">
          <cardinality min="0" max="1" />
          <solitaryFeature name="Mapear Informação USB"></solitaryFeature>
        </binaryRelation>
      </solitaryFeature>
    </binaryRelation>
    <binaryRelation name="R4">
      <cardinality min="1" max="1" />
      <solitaryFeature name="Reprodução">
        <binaryRelation name="R9">
          <cardinality min="0" max="1" />
          <solitaryFeature name="USB"></solitaryFeature>
        </binaryRelation>
        <setRelation name="R10">
          <cardinality min="1" max="1" />
          <groupedFeature name="CD"></groupedFeature>
          <groupedFeature name="Cassete"></groupedFeature>
          <groupedFeature name="DVD"></groupedFeature>
        </setRelation>
      </solitaryFeature>
    </binaryRelation>
    <binaryRelation name="R5">
      <cardinality min="1" max="1" />
      <solitaryFeature name="Controle">
        <binaryRelation name="R13">
          <cardinality min="1" max="1" />
          <solitaryFeature name="Seleção de Título/Canal"></solitaryFeature>
        </binaryRelation>
        <binaryRelation name="R14">
          <cardinality min="1" max="1" />
          <solitaryFeature name="Avançar/Voltar"></solitaryFeature>
        </binaryRelation>
        <binaryRelation name="R15">
          <cardinality min="1" max="1" />
          <solitaryFeature name="Volume"></solitaryFeature>
        </binaryRelation>
        <binaryRelation name="R16">
          <cardinality min="1" max="1" />
          <solitaryFeature name="Seletor"></solitaryFeature>
        </binaryRelation>
      </solitaryFeature>
    </binaryRelation>
    <binaryRelation name="R6">
      <cardinality min="0" max="1" />
      <solitaryFeature name="Formatos de Mídia">
        <binaryRelation name="R11">
          <cardinality min="0" max="1" />
          <solitaryFeature name="MP3"></solitaryFeature>
        </binaryRelation>
        <setRelation name="R12">
          <cardinality min="1" max="2" />
          <groupedFeature name="AAC"></groupedFeature>
          <groupedFeature name="WMA"></groupedFeature>
          <groupedFeature name="WAV"></groupedFeature>
        </setRelation>
      </solitaryFeature>
    </binaryRelation>
  </feature>
  <excludes name="RY" feature="USB" excludes="Cassete" />
  <requires name="RX" feature="Mapear Informação CD" requires="CD" />
  <requires name="RZ" feature="Mapear Informação USB" requires="USB" />
</feature-model>
```

Figura 6.3 Diagrama de características em formato XML para a LPS CAS.

Uma vez realizada a leitura e criação do objeto do diagrama de características, a próxima etapa consiste na identificação e localização das relações entre características, que juntamente com os operadores de mutação selecionados e da porcentagem de mutação fornecida serão utilizados para a criação dos diagramas mutantes. Esta identificação acontece durante uma segunda leitura sobre o diagrama criado. Durante a leitura, cada objeto identificado é armazenado para posterior uso. Entre estes objetos estão, características solitárias, características agrupadas, relações binárias, relações agrupadas e restrições.

Com os objetos armazenados, cada método para criação de mutantes para cada operador, fará uso do conjunto de objetos específicos, de forma que um novo diagrama clone ao diagrama de entrada será criado, a mutação será aplicada ao objeto armazenado, e posteriormente aplicada ao diagrama criado. Uma vez criado o diagrama mutante é avaliado, esta avaliação tem por objetivo evitar a criação de diagramas anômalos. Após a avaliação o diagrama mutante é salvo em um arquivo XML para posterior utilização. Estes passos se repetem para cada um dos pontos de mutação identificados para cada um dos operadores selecionados de acordo com a porcentagem estabelecida pelo usuário.

Para ilustrar a sequência de passos acima, considere, por exemplo, e execução do operador FDC para o diagrama da Figura 6.3. A partir da leitura inicial do diagrama, são identificadas todas as restrições de dependência de inclusão existentes: RX; e, RZ. Por definição, o operador FDC, apenas altera as características de uma restrição, trocando a característica (*feature*), pela característica requerida (*requires*). Em seguida, o método referente ao operador, realiza a troca destas características para cada uma das restrições identificadas, cria os diagramas clones, aplica a mutação gerada e salva os novos diagramas em formato XML. Desta forma, após a execução do operador FDC, têm-se dois novos diagramas mutantes: o primeiro, com mutação na restrição RX a qual a característica CD requer a característica Mapear Informação CD; e o segundo, no qual na restrição RZ a característica USB requer a característica Mapear Informação USB. A Figura 6.4 apresenta o pseudocódigo do método de geração de mutantes para FDC.

```
Nome: MutantGenerator.FDC
Entrada:
       fm: Diagrama de características a ser testado
       ListaDependencias: Lista de restrições de dependência
Início
       mutante m = fm
       Para cada dependência em ListaDepends de m
              Caracteristica a = dependencia.a
              Caracteristica b = dependencia.b
              dependencia.a = b
              dependencia.b = a
              AdicionaMutante(m)
                     fim se
              loop
       loop
Fim
```

Figura 6.4 Algoritmo para geração de mutantes segundo o operador FDC.

6.2 Geração de Produtos

O módulo de geração de produtos encarrega-se da geração de casos de teste. Este módulo recebe como entrada o diagrama de características a ser testado e o conjunto de operadores de mutação escolhidos, e tem como saída, um conjunto de produtos (não necessariamente válidos para o diagrama inicial) que será utilizado como casos de teste ao processo de teste de mutação, com a finalidade de matar o maior número possível de diagramas mutantes gerados.

A geração dos casos de forma semelhante à geração de mutantes se inicializa com leitura do diagrama de características a ser testado e a identificação das estruturas e relações dos existentes no mesmo. São estas estruturas: relações; características; e, restrições. Uma vez identificadas, o FAMA é então utilizado para a geração do conjunto de todos os produtos do diagrama. Estes produtos servirão de base para os produtos a ser gerados pelo módulo.

Com a lista de produtos disponível, o próximo passo é então a geração dos produtos propriamente dita. Esta é feita por módulos específicos para cada um dos operadores. Em cada um dos módulos uma das estruturas armazenadas é utilizada para a realização da modificação dos produtos de forma que estes atendam aos requisitos do operador. Para ilustrar a execução, cita-se o operador DFL (pseudocódigo na Figura 6.5), que decrementa o valor da cardinalidade mínima de uma característica solitária. As estruturas utilizadas pelo operador são as características binárias. Para a geração dos produtos é feita para cada uma dessas características binárias a verificação se esta característica está presente em um

produto válido listado pelo FAMA. Caso esteja presente, o método remove essa característica gerando um novo produto e adicionando à lista de casos de teste.

```
Nome: TestCaseGenerator.DFL
Entrada:
       fm: Diagrama de Características a ser testado
       ListaRelacoesBinarias: Lista de relações binárias
       ProdutosFaMa: Lista de produtos gerados pelo FAMA
Início
       Para cada característica em ListaRelacoesBinarias
              Para cada produto em ProdutosFaMa
                     Se produto contém característica então
                            novoProduto = produto
                            remove(característica, produto)
                            adicionaListadeProdutos(novoProduto)
                            continua próxima característica
                     Fim se
              Loop
       Loop
Fim
```

Figura 6.5 Algoritmo para geração de casos de teste para o operador DFL.

6.3 Teste

O terceiro módulo é dividido em duas partes, a primeira responsável por executar os dados de teste, ou seja, avaliar a validade dos produtos gerados pelo modulo gerador de produtos ou fornecidos pelo usuário para os diagramas, e a segunda é responsável pela verificação de diagramas equivalentes.

6.3.1 Execução dos dados de teste

Para o processo de teste é fornecido como entrada o diagrama da LPS a ser testado, os diagramas mutantes gerados e o conjunto de produtos criados. É produzida como saída uma lista de diagramas mutantes mortos e de quais casos de teste são capazes de matar cada um dos mutantes.

De maneira simplificada, a execução do dado de teste ocorre pela verificação de validade dos produtos gerados para cada um dos diagramas mutantes. Ou seja, é

carregado primeiramente o diagrama mutante, e um produto gerado. Então, através do FAMA, a validade do produto para o diagrama é obtida. Após é validado para o diagrama original o mesmo produto. Um diagrama mutante é considerado morto quando um produto valido para o diagrama original, não é valido para o mutante, ou quando o produto não é valido para o digrama original e é valido para o diagrama mutante. Esse processo se repete até que todos os produtos tenham sido testados para todos os diagramas mutantes.

6.3.1 Identificação de Diagramas Equivalentes

Ao se executar o processo de teste, verifica-se que alguns diagramas mutantes não são mortos. Isso pode ocorrer por dois fatores: (i) os casos de teste gerados não foram suficientes para matar todos os mutantes; ou, (ii) não há casos de testes para matar o diagrama mutante, neste caso ele é considerado um diagrama mutante.

Pelo processo de identificação de diagramas equivalentes é possível descobrir em qual dos casos o mutante se enquadra. Após a identificação de quais mutantes não foram mortos pelo conjunto de caso de teste, o processo inicia pela geração da lista completa, utilizando o FAMA, de todos os produtos pertencentes ao diagrama original e ao diagrama mutante. Após a obtenção desta lista, são validados estes produtos para o diagrama mutante e o diagrama da LPS a ser testada. Os critérios para descobrir se um mutante foi morto é o mesmo do utilizado no processo de teste.

Caso o mutante seja morto o produto que matou é adicionado à lista de produtos para que se possa em uma próxima etapa fazer uma otimização ou listagem dos resultados necessários para obter um score de mutação igual a 1. Entretanto, caso após a verificação de todos os produtos, o mutante não tenha sido morto, este é considerado como um diagrama equivalente, visto que nenhum valor das possíveis entradas para a aplicação conseguiu diferenciar o comportamento de ambos os diagramas. Apesar de ser um procedimento que pode ser extremamente custoso, devido ao aumento exponencial do número de produtos válidos para uma LPS com o aumento do número de características, é um processo que garante a correta identificação dos mutantes equivalentes de maneira automatizada.

6.5 Otimização do Resultado

O módulo de geração de produtos utiliza todos os produtos da LPS para matar os diagramas mutantes, mas sem aplicar qualquer otimização, por isto um número alto

de produtos é gerado. Para reduzir este conjunto foi implementado o algoritmo de otimização *Hill Climbing* (subida de encosta) [40]. Este algoritmo é apresentado na Figura 6.6. A ideia é reduzir o conjunto de produtos mantendo o escore de mutação.

O algoritmo de otimização de subida de encosta, pertencente à família das buscas locais, e consiste em um algoritmo iterativo que a partir de uma solução arbitrária para o problema, procura uma solução melhor pela execução incremental de mudanças em um único elemento da solução de cada vez. Para a otimização dos resultados encontrados pelo FMTS, utiliza-se como ponto de partida o conjunto de todos os casos de teste que matam algum diagrama mutante. O próximo passo, a partir da escolha arbitrária de um dos casos de teste da solução, é verificado se após a remoção deste, se ainda são mortos os mesmos mutantes que na solução inicial. Em caso positivo, esta nova solução é aceita e o algoritmo continua com a escolha arbitrária de novos casos de teste. Caso contrário, um novo caso de teste é selecionado, e a verificação é refeita. Estes passos se repetem até que o número de execuções do algoritmo, nas quais nenhuma solução melhor foi encontrada, seja igual ao número de produtos gerados.

```
Algoritmo Subida de Encosta
Início:
	noAtual = noInicial
	contador = 0
	enquanto (contador < tamanho(conjunto de casos de teste))
		V = vizinhos(noAtual)
		proxAval = -INF
		proxNo = NULL
		para cada x em V
			se (Aval(x) == Aval(noAtual))
				noAtual = x
			fim se
		loop
	loop
Fim
```

Figura 6.6 Algoritmo de Subida de Encosta (Hill-Climbing).

6.6 Opções e Comandos da Aplicação

A Tabela 6.1 apresenta as opções de comandos disponíveis para execução na ferramenta FMTS. No Apêndice A são apresentados os principais comandos associados ao FMTS, ilustrando resultados apresentados e telas da aplicação.

Tabela 6.1 Opções de comandos disponíveis na FMTS.

Comando	**Descrição**
help	Exibe as opções de comandos disponíveis na aplicação
load	Carrega um diagrama de características no formato XML para a aplicação
p-gen	Gera um conjunto casos de teste para o diagrama de características
p-add	Adiciona um novo produto para o conjunto de casos de teste
p-print	Exibe na tela todo o conjunto de casos de teste
p-initial	Exibe na tela o conjunto de todos os produtos válidos gerados pelo FAMA para o diagrama a ser testado
p-valid	Exibe na tela a lista dos produtos do conjunto de casos de teste e o status, informando se é válido ou não para a linha de produto a ser testada.
m-gen	Gera os diagramas mutantes para o diagrama a ser testado
m-print	Exibe na tela o diagrama de características mutante fornecido
m-dead	Exibe na tela a lista dos mutantes e seus status
test	Executa para o conjunto de mutantes o conjunto de casos de teste disponíveis
fm-print	Exibe na tela o diagrama de características a ser testado
eq	Realiza a verificação dos diagramas equivalentes
score	Exibe o resultado do cálculo do escore de mutação com base nos resultados obtidos pelo comando teste
save-score	Gera uma planilha listando o status do resultado da verificação de todos os diagramas mutantes pelos casos de teste gerados
exit	Encerra e execução da aplicação

6.2 Considerações Finais

O framework FMTS descrito neste capítulo é de fundamental importância para tornar possível a aplicação do teste de variabilidade em LPS baseado em mutação do diagrama de características. Além disso, esta ferramenta permite a realização de experimentos de aplicação da proposta. Estes experimentos estão descritos no próximo capítulo.

CAPÍTULO 7 – EXPERIMENTO

O experimento conduzido tem como objetivo a avaliação da abordagem de teste baseado em defeitos no contexto de linha de produto de software. Além disso, pretende-se com o experimento avaliar os operadores de mutação inicialmente propostos. A avaliação foi feita pela análise dos operadores considerando aspectos do diagrama de características e fatores, tais como, custo, em termos de número de mutantes gerados. Na avaliação, o conjunto de produtos gerados pela abordagem é comparado com o conjunto de produtos gerados utilizando o teste *pair-wise* implementado pelo algoritmo AETG.

7.1 Linhas de Produto de Software

Para a execução do experimento foram escolhidas as seguintes linhas de produto de software:

- **CAS** [46]**:** Linha de produto de software para sistemas de som automotivo, descrita no Capítulo 2.
- **JAMES** [3]: É uma linha de produto de software para sistemas web colaborativo.
- **Weather Station** [5]: Descreve uma linha de produto de software para sistemas meteorológicos.
- **E-Shop** [42]: Linha de produto de software simplificada para E-commerce SPL e que ilustra a maneira como características são especificadas e utilizadas na construção de sistemas de compra online.

Os diagramas de características utilizados estão disponíveis no Apêndice B. Algumas características dos diagramas destas LPS são apresentadas na Tabela 7.1. Observa-se que: (i) a LPS E-Shop é a que possuí um maior número de produtos; (ii) todas as LPS possuem restrições; e, (iii) quanto maior o número de características agrupadas maior o número de produtos.

Tabela 7.1 Características dos diagramas analisados.

	CAS	JAMES	Weather Station	E-Shop
Características Mandatórias	7	4	4	4
Características Opcionais	5	1	1	4
Características Agrupadas	8	8	15	13
Relações Binárias	12	5	5	8
Relações Agrupadas	2	3	6	6
Restrições de Dependência	1	1	0	1
Restrições de Exclusão	1	1	0	1
Número de Produtos	449	67	503	1151

Pode se observado na Figura 7.1 que a maior influência no número de produtos deve-se ao número características agrupadas e opcionais que um diagrama de características contém. Levando-se em conta o número de características opcionais e mandatórias temos: (i) para a LPS JAMES, com 9 características, um total de 67 produtos; (ii) para a CAS, com 13 características, tem-se um total de 499 produtos; (iii) para a Weather Station, com 16, tem-se 503 produtos; e, (iv) para a E-Shop, com 17 tem-se um total de 1151 produtos. Esta tendência se explica pelo número de possibilidades de combinação destas características.

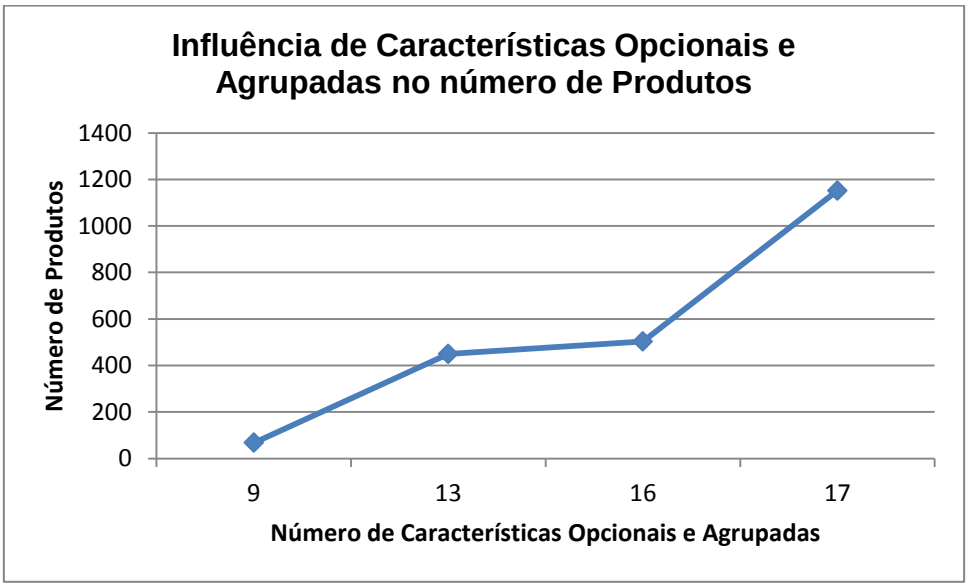

Figura 7.1 Influência de características opcionais e agrupadas no número de produtos.

De forma inversa às características opcionais e agrupadas, as restrições de uma LPS tendem a diminuir o número de produtos, uma vez que essas impossibilitam

a validação de produtos dependendo da combinação das características. Por fim, características mandatórias não alteram o número de produtos, uma vez que estas estarão contidas em todos os produtos gerados.

7.2 Metodologia Adotada

O experimento foi realizado pela execução ordenada dos seguintes passos:

1. Inicialmente, utilizando a FMTS foi gerado, para todos os operadores de mutação, o conjunto inicial de mutantes. Entretanto, por limitação da ferramenta FAMA, em não conseguir identificar e validar diagramas contendo características clonadas, os operadores DRU e IRU não foram utilizados nos experimentos. Os mutantes gerados foram salvos em disco no formato XML para posterior utilização. Após a geração de cada um dos diagramas mutantes foi realizada a sua validação, e apenas os mutantes válidos consistentes foram considerados;
2. O próximo passo foi a geração do conjunto de casos de teste para cada um dos operadores de mutação.
3. Com o conjunto de mutantes e o conjunto de casos de teste foi realizada a validação dos diagramas mutantes e original para cada um dos produtos gerados, com o intuito de matar os diagramas mutantes;
4. Os resultados obtidos no Passo 3 foram armazenados e serviram de base para a identificação dos diagramas equivalentes. Essa identificação foi realizada pela validação de todos os produtos do diagrama original e do diagrama mutante, em caso de divergência neste resultado o mutante foi considerado morto e o produto que causa esta divergência foi adicionado ao conjunto de casos de teste. Caso contrário, o diagrama foi considerado equivalente;
5. Com o novo conjunto, de todos os produtos que matam os diagramas não equivalentes, os diagramas mutantes foram novamente testados. O intuito desta etapa é avaliar os novos produtos gerados na etapa de identificação de diagramas equivalentes para os demais produtos;
6. Após a validação do conjunto, executou-se o algoritmo de otimização *hill climbing* para redução do conjunto de produtos. O resultado desta etapa são os produtos a serem considerados para posteriores etapas de teste;

7. Para a avaliação da metodologia proposta, o conjunto de mutantes foi executado com o conjunto de produtos gerados utilizando o algoritmo AETG e foi calculado o seu escore de mutação;
8. Foram comparados os resultados encontrados do teste dos produtos gerados pelo FMTS com o resultado utilizando produtos gerados pelo AETG.

Os resultados obtidos são avaliados nas próximas subseções com objetivo de analisar a aplicabilidade da abordagem baseada em mutação e implementada pela ferramenta FMTS, e de comparar os seus resultados com o teste *pair-wise* utilizando o algoritmo AETG.

7.3 Análise de Aplicabilidade

A Tabela 7.2 apresenta o número de diagramas mutantes gerados, o número de mutantes equivalentes identificados pela ferramenta, o número de produtos necessários (gerados automaticamente como conjunto inicial de produtos a serem testados somado com o número de produtos gerados durante a fase de identificação dos diagramas equivalentes), o número de mutantes equivalentes, e o número de produtos utilizados para matar todos os mutantes de cada operador após a otimização utilizando o algoritmo *hill climbing*.

Pela Tabela 7.2 pode-se observar que foram gerados 268 mutantes para a LPS CAS, 129 para a JAMES, 387 para a Weather Station e 453 para a E-Shop. Tendo, novamente como maiores responsáveis pelo número maior de mutantes, as características opcionais, sejam elas em relações agrupadas ou em relações binárias, ou seja, o número de mutantes gerados é proporcional ao número de produtos da LPS.

Como observado na Tabela 7.3 os operadores CDC e CEC, que criam restrições de dependência e de exclusão, geram a maior quantidade de mutantes, seguidos pelos operadores RFS, DFL, IFL e RSR que alteram a cardinalidade de características (IFL e DFL) solitárias ou alteram propriedades de relações agrupadas (RFS, RSR). Por outro lado, os operadores REC e RDC, pertencentes à classe de defeitos relacionados à cardinalidade, assim como o AFS, relacionado a adição de características à relações agrupadas, geraram poucos mutantes

Essa diferença expressiva no número de mutantes se explica pela característica dos operadores, os primeiros estão relacionados a restrições ausentes no diagrama de características. Desta forma, visando a identificar a restrição não

existente, são gerados mutantes pela adição de combinação de características opcionais duas a duas. Enquanto que os últimos, com menor número de mutantes, lida com restrições presentes no diagrama, que em geral, não existem em um número expressivo. Operadores que alteram cardinalidade de relações ou características devem em geral ser gerados em um número similar ao número total dessas estruturas presentes no diagrama.

Tabela 7.2 Resultados da execução do processo de teste pela FMTS para os diagramas de característsticas análisados.

	CAS				**JAMES**				**Weather Station**				**E-Shop**			
Operador	Número de Mutantes Gerados	Número de Mutantes Equivalentes	Número de Produtos Gerados	Número de Produtos Otimizado	Número de Mutantes Gerados	Número de Mutantes Equivalentes	Número de Produtos Gerados	Número de Produtos Otimizado	Número de Mutantes Gerados	Número de Mutantes Equivalentes	Número de Produtos Gerados	Número de Produtos Otimizado	Número de Mutantes Gerados	Número de Mutantes Equivalentes	Número de Produtos Gerados	Número de Produtos Otimizado
DFL	10	0	12	10	4	0	4	4	1	0	6	1	6	0	12	6
IFL	12	0	10	8	1	0	1	1	1	0	1	1	5	0	10	5
AFS	2	0	12	1	0	0	6	0	0	0	11	0	0	0	6	0
RFS	6	0	8	4	8	0	8	6	15	0	15	7	13	0	11	7
RSR	2	0	3	2	3	0	3	3	6	0	6	6	6	0	6	6
DRL	2	0	2	1	3	0	3	3	1	0	1	1	6	0	6	6
IRL	1	0	1	1	2	0	2	2	1	0	1	1	5	0	5	5
DRU	1	0	1	1	2	0	2	2	1	0	1	1	5	0	5	5
IRU	2	0	2	1	1	0	1	1	1	0	1	1	2	0	2	1
FDC	2	0	2	1	1	0	1	1	0	0	0	0	1	0	2	1
RDC	2	0	2	2	1	0	1	1	0	0	0	0	1	0	1	1
REC	1	0	1	1	1	0	1	1	0	0	0	0	1	0	1	1
CDC	150	2	152	26	68	2	67	6	240	4	44	22	268	2	270	32
CEC	75	0	77	1	34	2	61	6	120	13	69	37	134	2	145	28
Total	268	2	285	27	129	4	161	15	387	17	156	44	453	4	482	52

Para cada uma das linhas de produto de software foi gerado automaticamente uma quantidade diferente de produtos iniciais a serem validados, sendo 268 para a CAS, 161 para a JAMES, 156 para a Weather Station e 482 para a E-Shop. A quantidade de produtos é grande visto que o objetivo desta etapa não é encontrar a menor quantidade de produtos que matem todos os mutantes, mas sim encontrar produtos cuja validação resulte em um escore igual a 100%. A otimização destes resultados é realizada em uma etapa posterior.

Tabela 7.3 Resumo da avaliação dos conjuntos de produtos gerados pelo FMTS

Operador	Total de Mutantes gerados	Média Aritmética de Mutantes Gerados	Total de Mutantes Equivalentes	Média Aritmética de Mutantes Equivalentes	Total de Produtos Gerados	Média de Produtos Gerados	Total de Produtos Otimizados	Média de Produtos Otimizados	Redução do Número de Produtos
DFL	21	5,25	0	0	34	8,5	21	5,25	0,38
IFL	19	4,75	0	0	22	5,5	15	3,75	0,32
AFS	2	0,5	0	0	35	8,75	1	0,25	0,97
RFS	42	10,5	0	0	42	10,5	24	6	0,43
RSR	17	4,25	0	0	18	4,5	17	4,25	0,06
DRL	12	3	0	0	12	3	11	2,75	0,08
IRL	9	2,25	0	0	9	2,25	9	2,25	0,00
DRU	9	2,25	0	0	9	2,25	9	2,25	0,00
IRU	6	1,5	0	0	6	1,5	4	1	0,33
FDC	4	1	0	0	5	1,25	3	0,75	0,40
RDC	4	1	0	0	4	1	4	1	0,00
REC	3	0,75	0	0	3	0,75	3	0,75	0,00
CDC	726	181,5	10	2,5	533	133,3	86	21,5	0,84
CEC	363	90,75	17	4,25	352	88	72	18	0,80
Total	1237	309,3	27	6,75	1084	271	138	34,5	0,87

CDC e CEC são os únicos operadores responsáveis pela geração de diagramas equivalentes. Para a maioria dos operadores existe uma relação entre a quantidade de mutantes gerados e a quantidade de produtos necessários. Mas isto não acontece com os operadores CDC e CEC que são responsáveis pela geração da

maior quantidade de mutantes e também de produtos. Os operadores AFS, FDC, RDC e REC por sua vez geraram a menor quantidade de mutantes.

Após a otimização, utilizando um algoritmo de busca local, dos resultados tem-se uma redução significativa no número de produtos necessários (87% de redução no número de produtos). Observa-se que os operadores que geraram mais mutantes são os que mais se beneficiaram com a otimização, sendo estes AFS (97% de redução), CDC (86 produtos e 84% de redução) e CEC (72 produtos e 80% de redução). Por outro lado IRL, DRU, RDC, REC mantiveram-se com o mesmo número de produtos. Estes resultados podem ser utilizados para avaliar os algoritmos de geração de produtos para os operadores, de forma que, um maior número de redução significa menor eficiência do algoritmo e maior trabalho necessário para avaliação.

7.4 Comparação com *Pair-Wise*

Como mencionado anteriormente foi calculado o escore de mutação para os conjuntos de produtos gerados pelo algoritmo AETG (*pair-wise*). Os resultados são apresentados nas Tabela 7.4 e 7.5. O número de produtos gerados pelo AETG para cada um dos diagramas testados é: (i) CAS, 12 produtos; (ii) JAMES, 9 produtos; (iii) Weather Station, 11 produtos; (iv) E-Shop, 11 produtos.

Tabela 7.4 Resumo dos resultados da execução do processo de teste utilizando o AETG como algoritmo para geração de produtos.

LPS	Mutantes	Equivalentes	Mortos	Escore de Mutação
CAS	268	2	217	0,819
JAMES	129	4	86	0,688
Weather Station	387	17	195	0.520
E-Shop	453	4	86	0,189

Para a linha de produto CAS tem-se um escore de mutação de 0,819, o que significa que 18% dos mutantes não foram mortos, para a JAMES com um escore de 0,688, resulta em 31% de mutantes não mortos, para a Weather Station, com escore 0,52 tem-se um total de 49% de mutantes não mortos, e por fim, para a E-Shop, 0,189 de escore de mutação e um total de 81% de mutantes não mortos. Os escores de mutação apresentados indicam que quanto maior o número de produtos uma LPS contém, menor é escore de mutação do conjunto.

Tabela 7.5 Resultados detalhados da execução do processo de teste utilizando o AETG como algoritmo para geração dos produtos.

	CAS			JAMES			Weather Station			E-Shop		
Operador	Mutantes Gerados	Mutantes Mortos AETGSet	Produtos Requridos AETGSet	Mutantes Gerados	Mutantes Mortos AETGSet	Produtos Requridos AETGSet	Mutantes Gerados	Mutantes Mortos AETGSet	Produtos Requridos AETGSet	Mutantes Gerados	Mutantes Mortos AETGSet	Produtos Requridos AETGSet
DFL	10	0	0	4	0	0	1	0	0	6	0	0
IFL	12	4	3	1	1	1	1	0	0	5	1	1
AFS	2	1	1	0	0	0	0	0	0	0	0	0
RFS	6	5	4	8	4	4	15	10	4	13	2	2
RSR	2	0	0	3	0	0	6	0	0	6	1	1
DRL	2	0	0	3	0	0	1	0	0	6	0	0
IRL	1	0	0	2	0	0	1	0	0	5	0	0
DRU	1	0	0	2	0	0	1	0	0	5	0	0
IRU	2	0	0	1	0	0	1	0	0	2	0	0
FDC	2	2	2	1	1	1	0	0	0	1	1	1
RDC	2	2	2	1	1	1	0	0	0	1	1	1
REC	1	1	1	1	1	1	0	0	0	1	0	0
CDC	150	127	7	68	57	6	240	142	4	268	66	1
CEC	75	75	1	34	21	6	120	43	4	134	14	1
Total	268	217	11	129	86	9	387	195	11	453	86	8

Como apresentado na Tabela 7.5 e 7.6, em particular, o conjunto gerado pelo AETG apresenta um resultado ruim para os operadores DFL, DRL, IRL, DRU, IRU não matando nenhum dos diagramas mutantes e para RSR e AFS com apenas 1 diagrama morto. Os produtos gerados pelo AETG mostram-se mais eficazes para matar produtos gerados pelos operadores CDC, CEC e RFS, operadores que demandam um maior número de produtos para matar os mutantes e que estão associados as restrições do diagrama. Em geral o AETG demonstra uma especial dificuldade de identificar defeitos relacionados à cardinalidade das características, demonstrado pelo

seu baixo desempenho para estes operadores. O conjunto específico dos *pair-wises* para o diagramas de características da LPS CAS está disponível no Apêndice C.

Tabela 7.6 Resumo da avaliação do conjunto gerado pelo AETG.

Operador	Soma dos Mutantes Gerados	Soma dos Mutantes Mortos pelo AETG	Soma dos Produtos Requeridos pelo AETG
DFL	21	0	0
IFL	19	6	5
AFS	2	1	1
RFS	42	21	14
RSR	17	1	1
DRL	12	0	0
IRL	9	0	0
DRU	9	0	0
IRU	6	0	0
FDC	4	4	4
RDC	4	4	4
REC	3	2	2
CDC	726	392	18
CEC	363	153	12
Total	1237	584	39

Devido à natureza do operador CEC, o qual gera diagramas mutantes pela adição de restrições de exclusão geradas pela combinação de características não obrigatórias e pelo REC que gera diagramas mutantes pela remoção de restrições de exclusão, tem-se que para um conjunto de produtos conseguir matar todos os mutantes gerados por estes operadores este deve conter todo o conjunto de *pair-wises* para a linha de produto de software em teste.

Observa-se que, exceto para a LPS CAS, todos os produtos dos conjuntos de produtos gerados pelo AETG foram necessários para matar os mutantes gerados. Isto

pode ser explicado pelos tipos de defeitos descritos pelo teste pair-wise e operador CEC.

Para matar os diagramas gerados pelos operadores CEC um conjunto de produtos precisa conter necessariamente as combinações das duas características que foram adicionadas à nova restrição de exclusão. Este fato garante que todos os pares de características opcionais (*pair-wise*) estejam contidos nos produtos necessários para matar os mutantes gerados por este operador. Desta maneira, pode-se considerar que um conjunto de produtos que possua um escore de mutação igual a 100% vai ao que tudo indica necessariamente conter todos os conjuntos *pair-wise* de características de uma LPS. Mas isto deverá ser avaliado futuramente em novos experimentos.

7.4 Considerações Finais

Pela observação dos resultados obtidos pode-se observar que os operadores CDC e CEC precisam da maior quantidade de produtos, estes operadores foram os únicos que geraram diagramas de características equivalentes. Além disso, é possível observar também que o algoritmo de otimização *Hill Climbing* conseguiu reduzir o conjunto de produtos necessários para atingir um escore de 100% em 87%.

Pela realização do experimento com o conjunto dos produtos gerados pelo AETG tem-se que os resultados dos escores de mutação aparentemente decaem conforme o número de produtos de uma LPS aumenta. Outro ponto importante deve-se ao fato de ao que tudo indica um conjunto de produtos com escore de mutação igual a 100% contém o conjunto *pair-wise* de características.

CAPÍTULO 8 - CONCLUSÃO

Este trabalho apresenta uma abordagem baseada em defeitos para o teste de diagramas de características de linhas de produto de software. A ideia é utilizar o teste de mutação como critério de seleção de produtos para teste, considerando defeitos comuns que podem estar presentes nestes diagramas. Estes defeitos por sua vez, são categorizados e descritos como operadores de mutação.

Para a aplicação dos operadores, foi proposto um processo semelhante ao teste de mutação de programas. Neste processo um mutante é considerado morto quando o resultado da validação de um produto para um diagrama mutante é diferente do resultado da validação para o diagrama a ser testado. Ao final, é obtido o escore de mutação que pode ser utilizado como guia para a geração de produtos, ou para avaliar a qualidade de um conjunto de produtos disponíveis.

Para auxiliar no processo, foi desenvolvida uma ferramenta FMTS, que automatiza o processo de teste proposto, pela geração dos diagramas mutantes, pela geração de um conjunto inicial de produtos a serem avaliados e pela possibilidade de avaliação do conjunto gerado, ou de qualquer outro conjunto fornecido. Com o auxílio desta ferramenta foi conduzido um experimento com quatro LPSs. O experimento mostrou a aplicabilidade da abordagem e que o número de produtos necessários é proporcional ao número de mutantes gerados e/ou ao número de produtos da LPS.

Além disso, foi observado que em média 18% dos mutantes não são mortos pelo conjunto dos mutantes gerados pela abordagem de geração de produtos do teste *pair-wise* utilizando o algoritmo AETG. De modo geral, o conjunto dos produtos gerados pelo *pair-wise* não é capaz de revelar defeitos associados principalmente à cardinalidade das características. A maioria dos defeitos revelados por esta abordagem está relacionada às restrições geradas. Desta forma é possível concluir que a abordagem proposta no trabalho pode ser utilizada de maneira complementar ao teste combinatorial, possibilitando assim a descoberta de diferentes tipos de defeitos, e desta forma, aumentando a confiança de que os produtos de um diagrama de características foram criados de acordo com a especificação.

8.1 Contribuições

São contribuições geradas pelo trabalho:

- Um conjunto de operadores de mutação para diagramas de características de LPS.
- Um processo de teste dividido em três etapas: (i) geração de mutantes; (ii) geração de produtos; e, (iii) avaliação e cálculo do escore de mutação.
- Uma ferramenta automatizada de auxílio ao processo de teste, para geração de mutantes, geração de produtos e avaliação de conjuntos de produtos.
- Resultados do experimento que mostram a viabilidade da aplicação da metodologia apresentada, como forma de seleção de produtos para teste de LPS ou para avaliação de um conjunto de produtos pré-definido.

8.2 Trabalhos Futuros

Dentre as possibilidades de extensão do trabalho apresentado pode-se incluir:

- Realização de novos experimentos de avaliação do conjunto de produtos gerados com escore de mutação de 100% na identificação de defeitos associados não somente a diagramas de características, como por exemplo, defeitos de código, de arquitetura, etc.
- Implementação de um novo avaliador de diagramas de características que possa substituir o FAMA. Pelas características de implementação do FAMA sua utilização em aplicações que trabalham com inúmeras avaliações, como a proposta por este trabalho, se torna extremamente custosa tornando a aplicação lenta. A alteração na ferramenta de avaliação se torna necessária também pelo fato de poder incluir os operadores, DFU e IFU que não foram avaliados.
- Implementação de novos algoritmos de otimização, incluindo algoritmos multiobjetivos para melhores resultados na redução dos produtos necessários sem reduzir o escore de mutação encontrado.
- Condução de experimentos para avaliação de diagramas de características maiores, verificar a escalabilidade da abordagem e refinamento do conjunto inicial de operadores propostos

BIBLIOGRAFIA

[1] M. A. Ardis e D. M. Weiss, “Defining Families: The Commonality Analysis,” em *ICSE'97 Proceedings of the 19th International Conference on Software Engineering*, New York, NY, USA, pp. 671-672, 1997.

[2] F. Bachmann e L. Bass, “Managing Variability in Software Architecture,” em *In Proceedings of the ACM SIGSOFT Symposium on Software*, pp. 126-132, 2001.

[3] D. Benavides, S. Trujillo e P. Trinidad, “On the Modularization of Feature Models,” em *First European Workshop on Model Transformation*, Bilbao, Espanha, 2005.

[4] D. Benavides, S. Segura, P. Trinidad e A. Ruiz-Cortés, “FAMA: Tooling a framework for the automated analysis of feature models.,” *First International Workshop on Variability Modelling of Software intensive Systems,* pp. 129-134, 14 Maio 2007.

[5] D. Beuche e M. Dalgarno, “Software Product Line Engineering with Feature Models,” 2006. [Online]. Available: http://www.pure-systems.com/fileadmin/downloads/pure-variants/tutorials/SPLWithFeatureModelling.pdf. [Acesso em 20 Março 2013].

[6] I. Cabral, M. B. Cohen e G. Rothermel, “Improving the Testing and Testability of Software Product Lines,” em *SPLC'10 Proceedings of the 14th International Conference on Software Product Lines*, Berlin, Heidelberg, pp. 241-255, 2010.

[7] P. Clements e L. Northrop, “Software Product Lines: Practices and Patterns”, Addison-Wesley, 2001.

[8] T. Y. Chen, S. C. Cheung e S. M. Yiu, “Metamorphic testing: a new approach for generating next test cases, Technical Report HKUST-CS98-01,” University of Science and Technology, Hong Kong, 1988.

[9] D. M. Cohen, S. R. Dalal, M. L. Fredman e G. C. Patton, “The AETG System: An Approach to Testing Based on Combinatorial Design,” *IEEE Transactions On Software Engineering,* vol. 23, pp. 437-444, 7 Julho 1997.

[10] M. B. Cohen, M. B. Dwyer e J. Shi, “Coverage and Adequacy in Software Product Line Testing,” em *ROSATEA '06 Proceedings of the ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis*, New York, NY, pp. 53-63, 2006.

[11] J. Czerwonka , “Pairwise Testing - Combinatorial Test Case Generation,” Dezembro 2011. [Online]. Available: http://www.pairwise.org/. [Acesso em 26 4 2012].

[12] P. A. da Mota Silveira Neto, I. C. Machado, J. D. McGregor, E. S. Almeida e S. R. Lemos Meira, “A systematic mapping study of software product lines testing,” *Information and Software Technoloy,* pp. 407-423, 16 (12) 2010.

[13] M. E. Delamaro, M. Jino e J. C. Maldonado, Introdução ao Teste de Software, Rio de Janeiro: Elsevier, 2007.

[14] R. A. DeMillo, R. J. Lipton e F. G. Sayward, “Hints on test data selection: Help for the practicing programmer,” *IEEE Computer,* pp. 31-11, 4 (11) 1978.

[15] M. Deutsch, “Verification and Validation,” em *Software Engineering*, Prentice-Hall, 1978, pp. 329-408.

[16] I. K. El-Far e J. A. Whittaker, “Model-based software testing.,” *Encyclopedia on Software Engineering,* 2001.

[17] E. Engström e P. Runeson, “Software product line testing - A systematic mapping study,” *Information and Software Technology,* pp. 2-13, 29 (7) 2010.

[18] T. L. Grave, M. J. Harrold, J. -M. Kim, A. Porter e G. Rothermel, “An empirical study of regression test selection techniques,” *ACM Transaction on Sofwtare Egineering Methodology,* pp. 184-208, volume 10, 2001.

[19] P. Heymans e J.-C. Trigaux, “Software Product Lines: State of the art,” *Technical Report for PLENTY project, Institut d’Informatique FUNDP*, pp. 13-37, 2003.

[20] IEEE, “IEEE Standard Glossary of Software Engineering Terminology,” *Standard 610.12-1990. IEEE Computer Society Press*, 2002.

[21] I. Jacobson, M. Griss e P. Jonsson, Software Reuse. Arquiteture, Process and Organization for Business Success., Addison-Wesley, 1997.

[22] K. Kang, S. Cohen, J. Hess, W. Novak e S. Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study,” Novembro 1990. [Online]. Available: http://www.sei.cmu.edu/reports/90tr021.pdf. [Acesso em 27 4 2012].

[23] K. C. Kang, J. Lee e P. Donohoe, “Feature-Oriented Product Line Engineering,” *IEEE Software,* pp. 58-65, Julho/Setembro 2002.

[24] R. Kuhn, Y. Lei e R. Kacker, “Practical Combinatorial Testing: Beyond Pairwise,” Maio/Junho 2008. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/itpro-final.pdf. [Acesso em 15 Maio 2013].

[25] R. Kuhn, R. Kacker, Y. Lei e J. Hunter, “Combinatorial Software Testing,” Agosto 2009. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/documents/kuhn-kacker-lei-hunter09.pdf. [Acesso em 15 Maio 2013].

[26] C. T. R. Lai e D. M. Weiss, Software Product-Line, Addison-Wesley, 1999.

[27] B. P. Lamancha e M. P. Usaola, “Testing Product Generation in Software Product Lines Using Pairwise for Features Coverage,” em *ICTSS'10 Proceedings of the 22nd IFIP WG 6.1 International Conference on Testing Software and Systems*, Berlin, pp. 111-125, 2010.

[28] B. P. Lamancha, M. P. Usaola e M. P. Velthius, “Software product line testing, a systematic review,” *International Conference on Software Paradigm Trends,* vol. 49, pp. 78-81, 2009.

[29] P. C. Len Bass e K. Rick, Software Architecture in Practice. SEI Series in Software Enginnering, Segunda Edição ed., Addison-Wesley, 2003.

[30] J. D. McGregor, A Pratical Guide to Testing Object-Oriented Software, Addison Wesley, 2001.

[31] J. D. McGregor, “Toward a Fault Model for Software Product Lines,” em *12th International Software Product Line Conference*, Limerick, pp. 157-162, 2008.

[32] G. J. Myers, “The Art of Software Testing,” 2 ed., New Jersey, John Wiley & Sons, 2004.

[33] C. Nie e H. Leung, “A survey of combinatorial testing,” *ACM Computing Surveys (CSUR),* vol. 43, pp. 11:1 - 11:29, 2011.

[34] L. M. Northrop, “SEI's Software Product Line Tenets,” *IEEE Software,* n. Julho/Agosto, pp. (19);32-40, 2002.

[35] L. M. Northrop e P. C. Clements , “A Framework for Software Product Line Practice, Version 5.0,” Julho 2007 . [Online]. Available: http://www.sei.cmu.edu/productlines/frame_report/what.is.a.PL.htm. [Acesso em 6 Fevereiro 2012].

[36] S. Oster, M. Zink, M. Lochau e M. Grechanik, “*Pair-wise* feature-interaction testing for SPLs: potentials and limitations,” *15th International Software Product Line Conference,* pp. 6:1-6:8, 2011.

[37] G. Perrouin, S. Sen, J. Klein, B. Baudry e Y. le Traon, “Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines,” em *Third International Conference on Software Testing, Verification and Validation*, Luxembourg, pp. 459-468, 2010.

[38] R. S. Pressman, em *Engenharia de Software*, Pearson, 2010, pp. 787-822.

[39] A. Reuys, S. Reis, E. Kamsties e K. Pohl, “The ScenTED Method for Testing Software Product Lines,” *PFE'06 Proceedings of International Workshop of Software Product Family Engineering,* pp. 479-520, 2006.

[40] S. J. Russell e P. Norving, Artificial Intelligence: A Modern Approach, Upper Saddle River, New Jersey: Prentice Hall, 2003.

[41] P. Schobbens, P. Heymans, J. Trigaux e Y. Bontemps, “Feature Diagrams: A Survey and A Formal Semantics,” em *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, pp. 139-148, 2006.

[42] S. Segura, R. M. Hierons, D. Benavides e A. Ruiz-Cortés, “Automated Test Data Generation on the Analyses of Feature Models: A Metamorphic Testing Approach,” em *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, Washington, DC, USA, 2009, pp 35-44.

[43] M. Shafique e Y. Labiche, “A Systematic Review of Model Based Testing Tool, Technical Report SCE-10-04,” Carleton University, Ottawa, Canada, 2010.

[44] Z. Stephenson, Y. Zhan, J. Clark e J. McDermid, “Test Data Generation for Product Lines - A Mutation Testing Approach,” em *Proceedings of the International Workshop on Software Product Line Testing SPLIT 2004*, Boston, 2004.

[45] A. Tevanlinna, J. Taina e R. Kauppinen, “Product Family Testing - a Survey,” *ACM SIGSOFT Software Engineering Notes,* p. 12, 29 (2) 2004.

[46] S. Weissleder, D. Sokenou e B.-H. Schlingloff, “Reusing State Machines for Automatic Test Generation in Product Lines,” em *MoTiP'08 1st Workshop on Model-based Testing in Practice*, Berlin, Germany, 2008.

[47] D. Weiss, “Software Synthesis: The FAST Process,” em *Proceedings of the International Conference on Computing in High Energy Physics*, 1995.

[48] E. J. Weyuker, “On testing non-testable programs,” *The Computer Jornal,* pp. 465-470, 25 (4) 1982.

[49] W.E. Wong, A. Mathur, and J. Maldonado, “Mutation x all-uses: An empirical evaluation of cost, strength and effectiveness,” em *Soft. Quality and productivity Theory, Practice, Education and Training,* 1994.

[50] Z. Q. Zhou, T. H. Tse, Z. Yang, H. Huang e T. Y. Chen, “Metamorphic testing and its applications,” em *Proceeding of the 8th International Symposium on Future Software Technology*, pp. 20-22, 2004.

[51] C. A. Zorzo, R. A. Lopes, T. E. Colanzi e S. R. Vergilio, “Aplicação de uma Estratégia Incremental para o Teste de Linha de Produto de Software,” em *X Simpósio Brasileiro de Qualidade de Software*, Curitiba, Paraná, 2011.

APÊNDICE A – OPÇÕES DA APLICAÇÃO

Esse apêndice contém os comandos disponíveis na FMTS além de ilustrações de telas e resultados.

TELA DE ABERTURA

A seguinte tela de abertura é apresentada (Figura A.1)

```
Welcome to the FMTS
---------------------------------------------------
For bug reporting: jmferreira@inf.ufpr.br
Version 1.0 – 2012

Type <help> for assistance.
```

Figura A.1 Tela de apresentação da FMTS.

OPÇÕES E COMANDOS

help

O comando *help* exibe as opções de comandos disponíveis na aplicação.

```
Command                                Description
-----------------------------------------------------
help                                   : Display Menu
load <feature_model_path>              : Load Feature Model to be tested
p-gen                                  : Automatically generate test cases
p-add <featA>, <featB>,..., <featN>    : Add a new product to the test cases
p-print                                : Print all test cases
p-initial                              : Print initial set of products generated
p-valid                                : Print list with prod. validation result
m-gen                                  : Generate mutants
m-print                                : Print all mutants
m-dead                                 : Print mutants status
m-print <mutant number>                : Print a mutants
test                                   : Execute test cases
fm-print                               : Print feature model under testing
eq                                     : Check for equivalent diagrams
score                                  : Print mutation score
save-score                             : Save mutation score
hc                                     : Execute Hill Climbing algorithm
exit                                   : Exit Ultimate Feature Model Test Suit
```

Figura A.2 Resultado do comando help

load

O comando *load* carrega um diagrama de características em formato XML para a aplicação a partir de um caminho válido fornecido. Logo após o carregamento a aplicação se encarrega, utilizando o FAMA, de realizar a verificação de validade do diagrama.

```
$ load F:\johnny\Downloads\FaMa-1.1.1\fm-samples\CAS.xml
Loading feature into FAMA: \fm-samples\CAS.xml
Validating Feature Model
Feature Model: OK
```

Figura A.3 Resultado do comando load para o diagrama CAS

p-gen

O comando *p-gen* gera os casos de teste para o diagrama de características fornecido.

```
$ p-gen
Creating Test Cases
DFS #7
IFL #5
AFS #9
RFS #6
RSR #2
DRL #0
IRL #0
DRU #0
IRU #0
FDC #1
FEC #0
RDC #449
REC #437
CDC #0
CEC #0
Test cases were created successfully: #916
```

Figura A.4 Resultado do comando p-gen para o diagrama CAS.

p-add

O comando *p-add* adiciona um produto fornecido pelo usuário ao conjunto casos de teste.

```
$ p-add CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CD CONTROL TITLE-CHANEL-
SELECTION FORWARD-BACKWARD VOLUME SWITCH
Product added successfully
```

Figura A.5 Resultado do comando p-add.

p-print

O comando *p-print* exibe na tela todo o conjunto de casos de teste. Seja esse composto por produtos gerados automaticamente, fornecidos pelo usuário ou ambos.

```
$ p-print
1:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL PLAYBACK DVD CONTROL TITLE-
CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH MEDIA-FORMAT AAC WMA
WAV
2:CAS TRAFIC-MESSAGE-CHANEL NAVIGATION-SYSTEM PLAYBACK DVD CONTROL
TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH MEDIA-FORMAT AAC
WMA WAV
3:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL NAVIGATION-SYSTEM PLAYBACK
DVD CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH
MEDIA-FORMAT AAC WMA WAV
4:CAS TRAFIC-MESSAGE-CHANEL PLAYBACK USB DVD CONTROL TITLE-CHANEL-
SELECTION FORWARD-BACKWARD VOLUME SWITCH MEDIA-FORMAT AAC WMA WAV
5:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL PLAYBACK USB DVD CONTROL
TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH MEDIA-FORMAT AAC
WMA WAV

...

914:CAS TRAFIC-MESSAGE-CHANEL NAVIGATION-SYSTEM MAP-DATA-VIA-USB
PLAYBACK USB DVD CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD
VOLUME SWITCH MEDIA-FORMAT MP3 AAC WMA WAV
915:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL NAVIGATION-SYSTEM MAP-
DATA-VIA-USB PLAYBACK USB DVD CONTROL TITLE-CHANEL-SELECTION FORWARD-
BACKWARD VOLUME SWITCH MEDIA-FORMAT MP3 AAC WMA WAV
Created by User Test Cases:
0:CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CD CONTROL TITLE-CHANEL-SELECTION
FORWARD-BACKWARD VOLUME SWITCH
```

Figura A.6 Resultado do comando p-print

p-valid

O comando *p-valid* exibe na tela a lista dos produtos do conjunto de caso de teste e o status, informando se este é valido ou não para a linha de produto a ser testada.

p-initial

O comando *p-initial* exibe na tela o resultado da listagem de todos os produtos válidos para o diagrama a ser testado, gerados pelo FAMA.

```
$ p-initial
0:CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CD CONTROL TITLE-CHANEL-SELECTION
FORWARD-BACKWARD VOLUME SWITCH
1:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL PLAYBACK CD CONTROL TITLE-
CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH
2:CAS TRAFIC-MESSAGE-CHANEL NAVIGATION-SYSTEM PLAYBACK CD CONTROL
TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH
3:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL NAVIGATION-SYSTEM PLAYBACK
CD CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH
4:CAS TRAFIC-MESSAGE-CHANEL NAVIGATION-SYSTEM MAP-DATA-VIA-CD PLAYBACK
CD CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH

...

445:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL PLAYBACK USB DVD CONTROL
TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH MEDIA-FORMAT MP3
AAC WMA WAV
446:CAS TRAFIC-MESSAGE-CHANEL NAVIGATION-SYSTEM PLAYBACK USB DVD
CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH MEDIA-
FORMAT MP3 AAC WMA WAV
447:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL NAVIGATION-SYSTEM PLAYBACK
USB DVD CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH
MEDIA-FORMAT MP3 AAC WMA WAV
448:CAS TRAFIC-MESSAGE-CHANEL NAVIGATION-SYSTEM MAP-DATA-VIA-USB
PLAYBACK USB DVD CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD
VOLUME SWITCH MEDIA-FORMAT MP3 AAC WMA WAV
449:CAS TRAFIC-MESSAGE-CHANEL WHEEL-CONTROL NAVIGATION-SYSTEM MAP-
DATA-VIA-USB PLAYBACK USB DVD CONTROL TITLE-CHANEL-SELECTION FORWARD-
BACKWARD VOLUME SWITCH MEDIA-FORMAT MP3 AAC WMA WAV
```

Figura A.7 Resultado do comando p-initial para o diagrama CAS.

p-valid

O comando *p-valid* exibe na tela a lista dos produtos do conjunto de caso de teste e o status, informando se este é valido ou não para a linha de produto a ser testada.

m-print

O comando *m-print* exibe na tela o diagrama de características mutante, no formato XML, de acordo com o número do mutante fornecido.

m-gen

O comando *m-gen* gera os diagramas mutantes para o diagrama a ser testado.

```
$ m-gen
Creating mutant diagrams
This operation might take several minutes to complete. Please wait for
message informing operation is over!
DFL #0
IFL #1
DRL #0
IRL #1
DRU #1
IRU #1
REC #1
RDC #2
FDC #22
FEC #11
CDC #152
CEC #77
RSR #2
RFS #6
mutants created sucessfully: #277
```

Figura A.8 Resultado do comando m-gen.

m-dead

O comando *m-dead* exibe na tela a lista dos mutantes e seus status, informado, após a execução do comando *test* quais mutantes foram mortos e quais estão vivos.

test

O comando *test* realiza a verificação de validação dos casos de teste para o diagrama a ser testado e para todos os diagramas mutantes.

fm-print

O comando *fm-print* exibe na tela o diagrama de características a ser testado, no formato XML.

eq

O comando *eq* executa a verificação dos diagramas equivalentes, adicionando na lista de casos de teste cada novo produto encontrado capaz de matar o diagrama sendo avaliado.

score

O comando *score* exibe o resultado do cálculo do escore de mutação, levando em conta a equação descrita no Capítulo 3. O valor do escore de mutação é representado por um *float* com precisão de duas casas decimais. No caso do comando *score* ser executado previamente ao comando *eq*, o número de mutantes equivalentes não serão contabilizados no cálculo do escore.

save-score

O comando *save-score* gera uma planilha no format csv listando qual o status de teste de todos os produtos fornecidos/gerados para todos os mutantes. No formato como o exemplo apresentado na Tabela A.1. No exemplo, temos situações como: (i) o produto 1 mata apenas o mutante 4; (ii) o produto 2 mata os mutantes 2 e 3; (iii) o produto 3 não mata nenhum dos diagramas mutantes; (iv) o produto 4 mata o mutante 2; e (v) nenhum dos produtos consegue matar o diagrama 1.

Tabela A.1 Exemplo de saída do comando save-score.

Produtos/Mutantes	**1**	**2**	**3**	**4**
1	VIVO	VIVO	ALIVE	MORTO
2	VIVO	MORTO	MORTO	VIVO
3	VIVO	VIVO	VIVO	VIVO
4	VIVO	MORTO	ALIVE	VIVO

hc

O comando *hc* executa o algoritmo de otimização de subida de encosta (*hill climbing*) visando reduzir o número de casos de teste necessários para satisfazer o critério do teste de mutação.

APENDICE B – DIAGRAMAS DE CARACTERISTICAS

A seguir são apresentados os diagramas de características das LPS utilizadas neste trabalho (Figura B.1, B.2 e B.3).

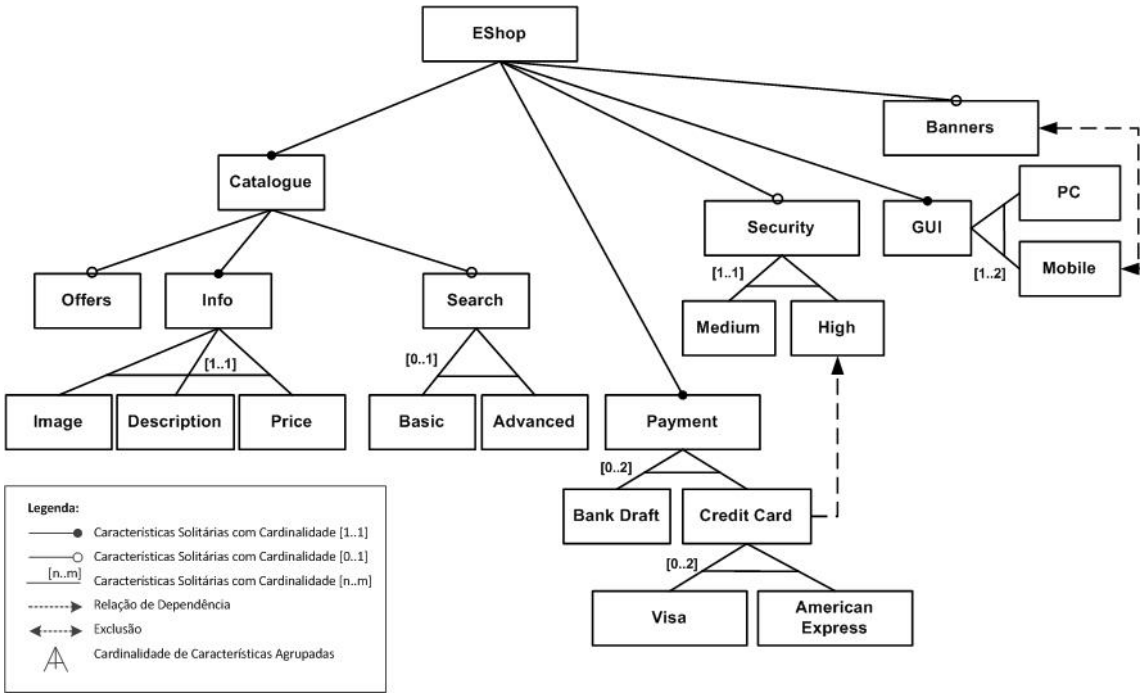

Figura B.1 Diagrama de características da LPS E-Shop.

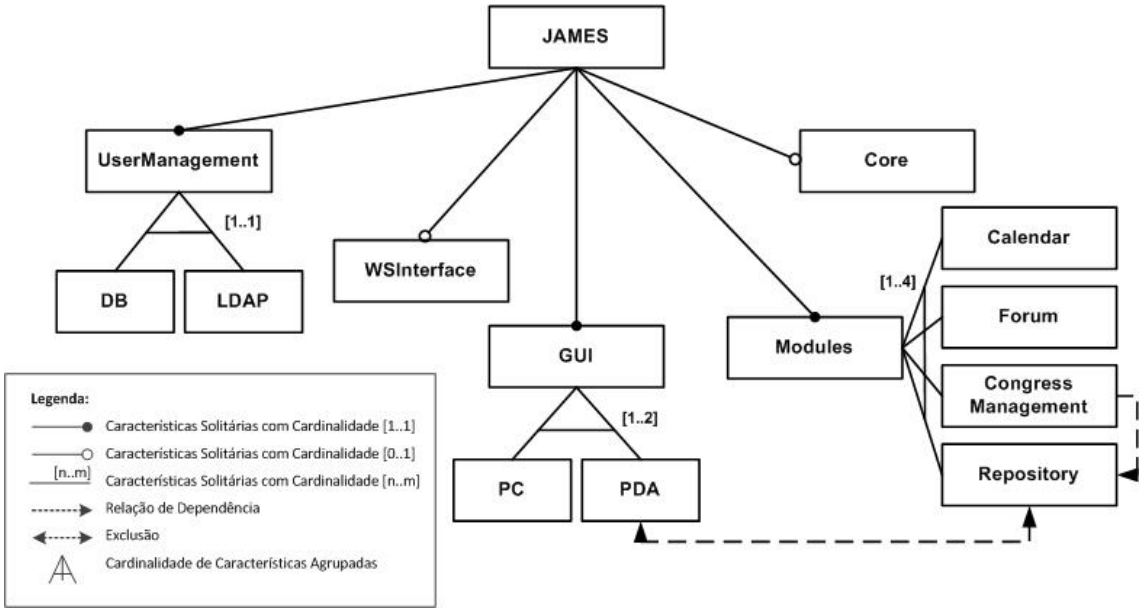

Figura B.2 Diagrama de características da LPS JAMES.

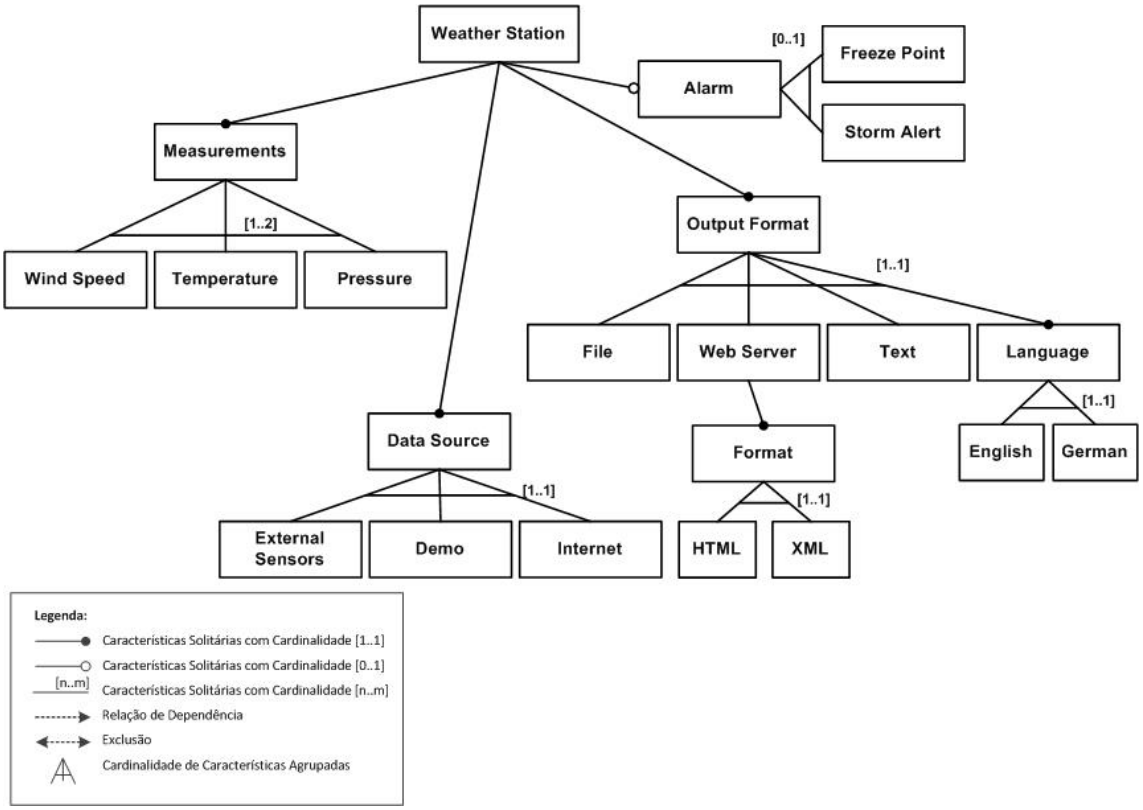

Figura B.3 Diagrama de características da LPS Weather Station.

APENDICE C – CAS CONJUNTO *PAIR-WISE* E AETG SET

Tabela C.1 Conjunto dos *pair-wises* para a LPS CAS.

Pair-wise	
WheelControl	NavigationSystem
WheelControl	MediaFormat
WheelControl	MapDataviaCD
WheelControl	MapDataviaUSB
WheelControl	CD
WheelControl	Cassete
WheelControl	DVD
WheelControl	USB
WheelControl	AAC
WheelControl	WMA
WheelControl	WAV
WheelControl	MP3
NavigationSystem	MediaFormat
NavigationSystem	MapDataviaCD
NavigationSystem	MapDataviaUSB
NavigationSystem	CD
NavigationSystem	Cassete
NavigationSystem	DVD
NavigationSystem	USB
NavigationSystem	AAC
NavigationSystem	WMA
NavigationSystem	WAV
NavigationSystem	MP3
MediaFormat	MapDataviaCD
MediaFormat	MapDataviaUSB
MediaFormat	CD
MediaFormat	Cassete
MediaFormat	DVD
MediaFormat	USB
MediaFormat	AAC
MediaFormat	WMA
MediaFormat	WAV
MediaFormat	MP3
MapDataviaCD	MapDataviaUSB
MapDataviaCD	CD
MapDataviaCD	Cassete
MapDataviaCD	DVD
MapDataviaCD	USB

MapDataviaCD	AAC
MapDataviaCD	WMA
MapDataviaCD	WAV
MapDataviaCD	MP3
MapDataviaUSB	CD
MapDataviaUSB	Cassete
MapDataviaUSB	DVD
MapDataviaUSB	USB
MapDataviaUSB	AAC
MapDataviaUSB	WMA
MapDataviaUSB	WAV
MapDataviaUSB	MP3
CD	Cassete
CD	DVD
CD	USB
CD	AAC
CD	WMA
CD	WAV
CD	MP3
Cassete	DVD
Cassete	USB
Cassete	AAC
Cassete	WMA
Cassete	WAV
Cassete	MP3
DVD	USB
DVD	AAC
DVD	WMA
DVD	WAV
DVD	MP3
USB	AAC
USB	WMA
USB	WAV
USB	MP3
AAC	WMA
AAC	WAV
AAC	MP3
WMA	WAV
WMA	MP3

Tabela C.2 Conjunto dos produtos gerados pelo algoritmo AETG.

Produtos Gerados pelo AETG
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL NAVIGATION-SYSTEM MEDIA-FORMAT MAP-DATA-VIA-CD CD USB AAC MP3
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL NAVIGATION-SYSTEM MEDIA-FORMAT MAP-DATA-VIA-CD CASSETE USB AAC MP3
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL NAVIGATION-SYSTEM MEDIA-FORMAT MAP-DATA-VIA-CD CASSETE WAV
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL NAVIGATION-SYSTEM MEDIA-FORMAT MAP-DATA-VIA-USB DVD USB WMA MP3
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL NAVIGATION-SYSTEM MAP-DATA-VIA-CD CD
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL MEDIA-FORMAT CASSETE USB WMA MP3
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL DVD
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH WHEEL-CONTROL CD
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH NAVIGATION-SYSTEM MEDIA-FORMAT MAP-DATA-VIA-CD CD USB WMA MP3
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH NAVIGATION-SYSTEM MEDIA-FORMAT MAP-DATA-VIA-CD DVD USB AAC
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH NAVIGATION-SYSTEM MAP-DATA-VIA-USB CASSETE
CAS TRAFIC-MESSAGE-CHANEL PLAYBACK CONTROL TITLE-CHANEL-SELECTION FORWARD-BACKWARD VOLUME SWITCH MEDIA-FORMAT CD USB WAV