UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

JEFFERSON DA SILVA BARBOSA

EXTRAÇÃO, SÍNTESE E AVALIAÇÃO DE MODELOS DE FEATURES EM LINHAS DE PRODUTO DE SOFTWARE

QUIXADÁ
2016

JEFFERSON DA SILVA BARBOSA

EXTRAÇÃO, SÍNTESE E AVALIAÇÃO DE MODELOS DE FEATURES EM LINHAS DE PRODUTO DE SOFTWARE

Monografia apresentada ao Curso de Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Orientadora Prof[a]. Msc. Carla Ilane Moreira Bezerra

QUIXADÁ
2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

B198e Barbosa, Jefferson da Silva
Extração, síntese e avaliação de modelos de features em linhas de produto de software/ Jefferson da Silva Barbosa. – 2016.
70 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Bacharelado em Engenharia de Software, Quixadá, 2016.
Orientação: Profa. Msc. Carla Ilane Moreira Bezerra
Área de concentração: Computação

1. Software – Desenvolvimento 2. Desenvolvimento de produto 3. Software – Controle de qualidade I. Título.

CDD 005.14

JEFFERSON DA SILVA BARBOSA

EXTRAÇÃO, SÍNTESE E AVALIAÇÃO DE MODELOS DE FEATURES EM LINHAS DE PRODUTO DE SOFTWARE

Monografia apresentada ao Curso de Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel. Área de concentração: computação

Aprovado em: 02 / fevereiro / 2016.

BANCA EXAMINADORA

_____

Prof[a]. MSc. Carla Ilane Moreira Bezerra
Universidade Federal do Ceará-UFC

_____

Prof[a]. Dra. Rossana Maria de Castro Andrade
Universidade Federal do Ceará-UFC

_____

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará-UFC

AGRADECIMENTOS

À Deus por me promover essa paz interior e serenidade que foram úteis durante toda minha vida para realizar tomadas de decisões.

Aos meus pais pelos conselhos, pelo apoio em todas as minhas decisões, pela educação e pelo carinho.

À Profa. Msc. Carla Ilane Moreira Bezerra, por seu excelente trabalho como orientadora e professora, pelas as oportunidades, pela confiança e pela sua exigência com os trabalhos realizados, que proporcionou um aumento da qualidade deste trabalho.

Aos professores da banca examinadora, Profa. Dra. Rossana Maria de Castro Andrade e Prof. Dr. José Maria da Silva Monteiro Filho, pelas colaborações e sugestões.

À Prof. Msc. Samy Soares Passos de Sá, por me ajudar com direcionamentos em algumas partes do meu trabalho.

À instituição por fornecer um ensino de alta qualidade com professores muito bem capacitados. Gostaria de agradecer em especial a Coordenação de Engenharia de Software que sempre foi muito atenciosa para minhas necessidades, realizando um trabalho de qualidade, com o mínimo de burocracia.

À toda equipe do GREat, por serem receptivos e pela transmissão de conhecimento recebida durante todo o estágio.

Aos meus colegas Rogério, Thiago, Segio, Laísa, Leúson, Erick, João Carlos, João Marcos, Jonas, Lucas, Adail, Natasha e Marcilio pela a amizade e união durante todos esses anos.

"Eu acredito demais na sorte. E tenho constatado que, quanto mais duro eu trabalho, mais sorte eu tenho." (Coleman Cox).

RESUMO

Desde o surgimento das Linhas de Produto de Software, elas têm se mostrado uma estratégia de sucesso para a construção de produtos com características relacionadas. Ao longo dos anos, a forma de representação das características comuns e variáveis de uma família de produtos utilizando modelos de *features* tem se tornado cada vez mais popular. *Feature* pode ser definida como uma unidade lógica de comportamento especificado por um conjunto de requisitos que é relevante para alguma das partes interessadas. Embora um modelo de *features* possa ajudar no gerenciamento de uma LPS, essa produção demanda uma grande quantidade de tempo e esforço. Para reduzir esse tempo e esforço na criação de modelos de *features*, foram desenvolvidos alguns processos de engenharia reversa que é utilizado para a sintetização de um modelo de *features*. A construção dos modelos a partir do processo de engenharia reversa é feita utilizando um conjunto de configurações e dependências que contêm uma hierarquia de recursos. A partir disso, o processo de engenharia reversa define um conjunto de passos de execução para extrair um ou mais modelos. O objetivo deste trabalho é definir um algoritmo a ser implantado na ferramenta DyMMer que realiza a construção de vários modelos de *features* com estruturas distintas, utilizando um processo de engenharia reversa que a partir de um conjunto de configurações dos produtos é realizado a síntese de vários modelos de *features*. Para realizar esta avaliação são utilizadas medidas da literatura para serem aplicadas diretamente na estrutura dos modelos de *features*. A metodologia foi dividida em três partes: (i) levantamento e estudo das ferramentas disponíveis para realizar avaliação de modelos de *features*, (ii) coleta de medidas, e (iii) avaliação dos modelos sintetizados resultantes da elaboração que tem como entrada as configurações de vários produtos. Como resultado deste trabalho, foi possível definir um conjunto de passos para a extração de modelos de *features* a partir de um conjunto de configurações dos produtos utilizando engenharia reversa e uma maneira de avaliar a estrutura do modelo sem o conhecimento do domínio.

Palavras-chave: Linha de Produto de Software. Modelo de *feature*. Avaliação da Qualidade. Engenharia Reversa.

ABSTRACT

Since the advent of software product lines, they have proved a successful strategy for building products with related features. Over the years, the way of representing characteristics common and variables of a family of products using a feature model has become increasingly popular. Feature may be defined as a logical unit of behavior specified by a set of requirements that is relevant to any of the stakeholders. Although a model of features may help manage a SPL, this production requires a lot of time and effort. To reduce this time and effort in creating feature models were developed some reverse engineering process that is used for synthesizing a feature model. The construction of models from the process of reverse engineering is performed using a set of configurations and dependencies containing a resource hierarchy. From this, the reverse engineering process defines a set of execution steps to extract one or more models. The objective of this work is to define an algorithm to be implemented in DyMMer tool to perform the construction of various features models with different structures, using a reverse engineering process from a set of product configurations is performed the synthesis of various features models. To accomplish this evaluation was used measures proposed in the literature to be applied directly in the structure of feature models. The methodology was divided into three parts: (i) research and study of the tools available to perform evaluation feature models, (ii) collect measures, and (iii) evaluation of synthesized models resulting from the elaboration which has as input a set of product configurations. As a result of this work, it was possible to define a set of steps for the feature models extraction from a set of product configurations using reverse engineering and a way to evaluate a model structure without the knowledge domain.

Keywords: Software Product Line. *Feature* Model. Quality Evaluation. Reverse Engineering.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Em um mercado cada vez mais competitivo, o desenvolvimento de softwares individuais nem sempre é uma alternativa viável. Muitas empresas, em diferentes domínios, produzem uma grande variedade de produtos de software semelhantes utilizando reuso, que além de trazer maior competitividade o reuso também ajuda no aumento da qualidade (ACHER *et al*., 2013). O reuso pode proporcionar menores custos de produção e manutenção de software, entregas mais rápida de sistemas e maior produtividade (SOMMERVILLE, 2011). No Brasil, o reuso de software é um fator que independe do tamanho da organização e da equipe, o que o torna praticável em todos os tipos de empresas dependendo da experiência da equipe nesse ramo e equipes especializadas no desenvolvimento para reuso (LUCRÉDIO *et al*., 2007).

Uma das estratégias utilizadas para o aumento da qualidade por meio do reuso de software é utilização de Linha de Produto de Software (LPS). O conceito de LPS é definido como uma reutilização sistemática de um conjunto de subsistemas de software e interfaces, através da exploração de pontos comuns e na gestão de variabilidade entre os produtos, que são estabelecidos sob uma estrutura comum a partir do qual um conjunto de produtos derivados pode ser desenvolvido e produzido de forma eficiente (PEREIRA *et al*., 2011; BÖCKLE, POHL e LINDEN, 2005).

Uma das formas de representar os produtos de uma LPS é por meio de modelos de *features*. Uma *feature* pode ser definida como "uma unidade lógica de comportamento especificado por um conjunto de requisitos funcionais e não funcionais" (BOSCH, 2000). Um conjunto de *features* é normalmente utilizado em uma LPS para ajudar a definir uma propriedade do sistema que é relevante para alguma das partes interessadas (FERNANDES; WERNER; TEXEIRA, 2010). Com esse conjunto de *features* é possível criar modelos que nos permite delinear as propriedades comuns e variáveis dos membros da linha de produto ao longo das fases de engenharia desse produto, além disso, pode-se iniciar um processo de engenharia para auxiliar na decisão de quais recursos devem ser apoiados (FERNANDES; WERNER; MURTA, 2008).

Desde o surgimento das LPSs, ela tem se mostrado uma estratégia de sucesso para a construção de produtos com características relacionadas. Ao longo dos anos, a forma de representação das características comuns e variáveis de uma família de produtos através de modelos de *features*, tem se tornado cada vez mais popular. Para auxiliar na modelagem dos

modelos de *features* foram desenvolvidas diversas ferramentas, como por exemplo: Gears[1], pure::variants[2], *Feature*IDE[3], FAMILIAR[4], SPLOT[5] e FAMA[6] que usam modelos de *features* para descrever a variabilidade dos seus produtos (CAPILLA; BOSCH, 2011).

Apesar do auxílio das ferramentas, a modelagem de *features* em grandes projetos acaba tomando muito tempo de um Engenheiro de Domínio. Devido a este alto custo de tempo foi adaptado alguns processos de engenharia reversa para a automação da criação de modelos de *features* a partir de suas configurações (SHE *et al*., 2011).

Ksynthesis é um exemplo de operador que suporta a modelagem de *feature* por meio da engenharia reversa. Sua eficácia, entretanto, não ultrapassa 60% utilizando tanto técnicas lógicas quanto ontológicas, o que pode exigir uma avaliação da qualidade dos modelos de *features* gerados para verificar o melhor modelo que se adequa ao contexto da LPS (BÉCAN *et al*., 2014).

Tendo em vista a importância da avaliação da qualidade de um determinado modelo de *feature* foi apresentado no trabalho de Bezerra *et al*., (2014), um catálogo de medidas para avaliação da qualidade do modelo de *features*. No trabalho de Bagueri e Gasevic (2011) foi proposto um conjunto de métricas estruturais para modelos de *features* com a finalidade de abranger várias características de qualidade de um modelo de *feature*. De acordo com a ISO/IEC 25010 padrão (SQuaRE, 2011) "Uma medida é um mapeamento de uma entidade para um número ou para um símbolo, a fim de caracterizar uma propriedade da entidade".

Para avaliação da qualidade dos modelos de *features* a partir de medidas, foi construída a ferramenta DyMMer (HOLANDA JÚNIOR, 2014). A ferramenta disponibiliza funcionalidades de visualização de um modelo com ou sem contexto, edição de um modelo com ou sem contexto e exportação de todos os resultados das medidas aplicadas em um modelo de *feature*.

Tendo em vista os problemas de implantação de uma LPS em um projeto, este trabalho foi desenvolvido com intuito de diminuir o tempo e esforço de elaboração de um modelo de *features*, evitar inserção de erros manuais, evitar que os erros sejam propagados para as próximas fases de engenharia de uma LPS, diminuir custos, facilitar automação da

[1] http://www.biglever.com/solution/product.html
[2] http://www.pure-systems.com/pure_variants.49.0.html
[3] http://marketplace.eclipse.org/node/22848
[4] http://familiar-project.github.io/
[5] http://www.splot-research.org/
[6] http://www.isa.us.es/fama/

construção de modelos de *features* e facilitar o desenvolvimento de um modelo de *feature* independentemente do contexto em que ele é aplicado.

Dessa forma, para o desenvolvimento desse trabalho foi levantada a questão: Dentre todos os modelos de *features* gerados qual o modelo deve ser selecionado para representar uma determinada LPS (LOPEZ-HERREJON *et al*., 2015)? A fim de responder a esta pergunta, este trabalho usa um conjunto de medidas propostas por Bezerra et al. (2014) para realizar uma avaliação dos modelos de *features* sintetizados e definir a melhor estrutura de acordo com o conjunto de configurações que é recebida como entrada do algoritmo.

Neste cenário, este trabalho tem como objetivo definir um algoritmo a ser implantado na ferramenta DyMMer que realiza a construção de vários modelos de *features* com estruturas distintas, utilizando um processo de engenharia reversa que a partir de um conjunto de configurações dos produtos é realizado a síntese de vários modelos de *features* e logo após é feito a avaliação para identificar a melhor estrutura entre todas as geradas com intuito de representar uma LPS. A definição do algoritmo teve como base trabalhados realizados anteriormente pelos autores She *et al*. (2014) e Czarnecki, Wasowski (2007). Após a definição do algoritmo foram estudadas regras matemáticas definidas por Cayley (1881) com intuito de restringir as estruturas geradas pelo algoritmo.

De acordo com o objetivo desse trabalho é possível identificar como público alvo, acadêmicos com pesquisas na área da qualidade de modelos de *features* e tem como necessidade analisar grandes modelos de *features* com o menor custo de tempo possível e pequenas, médias e grandes empresas desenvolvedoras de software que possuam atividades de reuso de software e desejam melhorar seu gerenciamento através dos modelos de *features*.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados que foram utilizados como fontes de pesquisa como parte principal para elaboração deste trabalho. Seção 3 descreve os conceitos de Linha de Produto de Software apresentando uma visão geral das fases de engenharia, também é apresentado o conceito de modelo de *features* onde é feito uma visão abrangente. Logo após é descrito cenários onde é realizado a sintetização dos modelos de *features*. Por último é descrita a avaliação da Qualidade, apresentado o catalogo de medidas utilizadas para a avaliação dos modelos sintetizados. Na Seção 4 são apresentados os passos utilizados para realizar este trabalho. Na Seção 5 são apresentados os passos executados para realizar a síntese do modelo de *features*. A Seção 6 apresenta as avaliações realizadas sobre os modelos de *features* gerados pelo algoritmo proposto. A seção 7 apresenta uma discussão sobre os resultados obtidos. Finalmente, na Seção 8 é realizada as considerações finais do trabalho.

2 TRABALHOS RELACIONADOS

O trabalho que foi desenvolvido tomou como base as metodologias dos trabalhos relacionados para definir um conjunto de passos para realizar a extração de uma fórmula proposicional a partir de um conjunto de configurações de produtos com características semelhantes mapeados em uma matriz, onde essa fórmula proposicional representa as relações entre as *features* de um determinado modelo. Também foram listados os trabalhos usados para coletar as medidas utilizadas para realizar as análises sobre os modelos resultantes da síntese, trabalhos que abordam os cenários de engenharia reversa sobre os modelos de *features* e o trabalho de Holanda Junior (2014) que desenvolveu a ferramenta DyMMer na qual foi utilizada para coletar as medidas dos modelos de *features*.

Archer *et al*., (2013), apresenta um processo de sintetização de modelos de *features* utilizando para isso uma técnica de engenharia reversa. O trabalho de Archer utiliza o operador *ksynthesis* para dar suporte à modelagem de *features* sintetizadas. Este trabalho usou alguns dos passos do processo de engenharia reversa apresentados por Archer. Vale ressaltar que não foi utilizado o operador *ksynthesis* para este trabalho devido à necessidade de ter um conhecimento semântico sobre os modelos, que foge do escopo e do objetivo deste trabalho.

Bezerra *et al*., (2014), tem como objetivo identificar um conjunto de características, atributos e medidas de qualidade para avaliar modelos de features de LPS. Assim como Bezerra *et al*., (2014) este trabalho propõe uma forma de avaliar o modelo de *features* utilizando medidas, que estão associadas com as características de qualidade do modelo de *features* e seus atributos de qualidade. O trabalho de Bezerra *et al*., (2014), ainda apresenta um catálogo de medidas que será usado neste trabalho para a avaliação da qualidade dos modelos de *features* gerados pelo algoritmo.

O trabalho de Czarnecki, Wasowski (2007) e She *et al*. (2014) apresentam a extração de um hipergrafo usando um algoritmo que é composto por um conjunto de passos para identificar os tipos de relações entre as *features* onde esse algoritmo tem como entrada uma fórmula binária que são representadas na Forma Normal Conjuntiva (FNC). Em especial o trabalho de She *et al*. (2014), também apresenta uma maneira de extração utilizando fórmulas na Forma Normal Disjuntiva (FND). Também é descrito que a partir deste hipergrafo é possível gerar diferentes modelos de *features*. Contudo, esses trabalhos não mencionam como é feita a extração desses modelos de *features* e não fazem qualquer avaliação sobre os possíveis modelos de *features* gerados a partir do gráfico. A partir desses

trabalhos foi possível a extração de alguns passos para realizar o mapeamento de uma matriz de configuração para uma fórmula proposicional.

Haslinger, Lopez-Herrejon e Egyed (2011) propõem a extração de modelos de *features* a partir de uma matriz de configurações onde é dado um conjunto de etapas em um algoritmo para atribuir os tipos de relações entre as *features*. No entanto, o trabalho gera apenas um modelo e esse modelo não é avaliado. Este trabalho utiliza a matriz do trabalho relacionado como uma forma visual de identificar as configurações dos produtos que serão recebidos como entrada pelo algoritmo.

Os trabalhos de Bécan *et al* (2015) e Al-Msie'Deen *et al*. (2014) tratam o mesmo conceito de extração dos modelos de *features* utilizando a engenharia reversa de um conjunto de configurações, porém para extrair os modelos os autores consideram o conhecimento sobre o domínio do usuário final para modelar a hierarquia do modelo.

Holanda Junior (2014) desenvolveu a ferramenta DyMMer que suporta a modelagem de *features* com alterações em sua configuração de acordo com a variabilidade de contextos pré-definidos. A ferramenta DyMMer, será utilizada para realizar a leitura dos modelos de *features* gerados pelo algoritmo e a partir desses modelos serão realizadas avaliações utilizando as medidas selecionadas do trabalho de Bezerra *et al*., (2014).

3 FUNDAMENTAÇÃO TEÓRICA

Na fundamentação teórica do trabalho, serão abordados os conceitos utilizados em seu desenvolvimento. Na Seção 3.1, será especificado o que é uma Linha de Produto de Software. Na Seção 3.2, serão definidos conceitos do modelo de *features*, como o modelo de *features* auxilia no gerenciamento de uma LPS, como funciona o processo de síntese dos modelos de *features* através da engenharia reversa e quais as ferramentas atualmente disponíveis para auxiliar nessa modelagem de *features*. Na Seção 3.3, serão definidas as abordagens para avaliar a qualidade de uma LPS, suas medidas de avaliação e como será realizada a seleção das melhores estruturas dos modelos de *features* baseado nessas medidas.

3.1 Linha de Produto de Software

Desde a década de 1990, a abordagem de LPS vem ganhando destaque tanto na área acadêmica quanto na área industrial (CAPILLA, BOSCH, 2011). A principal ideia de uma LPS é usar o conhecimento do domínio do negócio para separar as partes comuns de uma família de produtos. As partes comuns são usadas para criar uma base que poderá ser utilizada como uma linha de base comum para todos os produtos dentro de uma família de produtos. Uma LPS é uma abordagem com a finalidade de obter benefícios organizacionais, explorando semelhanças entre um conjunto de produtos que tratam de um segmento específico do mercado (ERIKSSON; HAGGLUNDS, 2003).

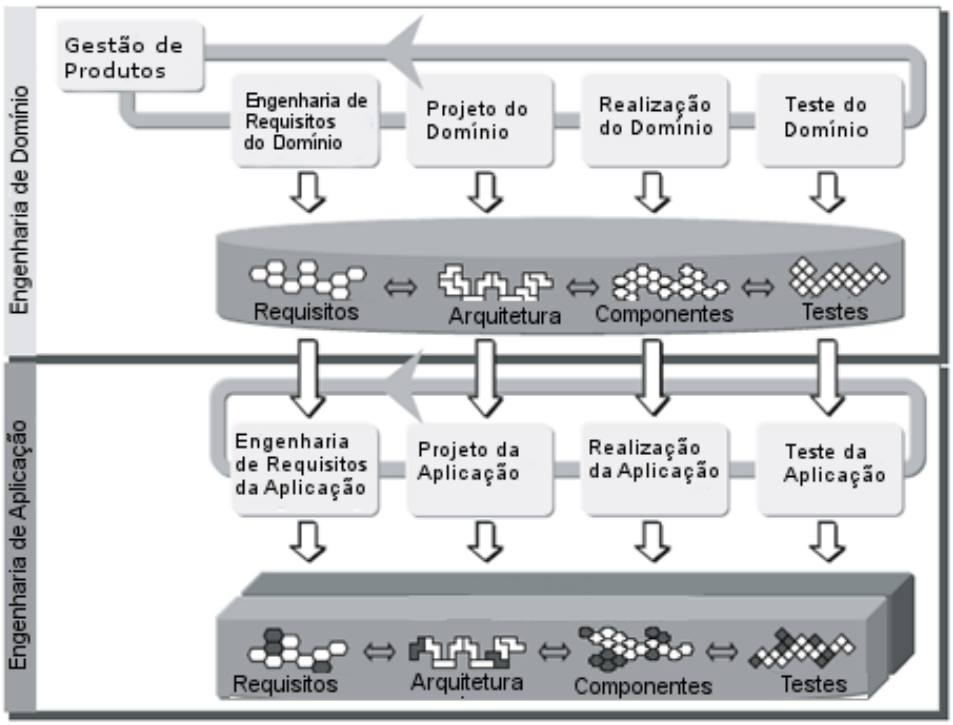
O paradigma de engenharia de uma LPS é dividido em Engenharia de Domínio e Engenharia de Aplicação. Na fase de Engenharia de Domínio, são definidas as partes comuns e variáveis de uma família de software. Já na parte de Engenharia de Aplicação, é construído um produto usando a reutilização de artefatos de domínio e verificando a variabilidade de um conjunto de produtos (BÖCKLE; POHL; LINDEN, 2005).

Atualmente existem alguns *framework*s que modelam o processo de uma LPS e destacam pontos importantes que ajudam em todo o ciclo de produção de um produto. Um exemplo de *framework* definido por Böckle, Pohl e Linden (2005, p. 21), apresenta um processo de engenharia de domínio que é composto por cinco subprocessos: gestão de produtos, requisitos da engenharia de domínio, design de domínio, desenvolvimento de domínio e teste de domínio. O processo de engenharia de aplicação é composto por quatro etapas que são: requisitos da aplicação, design da aplicação, desenvolvimento da aplicação e testes da aplicação.

Para cada etapa do processo é gerado um artefato específico, exceto na fase de gerenciamento de produto. Os artefatos gerados na fase de engenharia de domínio compõem a base da linha de produtos de software e são armazenados em um repositório comum. Os artefatos de aplicativo incluem todos os artefatos de desenvolvimento de um produto específico, incluindo o próprio aplicativo configurado e testado. (BÖCKLE; POHL; LINDEN, 2005). A visão geral do *framework* está ilustrada na Figura 1.

A vantagem da divisão em engenharia de domínio e aplicação é que há uma separação de propósitos, para construir uma base robusta e para construir aplicações específicas. Para serem eficazes, os dois processos devem interagir de uma maneira que ambos sejam beneficiados (PEREIRA *et al*., 2011).

Figura 1 - Framework da engenharia de uma Linha de Produto de Software.



Fonte: Böckle *et al*. (2005).

Normalmente o desenvolvimento de um modelo de *feature* se dá ao longo de toda a fase de Engenharia de Domínio, sendo o modelo criado a partir de um roteiro definido na etapa de Gerenciamento do Produto e refinado ao longo do processo de Engenharia de Domínio. A partir desse modelo é possível inter-relacionar a variabilidade que existe em diversos artefatos da Engenharia de Domínio, tais como, a variabilidade dos artefatos de requisitos, a variabilidade em artefatos de design, a variabilidade em componentes, e

variabilidade nos artefatos de teste. Apoiando a definição consistente de variabilidade em todos os artefatos de domínio. (BÖCKLE; POHL; LINDEN, 2005).

3.2 Modelo de *Features*

Modelos de *features* definem as combinações válidas de recursos em uma LPS com a finalidade de obter um produto consistente durante todo processo de engenharia de uma família de produto (DEHMOUCH, 2014).

O modelo de *features* é um tipo de modelo de variabilidade em que esse modelo de variabilidade é utilizado para representar a variabilidade de uma família de produtos (BERGER *et al*., 2013). Heymans e Trigaux (2003, p. 21) descrevem variabilidade como:

> A variabilidade é normalmente descrita com pontos de variação e variantes. Um ponto de variação localiza uma variabilidade e suas vinculações, descrevendo diversas variantes. Cada variante corresponde a uma alternativa projetada para realizar uma determinada variabilidade e, assim, vinculá-la de uma forma concreta. Em outras palavras, o ponto de variação localiza o ponto de inserção para as variantes e determina as características (atributos) da variabilidade.

Uma das principais vantagens dos modelos de variabilidade é o controle sobre as variabilidades existentes dentro das configurações do produto, das especificações de requisitos e dos produtos derivados (BERGER *et al*., 2013). Modelos de variabilidade realizam um relacionamento com o modelo base de uma linha de produtos, em que os elementos do modelo base podem se tornar elementos de um modelo específico, de acordo com a variabilidade do produto. Para expressar essa variabilidade de um domínio específico, foi proposto a *domain-specific language* (DSL) que consiste na captura da variabilidade entre os modelos em um determinado domínio ou de uma família de sistemas pelos construtores da linguagem, o que implica que todos os modelos possíveis nesta língua são as variações permitidas. Contudo, isso implica em uma dependência da linguagem de modelagem do modelo base (HAUGEN *et al*., 2008).

Com o intuito de contornar esse problema de dependência de linguagem, foi proposto a *Common Variability Language* (CVL) que é definida como uma linguagem genérica para modelagem de variabilidade em qualquer domínio que utiliza mecanismos para a definição de semântica que realizam análises para capturar os elementos no modelo. Um modelo CVL garante a independência de linguagem fazendo referências de sentido único para o modelo base, descrevendo como os elementos dos modelos de base podem variar. O modelo base é, portanto, independente das adições de variabilidade feitas pelo modelo CVL. A

execução de um modelo CVL trabalha sobre um domínio especifico ou uma DSL produzindo um modelo de transformação do modelo de base para um modelo de produto, onde a estrutura de alguns dos elementos do modelo base é modificada (SVENDSEN; HAUGEN; MØLLER-PEDERSEN, 2011).

O modelo de *features* é normalmente representado em *Unified Modeling Language* (UML) como um conjunto de características interligadas por um conjunto de relações que contêm dados que descrevem atributos dos recursos e são representadas de maneira hierárquica no formato de árvores. Um modelo de *features* pode ser exibido em difentes níveis de detalhamento, assim como em outros diagramas UML, pode-se adotar a melhor visão para se trabalhar com o modelo de *features* (GRISS; FAVARO; D'ALESSANDRO, 1998). Essas UMLs que representam o modelo de *feature* ajudam no gerenciamento de variabilidade de uma LPS sendo o método *Feature Oriented Domain Analysis* (FODA) a abordagem mais comum para a gestão de variabilidade (GHANAM, 2012).

A estrutura de um modelo de *feature* é definida através das relações entre as *features*. Benavides, Segura e Ruizcortés (2010), definem alguns tipos dessas relações para modelos básicos e relações baseadas em cardinalidade. Os tipos de relacionamento básicos são:

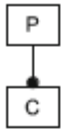
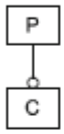
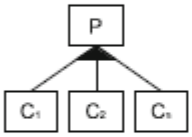
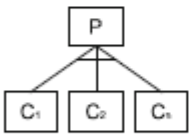
- Obrigatória – Inclusa em todos os produtos derivados da linha de produto;
- Opcional – Pode ou não estar inclusa em um produto;
- XOR – Apenas uma *feature* pode ser escolhida dentro de um grupo de *features*; e
- OR – Uma ou mais *features* podem ser escolhidas dentro de um grupo de *features*.

Os modelos baseados em cardinalidade definem as relações entre uma ou mais *features* através de um intervalo de números. Os tipos de relacionamento baseados na cardinalidade são:

- Cardinalidade de *feature* – A cardinalidade de *feature* é uma seqüência de intervalos indicados entre [n..m] com n como limite inferior e m como limite superior de instâncias. Essa relação pode ser usado para representar a obrigatoriedade ([1,1]) ou opção ([0,1]) das relações entre *features*; e
- Cardinalidade de grupo – A cardinalidade de grupo é um intervalo indicados entre [n..m] com n como limite inferior e m como limite superior de *features* filhas que podem ser instânciadas. Essa relação pode ser usado para representar grupos XOR (1, 1) ou grupos OR (1, n).

Existem algumas maneiras de representar uma estrutura de modelo de *features*, sendo elas através de elementos gráficos ou de fórmulas proposicionais. Segundo Benavides, Segura e Ruizcortés (2010, p. 625) "Uma fórmula proposicional consiste em um conjunto de símbolos primitivos de variáveis e um conjunto de conectivos lógicos que limitam os valores das variáveis". De maneira formal, um conjunto finito F de *features* pode ser representado como um conjunto de variáveis booleanas e seus conjuntos de dependência entre as *features* Φ pode ser considerada como uma fórmula proposicional sobre F (BATORY, 2005). Dessa maneira foi definido um mapeamento dos modelos de *features* para uma fórmula proposicional, com o intuito de validar modelos ou mesmo para manipular esses modelos através de ferramentas utilizando essas fórmulas. A Figura 2 apresenta este mapeamento a partir dos possíveis tipos de relações entre *features*.

Figura 2 – Mapeamento do modelo de features para logica proposicional

Relação		Mapeamento Formula Proposicional
Obrigatória		$P \leftrightarrow C$
Opcional		$C \rightarrow P$
OR		$P \leftrightarrow (C_1 \vee C_2 \vee ... \vee C_n)$
XOR		$(C_1 \leftrightarrow (\neg C_2 \wedge ... \wedge \neg C_n \wedge P)) \wedge$ $(C_2 \leftrightarrow (\neg C_1 \wedge ... \wedge \neg C_n \wedge P)) \wedge$ $(C_n \leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge ... \wedge \neg C_{n-1} \wedge P))$
Implicação		$A \rightarrow B$
Exclusão		$\neg(A \wedge B)$

Fonte: Benavides *et al*.(2010).

Estas fórmulas são geralmente utilizadas em algoritmos nos formatos FNC ou DNF, onde um CNF deve assumir a forma de uma conjunção de cláusulas com disjunções de literais e um DNF deve assumir a forma de uma disjunção de cláusulas com conjunções de literais. Neste trabalho o conjunto de produtos de uma SPL será representado em uma matriz de configuração onde seu conteúdo pode ser transformado em uma fórmula no formato CNF ou DNF. Com intuito de facilitar a visualização das relações entre as *features* de uma fórmula proposicional foi construído um gráfico de implicações proposto por Czarnecki, Wasowski (2007, p. 25) onde foi realizada uma exemplificação a partir de um modelo coletado da ferramenta SPLOT. Um exemplo de representação de uma fórmula em um grafo pode ser visto na Figura 9 na Seção 5.1.

A partir dessas fórmulas é possível construir diferentes tipos de modelos de *features* de maneira que se possam obter todos os modelos derivados a partir desta fórmula com estruturas distintas utilizando para isso propriedades das fórmulas tais como transitividade e equivalência (CZARNECKI; WASOWSKI (2007)). Com base nessa declaração, este trabalho realiza a extração de uma fórmula proposicional a partir de uma matriz que contém um conjunto de configurações de produtos. Após a extração da fórmula é realizado a modelagem de diferentes estruturas que essa fórmula pode tomar utilizado para isso um conjunto de cálculos matemáticas para limitar a quantidade de estruturas retornadas, mantendo apenas os modelos com raízes mais centrais com intuito de evitar modelos muito profundos, pois de acordo com o trabalho de Berger e Guo (2014), modelos com profundidades moderadas são os mais utilizados na prática. Para representar os modelos de *features* são utilizadas as notações propostas por Benavides, Segura e Ruizcortés (2010), para representar um modelo de *features*

3.2.1 Notações

Tendo em vista a importância do gerenciamento da variabilidade, foi definido no início da década de 1990 o método FODA. Esse método tem como finalidade o desenvolvimento de produtos de domínio genéricos e amplamente aplicáveis dentro de um domínio. A partir desse método foi possível desenvolver um processo com cinco etapas, em que cada etapa é constituída por uma entrada e tem como saída um produto (KANG *et al*., 1990). O Quadro 1 exibe um resumo do método FODA mostrando as fases da modelagem de *features*, assim como as entradas de cada processo, a nomenclatura de cada processo, os produtos gerados por estes processos e uma breve descrição dos produtos gerados.

Quadro 1 – Um resumo do método FODA

Fase	Entrada	Processo	Produto	Descrição
Análise de Contexto	Ambientes Operacionais Padrões	Análise de Contexto	Modelo do Contexto	O ambiente em que os aplicativos serão utilizados e operados
Modelagem de Domínio	*Features*, Modelo do Contexto	Análise das *Features*	Modelo de *Features*	Perspectiva do usuário final das capacidades do aplicativo em um domínio
	Conhecimento do domínio de aplicação	Modelagem entidade-relacionamento	Modelo Entidade-Relacionamento	Entendimento dos desenvolvedores sobre as entidades do domínio e seus relacionamentos
	Tecnologia do Domínio, Modelo do Contexto, Modelo de *Features*, Modelo Entidade-Relacionamento, Requisitos	Análise Funcional	Modelo de fluxo de dados	Perspectiva das funcionalidades da aplicação para Analistas de Requisitos
			Modelo de máquina de estados finitos	
Modelagem Arquitetônica	Tecnologia de Implementação, Modelo do Contexto, Modelo de *Features*, Entidade-Relacionamento Informação do Designer,	Modelagem Arquitetônica	Modelo de interação de processo	Perspectiva de alto nível da estrutura da aplicação para o Designer
			Gráfico estrutural do módulo	

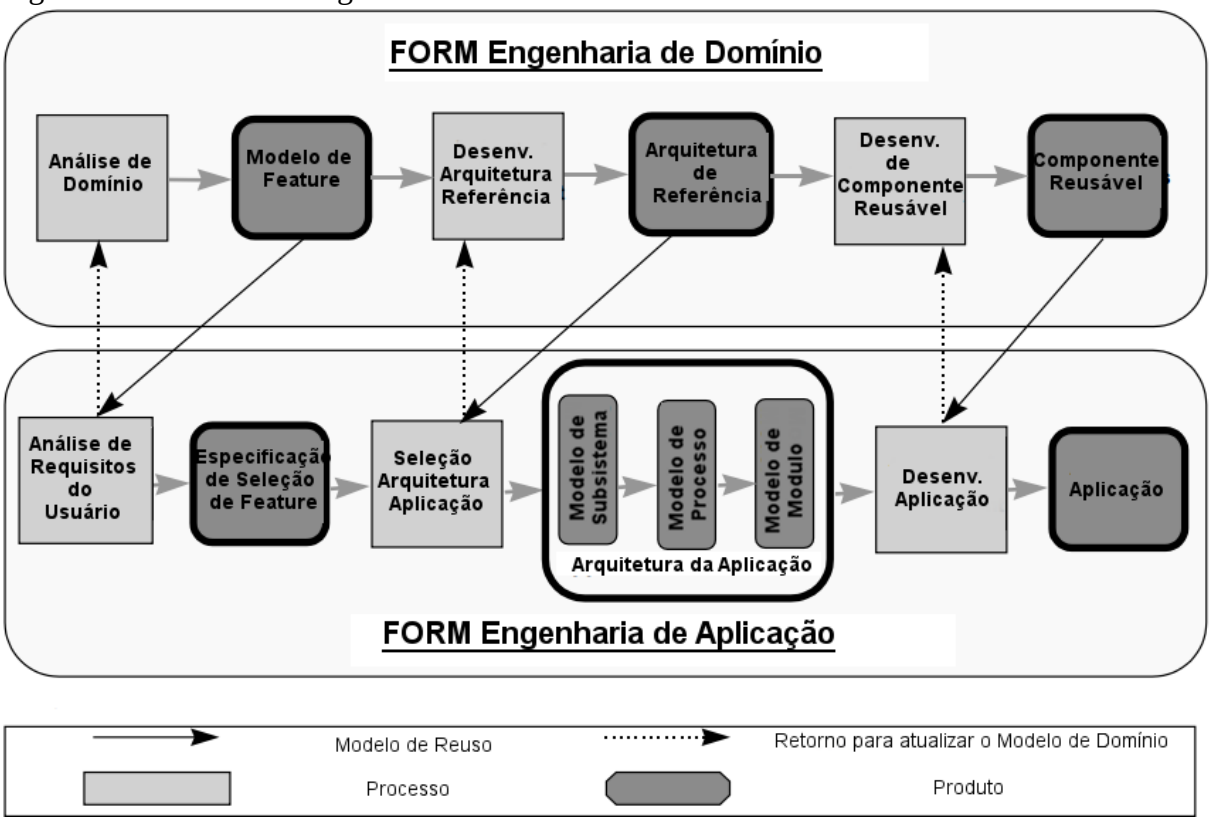
Fonte: Kang *et al*., (1990).

O método FORM é orientado a modelos de *features* e prevê o uso dessas *features* em suas fases de engenharia. O núcleo do método FORM se encontra na análise de *features* de domínio e uso dessas *features* para desenvolver artefatos de domínio reutilizáveis. Na Figura 3 é possível visualizar o Processo de Engenharia do método FORM separado nas fases de Engenharia de Domínio e Engenharia de Aplicação. Cada etapa da Engenharia de Domínio gera um artefato que é utilizado nas etapas de Engenharia de Aplicação que por sua vez, retornam sugestões de melhorais para a etapa geradora do artefato.

Uma abordagem mais recente denominada como modelos de *features* estendidos, propõe a adição de informações vinculadas às *features* em forma de atributo. Ainda não há um consenso de como se devem representar esses atributos, mas a maioria considera ao menos um nome, um domínio e um valor para cada atributo. Com a adição desses atributos, o modelo de *features* pode incluir complexas restrições entre as *features* e seus atributos (BENAVIDES; SEGURA; RUIZCORTÉS, 2010).

Todavia, este trabalho irá considerar apenas as relações do modelo básico para representação dos modelos de *features*, pois ainda não há uma forma de representar informações adicionais nas ferramentas selecionadas para este trabalho.

Figura 3 – Processo de Engenharia do método FORM



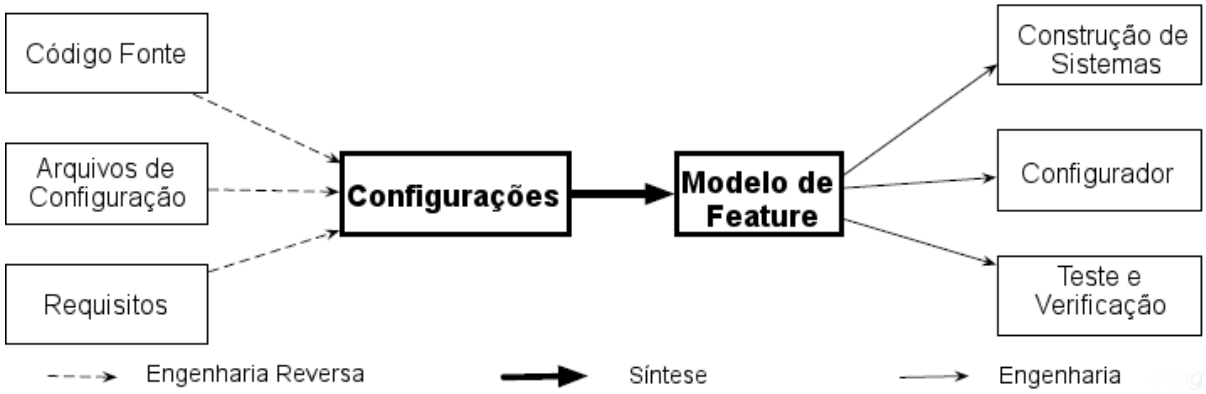
Fonte: Kang *et* al. (1998).

3.2.2 Síntese de Modelos de Features

Conforme foi mencionado, o modelo de *features* pode auxiliar no gerenciamento de uma LPS, porém uma construção manual de um modelo de *feature* é demorada e propensa a erros. Para esses tipos de problemas foi desenvolvida uma forma de elaboração de modelos de *features* de maneira automatizada utilizando engenharia reversa. Técnicas de engenharia reversa para modelos de variabilidade podem facilitar a adoção de práticas de linha de produto além de permitir uma migração de código legado para arquiteturas de famílias de produtos (SHE *et al*., 2014).

O processo de engenharia reversa é a parte principal da sintetização de um modelo de *features*. A construção dos diagramas é feita a partir de um conjunto de configurações e dependências que contêm uma hierarquia de recursos, que a partir disso, ele executa um algoritmo para excluir ou realizar implicações de restrições (SHE *et al*., 2014). O principal desafio da síntese de modelos de *features* é que muitos modelos podem ser extraídos das mesmas configurações de entrada, mas apenas alguns deles são significativos e de fácil manutenção (ARCHER *et al*., 2013). A Figura 4 ilustra como é utilizada a engenharia reversa para a construção de modelos de *features* sintetizados.

Figura 4 – Síntese de um Modelo de *Features* utilizando Engenharia Reversa



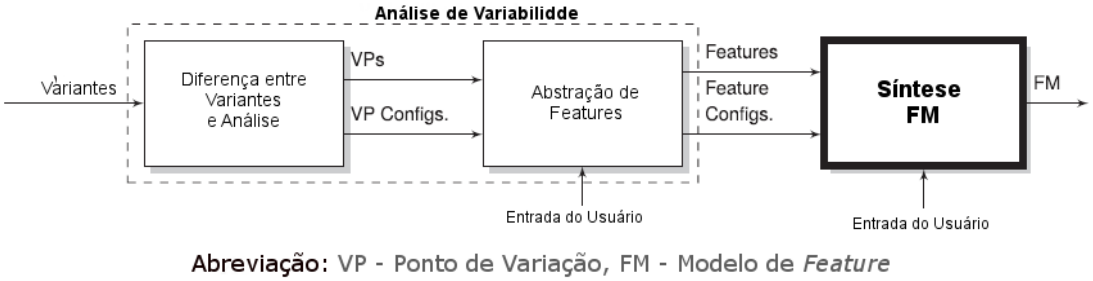
Fonte: Archer *et al*. (2013).

Conforme ilustrado na Figura 4, é possível realizar a síntese de um determinado modelo de *features* utilizando vários tipos de entradas, tais como, código, arquivos de configuração de um modelo de *features* ou requisitos. Tendo em vista essas entradas do processo, foram definidos alguns cenários envolvendo a síntese de modelos de *features*.

She et al., (2014) define 3 diferentes tipos de cenários que descrevem o *workflow* a partir dos artefatos de entrada e a forma como esses artefatos são transformados em entradas para o processo de sintetização. Os tipos de cenários são:

- Síntese de modelo de *features* a partir das configurações dos produtos – O primeiro cenário envolve a síntese de um modelo de *features* a partir de um conjunto de variantes que descrevem uma linha de produtos. A partir desse conjunto de variantes é derivado os pontos de variação que são usados para abstrair configurações de *features* são usadas como entrada na sintetização do modelo. A ilustração desse cenário pode ser vista na Figura 5;

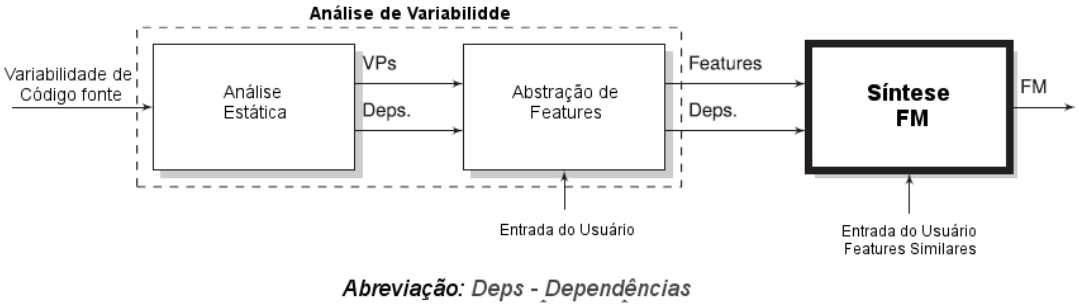
Figura 5 – Geração do modelo de *features* a partir das configurações dos produtos



Fonte: She *et al*. (2014).

- Síntese de modelo de *features* a partir do código com variabilidade – Este cenário descreve a engenharia reversa de um modelo de *features* a partir do código. As entradas para este cenário são módulos de código-fonte que contém variabilidade. Uma análise estática de código pode descobrir pontos de variação e dependências entre estes pontos que a partir de uma abstração, são transformados em *features* e suas dependências que são usadas como entrada na sintetização do modelo. A ilustração desse cenário pode ser vista na Figura 6; e

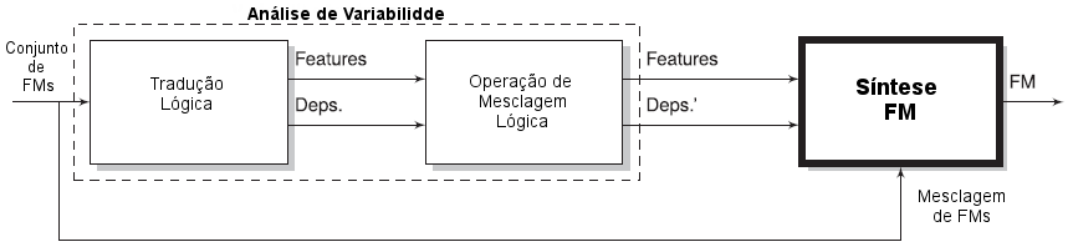
Figura 6 – Geração do modelo de *features* a partir do código fonte com variabilidade



Fonte: She *et al*. (2014).

- Modelo de *features* usando operações de mesclagem – O terceiro e último cenário descreve a operação de mesclagem que cria um novo modelo de *features* a partir da interseção ou união das descrições de configurações de dois ou mais modelos de *features*. Os modelos de *features* que são recebidos como entrada neste cenário são convertidos para suas fórmulas proposicionais, em seguida, uma operação de mesclagem proposicional é executada gerando um novo modelo de *features* com configuração semântica dependendo do tipo selecionado de mesclagem. A ilustração desse cenário pode ser vista na Figura 7.

Figura 7 – Geração do modelo de features a partir de operação mesclagem de modelos



Fonte: She *et al*. (2014).

Para este trabalho, será utilizada a geração dos modelos a partir das configurações dos produtos, pois este tipo de técnica nos permite a geração de múltiplas estruturas levando em consideração que não haverá nenhum tipo de entrada por parte do usuário tornando a geração dos modelos de *features* o mais automatizado possível.

3.2.3 Ferramentas

Segundo Kastner *et al*., (2009, p. 611) "Ferramenta de suporte, tais como ambientes de desenvolvimento integrado, é crucial para a aceitação e adoção de uma nova linguagem de programação ou paradigma, tanto no meio acadêmico e da indústria". Tendo essa necessidade em mente, foi realizada uma pesquisa por ferramentas que pudessem dar suporte a modelagem de *features*. As ferramentas encontradas foram mapeadas no Quadro 2 que fornece o nome, descrição e o tipo da ferramenta.

Quadro 2 – Ferramentas de suporte à modelagem de *features*

Nome	Descrição	Tipo Ferramenta	Referência
Gears SPL Lifecycle *Framework*	*Framework* que suporta todo o ciclo de desenvolvimento de uma linha de produtos	*Framework*	(KRUEGER, 2010)
FAMILIAR (*Feature* Model Script Language for Manipulation and Automatic Reasoning)	Linguagem de configuração de modelos de *features* que suporta tipos complexos e primitivos	Ferramenta Desktop e WEB	(ACHER, *et al*., 2013)
***Feature*IDE**	Ferramenta que dá suporte todas as fases de desenvolvimento orientado a *features* de software para o desenvolvimento de linhas de produtos	Plugin para o Eclipse	(THÜM, *et al*., 2014)
SPLOT	Fornece um ambiente para edição e configuração de modelos de *features*	Ferramenta WEB	(MENDONCA; BRANCO; COWAN, 2009)
FAMA (*Feature* Model Analyser)	*Framework* para a análise automatizada de modelos de *features* integrado com algumas das representações lógicas mais comuns	*Framework*	(BENAVIDES, *et al*., 2007)

Fonte: Elaborado pelo autor.

As principais ferramentas utilizadas para a elaboração desse trabalho foram: FAMILIAR, FeatudeIDE e SPLOT. A linguagem FAMILIAR foi utilizada para verificar se

fórmulas proposicionais de diferentes estruturas eram equivalentes, também foi realizado um estudo aprofundado sobre o operador *ksynthesis* com intuito de verificar se suas operações poderiam ajudar na geração de vários modelos de *features*. A *Feature*IDE foi utilizada para desenhar os modelos de *features* e verificar a consistência dos modelos gerados assim como verificar o número de configurações válidas permitidas por cada modelo. A ferramenta SPLOT foi usada como um repositório onde foi coletado um modelo para gerar exemplos, coletar algumas medidas e gerar arquivos XML para serem utilizados na ferramenta DyMMer.

3.3 Avaliação da Qualidade de LPS

A qualidade é um fator crucial no desenvolvimento de uma LPS. A grande importância da garantia de qualidade de uma LPS se deve à reutilização sistemática das linhas de produtos, pois caso ocorra alguma falha ou uma decisão inadequada do projeto, esse erro pode se propagar para vários produtos de uma família de software (MONTAGUD; ABRAHÃO, 2009). Assim como no desenvolvimento de produtos de software individuais, os defeitos devem ser descobertos o mais cedo possível, pois com a descoberta de um defeito no final do processo pode levar a custos muito elevados de correção. Na engenharia de uma linha de produto, isso significa dizer que os defeitos devem ser descobertos na fase de engenharia de domínio (LIU; DEHLINGER; LUTZ, 2007).

Um dos principais desafios enfrentados pela garantia de qualidade em engenharia de linha de produto é como considerar a variabilidade dessa linha. Esse problema decorre do fato de que um conjunto de subsistemas de software e interfaces que são reutilizáveis pode variar dentro de uma família de produto de software. Devido a esse problema, técnicas de garantia de qualidade de sistemas únicos não podem ser diretamente aplicadas a esses artefatos variáveis (LAUENROTH; METZGER; POHL, 2010).

Com intuito de contornar ou solucionar este problema de garantia da qualidade em uma LPS, algumas estratégias foram estudadas e apresentadas na literatura. Uma proposta feita por Lauenroth *et al*., (2010) consiste em três estratégias denominadas como *commonality strategy*, *sample strategy* e *comprehensive strategy*. A ideia da *commonality strategy* é garantir a qualidade apenas das partes comuns de todos os produtos de uma família de produtos de software. Já na *sample strategy*, a técnica de controle de qualidade é aplicada apenas nas amostras de produtos que são derivadas a partir da linha de produtos. E, por fim, a *comprehensive strategy* consiste na aplicação da técnica de controle de qualidade a todos os produtos possíveis da linha de produtos.

Uma das estratégias usadas para garantia da qualidade de uma LPS nesse tipo de artefato é a aplicação de medidas de qualidade. Existem diversas medidas relevantes para a avaliação da qualidade de um modelo de *features*, medidas essas que se referem a uma característica de qualidade e afetam um atributo de qualidade respectivo (Bezerra *et al*., 2014). Esses atributos de qualidade podem ser classificados em dois tipos: atributos de qualidade de uma LPS e atributos relevantes ao domínio. Atributos de qualidade de uma LPS são considerados atributos de desenvolvimento ou não observáveis através de execução. Os atributos relevantes ao domínio geralmente são operacionais ou observáveis através de execução (EXTEBERRIA; SAGRDUI; BELATEGI, 2008).

3.3.1 Medidas de Avaliação da Qualidade do Modelo de Features

Tendo em vista a necessidade do controle da qualidade nas fases iniciais de engenharia de uma LPS, Bezerra *et al*., (2014) propôs um catálogo de medidas que pode ser usado para apoiar a avaliação da qualidade do modelo de *features*. Dessa forma, foi realizado um levantamento por medidas já validadas no meio acadêmico ou industrial para serem aplicadas de forma automatizada em modelos de *features* gerados pelo processo de sintetização. As medidas encontradas foram mapeadas no Quadro 3 que fornece as características de qualidade, o atributo de qualidade e a medida.

Quadro 3 – Características de qualidade, atributos de qualidade e medidas para avaliação da qualidade do modelo de *features*

Características	Atributo de qualidade	Medida
Adequação funcional	Corretude funcional	Precision / Recall / F-measure
	Adequação funcional	Valor de importância da *feature*.
Manutenibilidade	Analisabilidade	Número de *features* folhas. Impacto da mudança.
	Instabilidade	Flexibilidade de mudança. Índice de manutenção de uma *feature.* Impacto quantitativo de *features* variáveis em atributos de qualidade.
	Complexidade cognitiva	Complexidade cognitiva de um modelo de *feature.*
	Extensibilidade	Extensibilidade de *Feature*.
	Flexibilidade	Flexibilidade de Configuração
	Modularidade	*Features* dependentes de ciclos únicos. *Features* dependentes de ciclos múltiplos.
	Reusabilidade	Comunalidade não funcional
	Complexidade estrutural	Complexidade ciclomática. Complexidade de configuração. Complexidade de restrição. Complexidade estrutural. Complexidade composta. Complexidade de variabilidade. Complexidade de variante. Restrições.

Características	Atributo de qualidade	Medida
		Profundidade da árvore. Número de *features*. Número de *features* topo.
	Variabilidade	*Features* de pontos de variação múltiplos. Número de *features* variáveis. Número de pontos variantes. Taxa de variabilidade. Número de configurações válidas. *Features* de pontos de variação rígidos. *Features* de pontos de variação únicos.
Usabilidade	Facilidade de Uso	-
Eficiência de desempenho	Precisão	-
	Utilização de recurso	-
	Escalabilidade	-
	Comportamento de tempo	Tempo de documentação. Tempo quando uma *feature* foi incluída no escopo do projeto.
Portabilidade	Adaptabilidade	Adaptabilidade estática de *feature.*
		Adaptabilidade dinâmica de *feature*
Confiabilidade	Disponibilidade	-
	Consistência	Taxa de consistência
Segurança	Autenticidade	-
	Integridade	-

Fonte: Bezerra *et al*. (2014).

Para este trabalho foi selecionado apenas um subconjunto de medidas apresentada na Seção 6. As analises realizadas sobre os modelos de *features* sintetizados foram baseadas na manutenibilidade do modelo. Os atributos de qualidade selecionados que apresentaram algum tipo de variação em suas medidas foram flexibilidade e a complexidade estrutural, onde na complexidade estrutural foi verificados a complexidade composta de um modelo e sua profundidade.

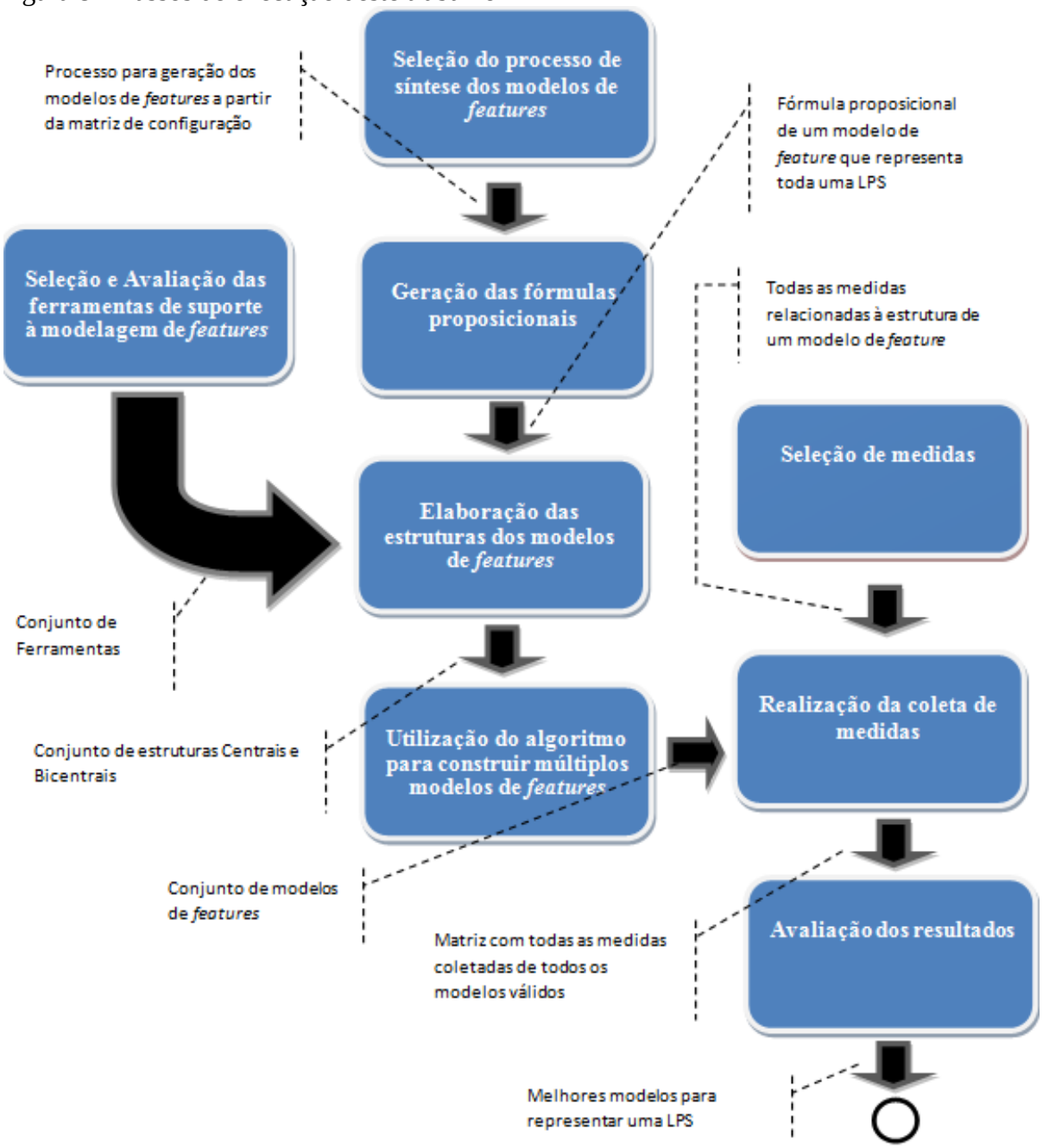
Algumas medidas não foram incorporadas neste trabalho devido a algumas delas necessitarem de mais informações do ciclo de vida de uma LPS, da evolução da construção do modelo de *features*, das decisões do cliente, das mudanças e do tempo, como por exemplo: *Precision*, *Recall*, *F-measure*, Valor de importância da *feature* (VFI), Complexidade de variabilidade (CVy), Complexidade de variante (CVt), Tempo de documentação (DT), Tempo quando uma *feature* foi incluída no escopo do projeto (TISP), Adaptabilidade estática de *feature* (FSA) and Adaptabilidade dinâmica de *feature* (FDA), Impacto da mudança (IC), Flexibilidade de mudança (FC), Índice de manutenção de uma *feature* (MI), Impacto quantitativo de *features* variáveis em atributos de qualidade (QIFVF), Complexidade estrutural (SC) e Taxa de consistência (CR) (BEZERRA *et al*., 2014).

4 PROCEDIMENTOS METODOLÓGICOS

Em Linhas de Produto de Software, a avaliação da qualidade é um fator crítico, porque um erro em uma LPS pode se propagar para vários produtos finais. No entanto, muitas vezes é impraticável assegurar a qualidade de todos os produtos de uma determinada LPS tanto por razões econômicas como por esforço necessários devido ao seu grande número (BEZERRA *et al*., 2014). Com o intuito de resolver esses problemas, foram desenvolvidas técnicas de síntese de modelos de *features* utilizando engenharia reversa para criar modelos de *features* de maneira automática, a fim de evitar erros inseridos pela criação e gerenciamento manual desses modelos e diminuição do esforço empregado para desenvolvê-los (SHE *et al*., 2014). Contudo, a síntese de modelos de *features* ainda apresenta problemas na geração desses modelos em que uma mesma entrada pode gerar vários modelos de *features* e muitos desses modelos gerados não são significativos para o contexto dos produtos de uma LPS (ARCHER *et al*., 2013).

Com base nas dificuldades enfrentadas pelas empresas e academia de avaliar a qualidade de um modelo de *feature*, sendo eles criados de maneira manual ou a partir de técnicas de sintetização, este trabalho desenvolve uma forma de automatização da construção dos modelos de *features*. Os passos metodológicos para a execução desse trabalho estão descritos de acordo com a Figura 8.

Figura 8 – Passos de execução deste trabalho



Fonte: Elaborada pelo autor.

4.1 Seleção e avaliação das ferramentas de suporte à modelagem de *features*

Nessa primeira etapa deste trabalho foi realizado um estudo aprofundado das ferramentas FAMILIAR, SPLOT, *Feature*IDE e DyMMer. Inicialmente foi realizado um estudo sobre a FAMILIAR onde foi identificada a possibilidade de utilizar sua funcionalidade de comparação de fórmulas proposicional sobre as fórmulas geradas pelo algoritmo desenvolvido neste trabalho com a finalidade de validar essas fórmulas e identificar se elas são equivalentes. Também foi feito um estudo sobre o operador *ksynthesis* utilizado pela

linguagem FAMILIAR para verificar se era possível a geração de múltiplos modelos a partir de uma fórmula. No entanto o operador utiliza conhecimento ontológico a partir de mineração de dados junto com heurísticas, onde são utilizadas as informações coletadas para determinar a hierarquia de um modelo, tendo como resultado apenas um modelo final.

Na ferramenta SPLOT foi verificada a possibilidade de utilização de alguns dos modelos presentes em sua base de dados para realizar exemplos, além de utilizar sua funcionalidade de exportação de modelos de *features* para o formato XML para ser utilizado como entrada na ferramenta DyMMer com intuito de realizar a coleta das medidas destes modelos.

A *Feature*IDE foi utilizada para desenhar os modelos de *features*, além de possibilitar a verificação da consistência do modelo.

Em seguida, foi feita a instalação da ferramenta DyMMer. A ferramenta DyMMer foi desenvolvida para coletar medidas de modelos de *features* em formato XML que devem seguir o formato dos modelos extraídos do repositório da ferramenta SPLOT. A DyMMer processa um arquivo de modelo de *features* e cria um modelo na memória em que a partir desse modelo é possível calcular os valores das medidas. Para entender melhor suas funcionalidades foi realizada uma leitura aprofundada de todo o seu projeto. Também foi feito uma leitura de todas as classes desenvolvidas, a fim de obter uma ampliação da visão do código e maior entendimento de como é realizado as operações no sistema.

Com a realização desses passos foi possível obter maior conhecimento sobre a finalidade das ferramentas e também maior conhecimento das possibilidades de seu uso.

4.2 Seleção do processo de síntese dos modelos de *features*

Nessa etapa foram pesquisados e analisados alguns cenários de síntese de modelos de *features* e a partir desses cenários analisados foi selecionado o processo que possibilitava a construção de múltiplos modelos a partir de uma dada entrada. Foram removidas do processo as interações que o usuário teria para determinar a hierarquia de um modelo, uma vez que a técnica utilizada para este trabalho irar gerar todos os modelos possíveis de acordo com a matriz de configuração recebida como entrada sem considerar a semântica do modelo, apenas a sintaxe que é representada pelos relacionamentos entre as *features*.

Também foram analisadas técnicas de refatoração bidirecional, generalização e especialização dos modelos para gerar diferentes modelos. Uma refatoração de um modelo de *features* pode ser vista como uma modificação da estrutura do modelo com intuito de melhorar sua qualidade em relação a sua manutenibilidade.

Foi então analalisada a possibilidade de gerar múltiplos modelos a partir da refatoração bidirecional devido à sua propriedade de manter as configurações do modelo apenas realizando a reorganização dos relacionamentos entre as *features* (ALVES *et al.*, (2006). Contudo, não foi identificada uma forma de gerar múltiplos modelos a partir de uma ferramenta e nem como aplicar algum tipo de limitador sobre o número de resultados. Devido a essa limitação, esta técnica não foi incorporada neste trabalho.

As refatorações de generalização e especialização não foram consideradas, pois a generalização ocasiona um aumento no número de configurações e a especialização ocasiona uma diminuição no número de configurações (GHEYI; MASSONI; BORBA, (2011)).

4.3 Geração das fórmulas proposicionais

Após selecionar o processo de como será feito a sintetização, foram estudados algoritmos identificados na literatura, para gerar fórmulas proposicionais a partir de um conjunto de configurações de produtos. A partir desses algoritmos foi extraído um conjunto de passos para serem aplicados em exemplos gerados a partir de modelos que se encontram na ferramenta SPLOT.

Para realizar a interpretação dos algoritmos, primeiramente foi realizado um estudo sobre mapeamento de modelos de *features* para fórmula proposicional. Logo após, foi feito um estudo sobre os formatos FNC e FND, devido à necessidade da fórmula de entrada do algoritmo estar em um destes dois formatos. Por fim, foi descrito o algoritmo em forma de passos para que se pudesse realizar a extração manual da fórmula para o exemplo apresentado nesse trabalho uma vez que os algoritmos propostos por She *et al*. (2014) e Czarnecki, Wasowski (2007) não estão disponíveis em uma implementação real.

4.4 Elaboração das estruturas dos modelos de *features*

Após a extração da fórmula proposicional foi verificado a quantidade de *features* de um determinado modelo e aplicado o cálculo proposto por Cayley (1881) para diferentes estruturas Centrais e Bicentrais. Uma árvore Central pode ser vista como uma estrutura que possui apenas um nó central. Da mesma forma, uma árvore é definida como Bicentral quando sua estrutura apresenta dois nos centrais. Para verificar se uma árvore é Central podem-se realizar repetidas operações de remoção de folhas até que se obtenha um único vértice. Em árvores Bicentrais são realizadas as mesmas operações, contudo deve ser obtido como resultado final dois vértices.

Estas estruturas foram aplicadas ao trabalho com intuito de limitar o número de estruturas geradas a partir de uma fórmula proposicional e assim realizar apenas a avaliação em um pequeno grupo de modelos com suas *features* organizadas de uma maneira mais balanceada.

4.5 Utilização do algoritmo para construir múltiplos modelos de *features*

Para realizar a geração de múltiplos modelos foi proposto um algoritmo de força bruta utilizando método *backtrack* em que sua entra é composto por um conjunto de estruturas e uma fórmula proposicional. O algoritmo proposto teve como base o algoritmo DPLL em que é utilizado em técnicas de solução de satisfatibilidade (SAT), cuja ideia principal é realizar a aprendizagem de operações que causam conflito. Logo após, realizar o *backtrack* um passo anterior é tentar resolver o problema por outro caminho até encontrar uma solução que satisfaça a fórmula dada como entrada ou até não haver outras opções de caminho.

O algoritmo proposto além de ter como base o DPLL ele também utiliza a técnica SAT para verificar se um modelo gerado satisfaz a fórmula proposicional recebida como entrada. Devido à natureza do problema e a necessidade de verificar se um modelo satisfaz a uma determinada fórmula é trivial deduzir que o problema esta do grupo dos NP-completo, pois os problemas de verificação de satisfatibilidade estão todas neste grupo o que pode tornar sua utilização impraticável para grandes quantidades de *features* sem a utilização de uma heurística (MENDONCA, WĄSOWSKI, CZARNECKI (2009); EÉN, SÖRENSSON (2004)).

4.6 Seleção de medidas

As medidas para avaliação da qualidade do modelo de *features* foram extraídas do catálogo de medidas propostas por Bezerra et. al. (2014). Foi realizada uma análise de quais medidas do catálogo se adequam aos modelos de *features* gerados pelo algoritmo proposto. A partir dessa análise foram selecionadas as medidas que possuíam algum tipo de relação com os critérios de avaliação com intuito de classificar as melhores estruturas dos modelos de *features*.

Nesta fase também foi verificado a existência de medidas que não apresentaram nenhum tipo de variação, pois estavam relacionadas aos tipos de *features* que é uma característica semântica e que não se altera apenas com a mudança da estrutura de um modelo. Devido a essa não alteração dos valores, todas as medidas com essas características foram excluídas da avaliação.

4.7 Realização da coleta de medidas

Todos os modelos gerados foram transformados em arquivos XML para que pudessem ser inseridos na ferramenta DyMMer e a partir de sua funcionalidade, foram coletadas dos modelos todas as medidas selecionadas do catalogo de Bezerra et. al. (2014) e enviadas para um arquivo no formato Excel.

Logo após, todas as medidas foram transferidas para uma única tabela e inseridas na ferramenta R para que se pudesse fazer um histograma para facilitar a análise das medidas e a visualização da frequência que uma determinada medida repetia seus valores entre todas as estruturas geradas.

4.8 Avaliação dos resultados

Para realizar a avaliação dos resultados das medidas coletadas foram utilizadas relações entre as medidas proposta por Bagheri e Gasevic (2011) onde os autores apresentam um conjunto de medidas em uma tabela que indica o índice de relação entre as medidas.

A partir dessa tabela foram verificadas as medidas que tinham um maior relacionamento e logo após foi aplicado esse índice sobre as medidas para verificar quais os modelos que se aproximavam dessa relação entre as medidas de acordo com o índice.

Também foram consideradas para a avaliação a complexidade e a flexibilidade, em que os melhores modelos deviam apresentar uma menor complexidade e uma maior flexibilidade, pois ambas as medidas impactam na manutenibilidade do modelo.

5 EXTRAÇÃO E SÍNTESE DE MODELOS DE *FEATURES*

Para extrair a melhor estrutura de um modelo de *features* este trabalho propõe três etapas. No primeiro passo é realizada a extração da fórmula proposicional a partir da matriz de configurações dos produtos, onde essa matriz é um mapeamento das *features* que constituem os produtos de uma SPL. Nessa mesma fase será feito a representação gráfica dessa fórmula a partir de um grafo de implicações, onde será possível visualizar as relações existentes entre cada *feasture*. A segunda etapa consiste na geração de todas as estruturas possíveis que podem ter uma árvore a partir do cálculo proposto por Cayley (1881) utilizando como entrada o número de variáveis ou *features* existentes na fórmula proposicional gerada. Na última etapa é feita a construção dos modelos de *features*, onde é realizada a distribuição das *features* em todas as estruturas geradas da segunda etapa de forma que os múltiplos modelos gerados a partir dessa operação mantenham as configurações iguais as da fórmula proposicional gerada no primeiro passo.

5.1 Extração do Grafo de Implicações

Como mencionado na Seção 5 o gráfico implicação é extraído a partir de uma matriz configuração, onde a principal função deste gráfico é a representação visual de uma fórmula proposicional para facilitar a identificação das relações entre as *features*.

Para realizar a extração do gráfico tendo como entrada uma matriz de configuração foi definido um conjunto de passos baseado nos trabalhos de Bécan *et al* (2015), Al-Msie'Deen *et al*. (2014), She *et al*. (2014), Haslinger, Lopez-Herrejon e Egyed (2011) e Czarnecki, Wasowski (2007).

Quadro 4 – Matriz de mapeamento entre *features* e produtos

Products	Features									
	E	L	P	T	C	S	H	ST	SE	PR
P1	X	X	X	X	X	X	X	-	X	-
P2	X	X	X	X	X	X	-	X	X	X
P3	X	X	X	X	-	X	X	-	X	X
P4	X	X	X	X	-	X	-	X	-	-
P5	X	X	X	-	X	X	X	-	-	-
P6	X	X	X	-	X	X	-	X	-	-
P7	X	X	X	X	-	X	X	-	-	-
P8	X	X	X	-	X	X	X	-	X	X

Fonte: Elaborada pelo autor.

Para cada etapa é necessária identificar os tipos de relações que existem entre as *features* tendo como base os produtos representados na matriz de configuração. A matriz de

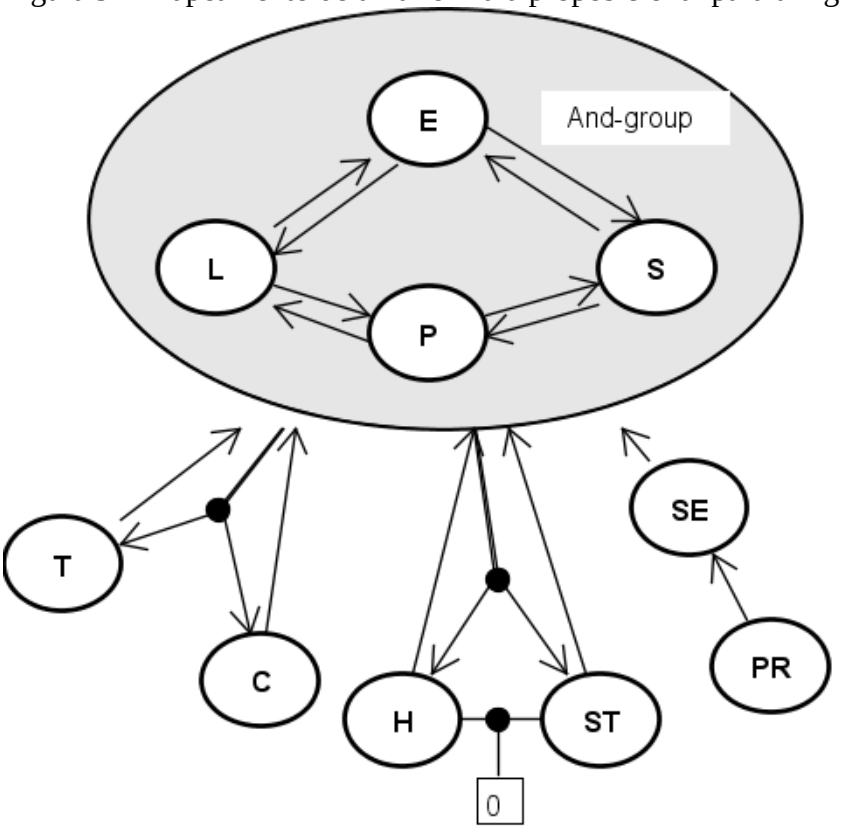
configuração é composta de linhas que representam os produtos de uma LPS e colunas que são as *features* desta LPS. Cada elemento desta matriz é uma variável que indica se uma *feature* pertence a um produto ou não. Para identificar os tipos de relações entre as *features* da matriz de configuração foram utilizadas as regras descritas abaixo:

- Conjunto atômico: As *features* f1 e f2 são parte de um conjunto atômico se para todos os produtos gerados, f1 e f2 sempre aparecem juntos. Formalmente isso quer dizer que tendo uma variável P que representa todos os produtos gerados de uma LPS, onde (f1 ∈ P) se (f2 ∈ P) e (¬f1 ∈ P) se (¬f2 ∈ P) onde ¬ representa a negação de uma *feature* o que significa que uma *feature* não está presente em um produto específico (HASLINGER, LOPEZ-HERREJON, EGYED (2011); CZARNECKI, WASOWSKI (2007); SHE *et al*. (2014));
- *Features* obrigatórias: Essas *features* não podem ser logicamente distinguidas de um conjunto atômicas, porque esses dois tipos são representados da mesma forma, com implicações de uma *feature* pai para uma *feature* filha e vice-versa (CZARNECKI; WASOWSKI, (2007));
- *Features* opcionais: Uma *feature* f é opcional se houver um parente g tal que f → g e não acontece g → f. Uma maneira de verificar se na matriz de configurações contém um relacionamento opcional é certificar-se de que ¬ f ⊆ ¬g (CZARNECKI, WASOWSKI (2007); SHE *et al*. (2014));
- Relações OR: Para identificar uma relação OR é necessário selecionar uma *feature* pai e compara-la com outras *features* da matriz e verificar se há pelo menos duas *feature* que tem uma relação direta com essa *feature* pai. Para identificar esse tipo de relação na matriz de configuração é necessário encontrar a relação *minimal*, onde é verificado fp = f1 ∪ f2 ∪ ... ∪ fn em que não é possível remover nenhum elemento sem alterar o resultado (BENAVIDES *et al.*, (2010); SHE *et al*. (2014));
- Relações XOR: Para identificar relações do tipo XOR é necessário primeiramente verificar as relações OR e posteriormente analisar as *features* OR que não aparecem juntos (CZARNECKI, WASOWSKI (2007); SHE *et al*. (2014)).

Para executar a extração da fórmula proposicional que será representada por um grafo de implicações foi definido um conjunto de passos com base no algoritmo apresentado na Figura 9 proposto nos trabalho (CZARNECKI, WASOWSKI (2007); SHE *et al*. (2014)). A entrada para esse algoritmo é dado por uma fórmula de variáveis binárias que representam

as configurações dos produtos em que o “0” representa a ausência de uma *feature* e o “1” representa a presença de uma *feature*. Essa fórmula binária foi mapeada para a matriz de configuração para facilitar a identificação visual das relações entre as *feastures*. Os passos extraídos desse algoritmo foram definidos logo abaixo e o resultado final destes passos de execução gera a fórmula proposicional representada pelo gráfico da Figura 10.

Figura 9 – Mapeamento de uma fórmula proposicional para um grafo



Fonte: Elaborada pelo autor.

- Passo 1 – Verificar *features* mortas: Uma *feature* morta é uma *feature* que está presente em um modelo, mas não é incluída em nenhum produto de uma LPS devido a erros de modelagem. Para identificar uma *feature* morta foi realizada uma busca e remoção das *features* que não aparecem em nenhum produto. A forma de representar uma *feature* morta na matriz de configuração é feita a partir de uma coluna totalmente em branco. (CZARNECKI, WASOWSKI (2007); SHE *et al*. (2014));
- Passo 2 - Computar os conjuntos atômicos: Neste passo é verificado se todas as variáveis de uma coluna i estão presentes em todos os produtos da matriz para extrair o conjunto atômico que estará relacionado com a *feature* raiz. Após esta operação é feita uma busca por *features* que sempre aparecem juntas nos produtos

para identificar os restantes conjuntos atómicas. As *features* que não têm um par são consideradas como conjuntos atômicos simples (HASLINGER, LOPEZ-HERREJON, EGYED (2011); CZARNECKI, WASOWSKI (2007); SHE *et al*. (2014));

- Passo 3 - Criar implicações entre *features* dos grupos atômicos: Criar dupla implicação entre todas as *features* dos conjuntos atômicos que contêm duas ou mais *features* para indicar uma forte relação entre elas, o que significa que deve haver uma relação obrigatória entre elas. O *And-group* da Figura 10 representa este relacionamento (HASLINGER, LOPEZ-HERREJON, EGYED (2011); CZARNECKI, WASOWSKI (2007); SHE *et al*. (2014));
- Passo 4 - Criar implicações entre conjuntos atômicos: Nesta etapa, será verificado para cada par de conjuntos atômicos se um conjunto M não está presente em um produto Pi então o conjunto N também não deve estar presente. Em uma maneira mais formal esta declaração pode ser representada como $N \subseteq \neg M$ (SHE *et al*. (2014); BÉCAN *et al*., (2015));
- Passo 5 – Verificar relações OR: É verificado na matriz de configuração se ela contém um grupo de *features* em que $fp = f1 \cup f2 \cup ... \cup fn$, onde fp é uma *feature* pai de um grupo atômico e fn é sua filha (SHE *et al*. (2014); BÉCAN *et al*., (2015));
- Passo 6 – Verificar relações XOR: Depois de verificar as relações OR é analisada qual dessas relações têm a propriedade $\forall i = 1... n$ e $\forall j = 1... n$. Onde $i \neq j$. $\varphi \rightarrow \neg(fi \wedge fj)$. Onde φ é a fórmula binária que contém as *features* que são representadas pela matriz de configuração. Os conjuntos OR que tiverem essa propriedade são convertidos para XOR (CZARNECKI, WASOWSKI (2007); BÉCAN *et al*., (2015)); e
- Passo 7 - Verificar implicações redundantes: É verificada a redundância de implicações através da propriedade de transitividade onde, $(a \rightarrow b)$ e $(b \rightarrow c)$ e $(a \rightarrow c)$, então a implicação de **a** para **c** deve ser removida devido a sua transitividade a partir da clausula $(b \rightarrow c)$ (CZARNECKI, WASOWSKI (2007)).

Para a definição desses passos foi considerado que o conjunto de configurações passado como entrada contém todas as *features* da linha de produtos e que todos os produtos são válidos, pois produtos inválidos podem apresentar contradições em sua modelagem e assim proporcionar a geração de um modelo de *features* insatisfatório. Um modelo é

insatisfatório nos casos em que não se pode derivar nenhum sistema a partir dele (MENDONÇA, 2009).

Figura 10 – Algoritmo tomado como base para extração da fórmula

FEATURE-GRAPH(φ : formula)

1 **if** not SAT(φ) **then** quit with an error.

2 ▷ Find and remove all dead features
3 $D = \{d_1, \ldots, d_m\} \leftarrow$ all $f_i \in F$ such that $\varphi \rightarrow \bar{f}_i$
4 $\varphi \leftarrow \exists d_1, \ldots, d_m . \varphi$

5 ▷ Compute the implication graph $G(E, V)$
6 $V \leftarrow F - D$
7 $E \leftarrow \{(u, v) \in V \times V \mid \varphi \wedge u \rightarrow v\}$

8 ▷ Compute strongly connected components of G
9 $C \leftarrow$ a set of SCCs in G

10 ▷ Contract SCCs in G creating and-groups
11 **do for** $c \in C$ **do** let f be a fresh node
12 $V \leftarrow V \cup \{f\} - c$
13 $E \leftarrow \{(u, v) \in E \mid u \notin c \wedge v \notin c\} \cup$
14 $\cup \{(u, f) \mid \exists v.(u, v) \in E\} \cup$
15 $\cup \{(f, v) \mid \exists u.(u, v) \in E\}$

16 ▷ G is acyclic (a DAG) at this point.
17 ▷ Compute OR-groups and XOR-groups
18 **for each** $f \in V$
19 **do for each** prime $\bar{f}_1 \wedge ... \wedge \bar{f}_k$ of $\varphi \rightarrow \bar{f}$
20 **do if** SAT($\bar{f}_1 \wedge \cdots \wedge \bar{f}_k \wedge \varphi$)
21 **then** Let f be a fresh node
22 $V \leftarrow V \cup \{f\}$
23 $E \leftarrow E \cup \{(f_1, f)\}$
24 $\cup \{(f, f_i) \mid i = 1...k\}$
25 **if** $f_2, \ldots, f_l$ satisfy (13)
26 **then** mark f as an XOR node

27 Compute the unique transitive reduction of G.

Fonte: Czarnecki, Wasowski (2007).

Após a definição da extração da fórmula proposicional foi iniciado a construção das estruturas e atribuição das *features* a essas estruturas para formar diferentes tipos de

modelos de *features*. O algoritmo que realiza a atribuição das *features* em diferentes tipos de estruturas pode ser visto na Figura 12 e suas etapas são descritas na Seção 5.2.

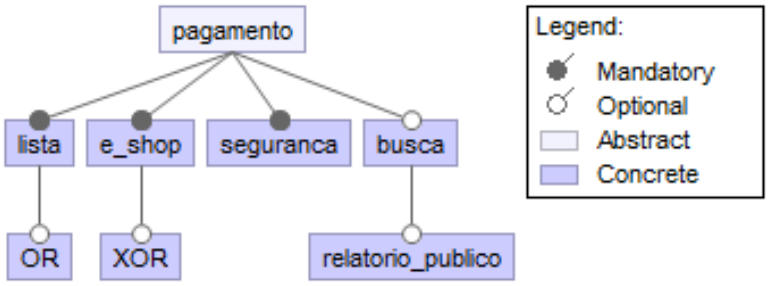
5.2 Construção das estruturas dos modelos de *features*

A fim de gerar uma gama de modelos que estão em conformidade com a fórmula proposicional extraída, foram realizados cálculos matemáticos em cima do número de *features* do modelo e assim foi possível gerar todas as estruturas, levando em consideração apenas às estruturas com os nós raiz mais centrais. A partir disso, as estruturas tendem a ter uma distribuição mais homogênea dos seus vértices e consequentemente uma diminuição de sua profundidade, que na prática, são os modelos mais utilizados segundo Berger e Guo (2014).

Os cálculos realizados sobre o número de *features* foram baseados no trabalho de Cayley (1881), onde foram contabilizados os números de árvores Centrais e Bicentral gerados de acordo com o número de *features* recebido como entrada.

Para o exemplo gerado neste trabalho foram consideradas as relações OR e XOR como apenas um nó cada uma, pois as *features* desses grupos devem ser sempre apresentadas juntas na representação de um modelo, obtendo-se assim um modelo exemplo com oito vértices que foi ilustrado na Figura 11.

Figura 11 – Modelo de *features* gerado a partir da matriz de configuração



Fonte: elaborada pelo autor.

De acordo com os trabalhos de Cayley (1859), (1875), (1881) tendo como Φn o número de árvores com N nós. Os sucessivos números Φ1, Φ2, Φ3, ... , Φn, é dado pela fórmula:

$$\phi n = \frac{1 \,.\, 3 \,.\, 5 \ldots \; 2n-3}{n!} \; 2^{n-1}$$

Logo após é verificado o número de estruturas repetidas e então são removidas de Φn tendo como resultado o Quadro 5 com o número de estruturas gerais.

Quadro 5 – Sufixo dos valores de Φ

ϕ	1	2	3	4	5	6	7	8
Valor	1	1	2	4	9	20	48	115

Fonte: Cayley (1881).

Logo após de se obter o número total de estruturas únicas, também chamadas de árvores raiz, a partir do cálculo de Φn é realizado um cálculo relacionado aos nós centrais de uma árvore em que são retirados às estruturas Centrais e Bicentrais utilizando para isso o seguinte cálculo:

$$\psi_1 = 1,$$
$$\psi_2 = \frac{1}{2}\,\phi_1\,(\phi_1 + 1),$$
$$\psi_3 = \textit{coeficiente de } x^2 \textit{ por } (1-x)^{-\phi_1},$$
$$\psi_4 = \frac{1}{2}\,\phi_2\,(\phi_2 + 1) + \text{ coeficiente de } x^3 \textit{ por } (1-x)^{-\phi_1},$$
$$\psi_5 = \textit{coeficiente de } x^4 \textit{ por } (1-x)^{-\phi_1}\,(1-x^2)^{-\phi_2},$$
$$\psi_6 = \frac{1}{2}\,\phi_3\,(\phi_3 + 1) + \textit{coeficiente de } x^5 \textit{ por } (1-x)^{-\phi_1}\,(1-x^2)^{-\phi_2},$$
$$\psi_7 = \textit{coeficiente de } x^6 \textit{ por } (1-x)^{-\phi_1}\,(1-x^2)^{-\phi_2}(1-x^3)^{-\phi_3},$$
$$\psi_8 = \frac{1}{2}\,\phi_4\,(\phi_4 + 1) + \textit{coeficiente de } x^7 \textit{ por } (1-x)^{-\phi_1}\,(1-x^2)^{-\phi_2}(1-x^3)^{-\phi_3},$$

O cálculo para identificação das estruturas Centrais e Bicentrais pode ser interpretado da seguinte forma.

O primeiro termo do cálculo para estruturas com quantidade de vértices pares representado por $\psi_n = \frac{1}{\Phi!}\,\phi_{\frac{n}{2}}(\phi_{\frac{n}{2}} + 1)$ retorna o número de árvores Bicentrais para uma quantidade de vértices igual a n. O valor $\frac{1}{\Phi!}$ representa o número de sub-árvores utilizadas para gerar as estruturas Bicentrais. O valor $\phi_{\frac{n}{2}}$ representa quais árvores devem ser combinadas para gerar as estruturas Bicentrais.

O segundo termo do cálculo utilizando tanto para estruturas com quantidade de vértices pares como para estruturas ímpares é representado por:

$\psi_n = \textit{coeficiente de } x^{n-1} \textit{ por } (1-x)^{-\phi_1} \ldots (1-x^{\frac{n}{2}-1})^{-\phi_{\frac{n}{2}-1}}$, que retorna as combinações das árvores restantes. De acordo com Cayley (1881) o termo $(1-x)^{-\phi_1} \ldots (1-x^{\frac{n}{2}-1})^{-\phi_{\frac{n}{2}-1}}$ é equivalente a $\phi_1 + x\phi_2 + x^2\phi_3 \ldots x^{n-1}\phi_n$ onde **x** representa o número de sub-árvores necessárias para serem combinadas e gerarem uma nova estrutura Central ou Bicentral.

O Quadro 6 ilustra a quantidade de estruturas Centrais e Bicentrais que podem ser extraídas de acordo com o número de *features* passada como entrada para o cálculo.

Quadro 6 – Quantidade de estruturas geradas de acordo com o número de *features*

Número de Features	1	2	3	4	5	6	7	8
Estruturas Centrais	1	0	1	1	2	3	7	12
Estruturas Bicentrais	0	1	0	1	1	3	4	11
Total	1	1	1	2	3	6	11	23

Fonte: Cayley (1881).

5.3 Construção dos modelos de *features*

Após a geração das estruturas Centrais e Bicentrais dos modelos, foi elaborado um algoritmo que utiliza técnicas de força bruta em conjunto com funções *backtrack* para atribuir as *features* a cada um dos vértices das estruturas geradas. Foi utilizada essa técnica para abordar todas as possibilidades válidas de modelos para cada uma das estruturas.

A entrada para o algoritmo é composta por uma fórmula proposicional gerada a partir do algoritmo apresentado na Figura 11 e as estruturas geradas na Seção 5.2 a partir dos cálculos de Cayley (1881). A fórmula deve ser ordenada antes de ser passada para o algoritmo proposto por este trabalho de modo que suas clausulas devam estar seguindo a ordem em largura dos relacionamentos entre as *features*. Esta ordem segue o padrão em que a primeira clausula deve representar a grupo atômico relacionado à raiz e as seguintes cláusulas devem ser as *features* relacianas a esse grupo, em seguida deve buscar os filhos das *features* relacionadas ao grupo atômico e assim por diante. Essa ordenação é importante para facilitar a atribuição das *features* as diferentes estruturas construídas. O algoritmo proposto está apresentado logo abaixo:

Construir_FMs (φ': fórmula ordenada, grafo: estrutura);

```
1   *Se não tiver mais clausulas para inserir na estrutura então a estrutura formada é válida
2    if (φ'.retorne Features Base() = nulo);
3          retorne: verdadeiro;

4   *Inserir as features do primeiro grupo atômico
5    estrutura.inserirFeatures(φ'.retorneFeaturesBase());

6   *Verificar se a sub-estrutura é válida após a inserção das features
7    if ( SAT(estrutura.retoneFórmula())) é válida then

8   *Chamar função recursivamente para ordenar os próximos grupos
9          Construir_FMs((φ'- φ'.retorneFeaturesBase()), estrutura)
```

```
10    *Verificar todas as possibilidades de reordenação das features
11     else  if (estrutura. temProximaOrdenação()) then
12           while (estrutura.temProximaOrdenação() )
13                 estrutura.reordenarFeatures(φ'.retorneFeaturesBase());

14           if ( SAT(estrutura.retoneFórmula())) é válida then
15                 Construir_FMs((φ'- φ'.retorneFeaturesBase()), estrutura)

16    *Tentar uma reordenação dos grupos anteriores
17     else
18           chamarBacktrack();

19    *Caso o backtrack retorne até a raiz então a estrutura não é válida
20           if (backtrackRaiz()) then
21                 if(realizaInserçãoConstraint())
22                       estrutura.inserirConstraint();
23                       retorne: verdadeiro;
24                 else
25                       retorne: falso;
```

Após receber as entradas, o algoritmo inicia realizando uma análise de condição de parada da recursão nas linhas 1 - 2 onde é verificado se existem alguma cláusula que deve ser inserida na estrutura em caso nulo o algoritmo encerra sua operação. Na linha 5 é inserido o primeiro grupo atômico que deve conter a *feature* root, onde a ordem de inserção pode ser realizada em largura e as *features* devem ser vizinhas para manter a relação da fórmula. As linhas 7 – 9 representam a verificação da inserção das *features*, onde é verificado se a inserção realizada corresponde à fórmula do grupo passado como parâmetro. Em caso afirmativo é realizado uma nova chamada da função através da recursão. Em caso negativo como descrito nas linhas de 11 – 15 é verificado uma nova forma de ordenação das *features* para satisfazer a fórmula do grupo até encontrar uma forma válida ou até esgotarem todas as alternativas. Caso não seja encontrada nenhuma alternativa é realizado o *backtrack* que é chamado na linha 18, onde é retornado um passo anterior e realizado uma nova ordenação das *features* uma camada acima da pilha de execução. Caso o *backtrack* retorne até sua raiz, então significa que não existe uma forma válida de aplicar as *features* na estrutura passada como parâmetro, então é verificado a possibilidade de inserir uma *constraint* de implicação ou exclusão na linha 21. Caso não seja possível à inserção de *constraint* é retornado que a estrutura é inválida.

A validação dos resultados do algoritmo foi realizada através da ferramenta FAMILIAR onde foram inseridos todos os modelos na ferramenta e a partir de sua funcionalidade *comp* foram verificados se todos os modelos eram equivalentes. A partir dessa verificação foi constatado que apenas 3 modelos não apresentavam fórmulas proposicionais equivalentes que foram os modelos A.12, B.13 e B.14. Essa não equivalência já era prevista uma vez que o algoritmo retornou que essas estruturas eram inválidas, pois não havia uma forma de organizar as features nas estruturas de maneira equivalente à fórmula proposicional extraída. Devido essa não conformidade com a fórmula proposicional, os modelos foram excluídos da avaliação.

6 AVALIAÇÃO DOS MODELOS DE *FEATURES*

Para realizar a avaliação dos modelos de *features* foi utilizado um conjunto de 21 medidas definidas no catálogo proposto por Bezerra et al., (2014) para avaliar a qualidade dos modelos a partir da característica de manutenibilidade. Logo após foi utilizado a ferramenta DyMMer para realizar a coleta das 21 medidas descritas no Quadro 7 para cada um dos modelos de *features* construídos.

Quadro 7 – Conjunto de medidas selecionadas para avaliar a qualidade do modelo de *features*

Medida	Descrição	Fórmula
Número de *features* (NF)	Soma de todas as *features* do modelo.	NF = Σ (Número de *features* do modelo de *features*)
Número de *features* folha (NLeaf)	O número de *features* com nenhuma *feature* filha ou ponto de variação	NLeaf = Σ (Número de *features* sem filhos do modelo de *features*).
Complexidade cognitiva (CogC)	Facilidade de comprenção de um modelo de *features* de acordo com sua variabilidade	CogC = Σ (Número de pontos variantes)
Flexibilidade de Configuração (FoC)	A taxa de possiveis configurações a partir do número opcional de *features*.	FoC = NO/NF NO - Número de *Features* Opcionais NF - Número de *Features*
***Features* dependentes de ciclos únicos (SCDF)**	Soma de todas as *features* que estão presentes nas restrições do modelo e são filhas de pontos de variação com cardinalidade [1..1].	SCDF = Σ (Número de *features* participantes em Constraints e filhas de pontos variantes com cardinalidade [1..1])
***Features* dependentes de ciclos múltiplos (MCDF)**	Soma de toda as *features* que estão presentes nas restrições do modelo e são filhas de pontos de variação com cardinalidade [1..*].	MCDF = Σ (Número de *features* participantes em restrições e filhas de pontos variantes com cardinalidade [1..*])
Extensibilidade de *Feature* (FEX)	Refere-se ao número de *features* que podem ser facilmente adicionadas ao modelo durante a fase de manutenção.	FEX = NLeaf + MCDF + SCDF
Complexidade ciclomatica (CyC)	Soma das restrições de integridade de um modelo de *features* que formam um ciclo de dependencia entre as *features*.	CyC = Σ (número de restrições de integridade)
Complexidade composta (ComC)	Representa o número de pontos variantes, o número de *features* variáveis, número de restrições de integridade e seus relacionamentos.	ComC = NF^2 + ($Rand^2$ + 2Ror^2 + 3$Rcase^2$ + 3Rgr^2 + 3R^2)/9 Rand = Número de relações mandatórias Ror = Número de relações de agrupamento OR Rcase = Número de relações de agrupamento XOR Rgr = Número de relações de agrupamentos R = Σ(Número de Relações de agrupamento) + Σ (Número de restrições)
Cross-tree Constraints (CTC)	Grau de envolvimento das *features* na definição das restrições de integridade.	CTC = NFRI / NF NFRI - número de *features* únicas envolvidas na restrição de integridade do modelo de *features*

Medida	Descrição	Fórmula
Coeficiente de densidade de conectividade (CoC)	Taxa de arestas entre as *features* do modelo.	CoC = NE / NF NE - Número de arestas
Número de *features* top (NTop)	Soma de todas as *features* filhas do nó raiz.	NTop = Σ (Número de descendentes da raiz)
Profundidade da árvore (DT)	O comprimento do caminho mais longo a partir da raiz do modelo de *features* até a *feature* folha do modelo de *features*.	DT = Σ (Número de *features* do maior caminho a partir da raiz do modelo de *features* até a *feature* folha do modelo de *features*)
Comunalidade de *features* não funcionais (NFC)	Representa as *features* que estão presentes em todas as configurações geradas pelo modelo.	NFC = NCNF / NF NCNF = Número total de *features* não funcionais comuns (obrigatórias).
***Feature* de ponto variante múltipla (MHoF)**	Soma de todas as *features* filhas de pontos de variação com cardinalidade [1..*])	MHoF = Σ (Número de *features* filhas de pontos variação com cardinalidade [1..*])
Número de configurações válidas (NVC)	O número de todas as configurações possíveis e válidas que podem ser derivadas a partir do modelo de *feature* de acordo com as restrições de integridade e estrutura da árvore.	NVC = Σ (Número de configurações possíveis e válidas do modelo de *feature*)
Número de *features* variáveis (NVF)	O número de *features* presente no modelo que estão relacionadas através de *features* de ponto variante.	NVF = NA + NO/NF NA = Número de *features* alternativas
Taxa de variabilidade (RoV)	O fator médio de ramificação que a *feature* pai apresenta no modelo de *features*. Em outras palavras, o número médio de filhos dos nós na árvore do modelo de *features*.	RoV = Σ (Número médio de filhos dos nós no modelo de *features*)
***Features* não ponto variantes rígidas (RNoF)**	O Número de *features* que podem ser adicionadas em tempo de execução, mas que não seja através de *features* de ponto variante.modelo de *features*. Essa medida representa o grau de envolvimento das *features* na definição das restrições de integridade.	RNoF = Σ (Número de *features* não filhas de pontos variantes)
***Features* de ponto variante única (SHoF)**	Representa o número de *features* que podem ser adicionadas em tempo de execução através de pontos de variação com cardinalidade [1..1]	SHoF = Σ (Número de *features* filhas de pontos variantes com cardinalidade [1..1])

Fonte: Bezerra *et al*., (2014).

A partir do exemplo que tinha um total de 10 *features* foi realizada uma redução de suas relações OR e XOR como mencionado na Seção 5.2, chegando a um total de oito *features*. Com esse modelo exemplo foi possível gerar um total de 34 modelos com estruturas distintas uma vez que para o resultado das estruturas Bicentrais o número de estruturas foram multiplicadas por 2 para que se pudesse obter estruturas com o nó raiz para os dois centros de uma estrutura Bicentral. Contudo, isso ocasionou a inserção de algumas estruturas repetidas

que foram eliminadas na fase de análise por possuírem medidas idênticas a seus pares. No final foram encontradas 3 pares de estruturas iguais que foram Bicentral 4, Bicentral 5 e Bicentral 11 onde foi removido uma estrutura de cada par.

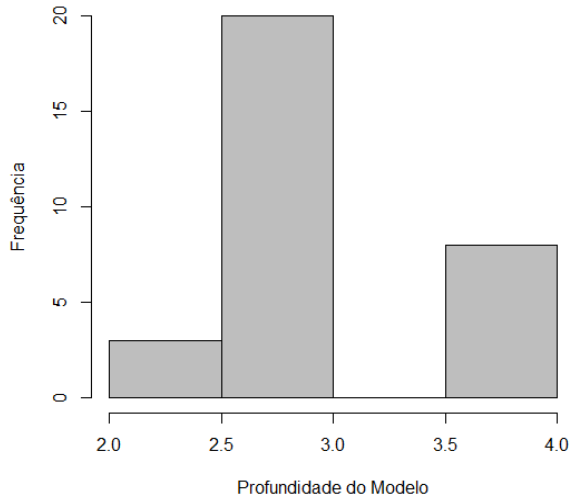
Com a execução dos passos do algoritmo foi possível verificar que a estrutura Central 12 e as estruturas Bicentral 7 e Bicentral 7.1 não permitem organizar as *features* de forma que seus modelos mantenham uma fórmula equivalente a extraída a partir da matriz de configuração, ocasionando uma alteração em seu número de configurações e, devido a isso, essas estruturas não foram consideradas na avaliação. Após as remoções restaram apenas um total de 28 estruturas que foram transformadas em formato XML para que pudessem ser importadas para a ferramenta DyMMer para realizar a extração das medidas de acordo com o Quadro 6.

Com o conjunto de medidas em mãos, foi possível verificar que algumas medidas estavam associadas ao tipo de relação entre as *features* o que resultou em nenhuma alteração dessas medidas em relação às diferentes estruturas. As medidas identificadas que não possuíram nenhum tipo de alteração foram NF, NA, NO, CogC, SCDF, MCDF, CoC, SHoF, MHoF, RNoF, NVC, NVF, RoV. Devido a essa não alteração das medidas elas foram desconsideradas da análise. Após a remoção das medidas, foram identificados quatro critérios de filtragem dos modelos candidatos que são: profundidade, complexidade e flexibilidade dos modelos de *features*.

6.1 Avaliação pela profundidade do modelo de *features*

O modelo de *features* é representado como uma estrutura de árvore em que seus vértices são um conjunto hierarquizado de *features*. Com intuito de analisar este aspecto hierárquico foi utilizado à medida Profundidade da Árvore (DT) que está representada com sua descrição e cálculo no Quadro 7. De acordo com o trabalho de Berger e Guo (2014), profundidades moderadas em modelos hierárquicos são preferidas na prática e que os desenvolvedores normalmente evitam árvores profundas. Além disso, verificou-se a partir do trabalho de Bagheri e Gasevic (2011) que a profundidade de uma árvore está relacionada com o número de folha e o Número de Configurações (NVC), porém como as estruturas têm a mesmo numero de NVC essa primeira avaliação foi baseada apenas no número de folhas em relação à profundidade do modelo.

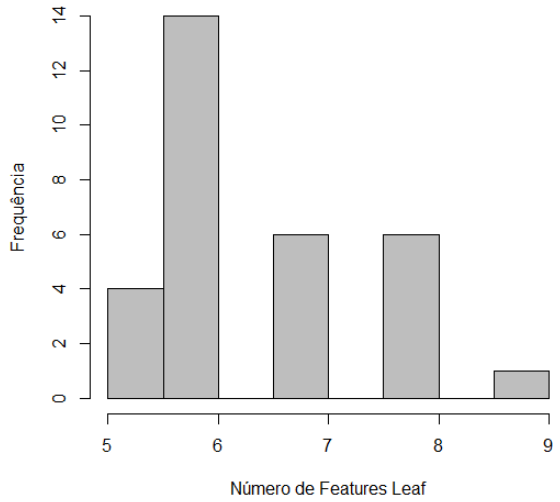
Figura 12 – Histograma para a medida DT



Fonte: Elaborada pelo autor.

A partir das Figuras 12 e 13 é possível verificar que a maioria dos modelos possuem NLeaf igual a 6 e DT igual a 3 o que significa um balanceamento entre as medidas. Para essa análise os modelos devem se aproximar dessas medidas para serem considerados balanceados.

Figura 13 – Histograma para a medida NLeaf



Fonte: Elaborada pelo autor.

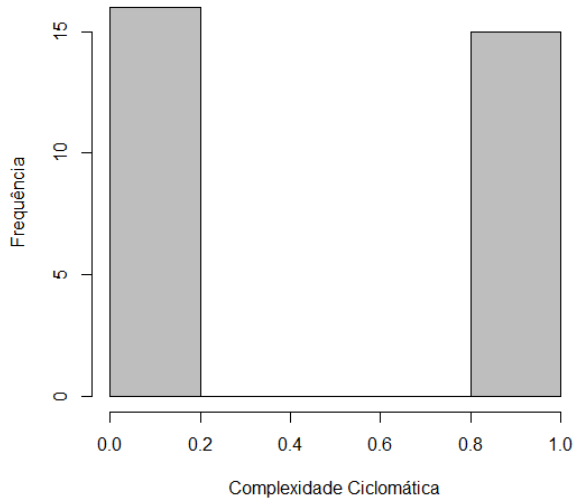
Considerando a influência da relação entre as medidas DT e NLeaf 0,54 foi realizado uma verificação de quais modelos apresentaram um estrutura mais equilibrada em relação a essas duas medidas. Foram então buscadas estruturas que possuíssem o número de folhas igual ao dobro de sua profundidade com no máximo uma unidade de diferença para cima e para baixo. No final da análise, observou-se que restaram apenas 15 modelos, sendo

eles os modelos Centrais A.2, A.3, A.4, A.5, A.7, A.8, A.9, A.11 e os modelos Bicentrais B.5, B.6, B.10, B.12, B.14, B.16, B.18 (BAGHERI; GASEVIC, 2011).

6.2 Avaliação pela complexidade do modelo de features

A complexidade estrutural está relacionada com a compreensão da estrutura do modelo de *features* (ISO. IEC 25010, 2011). Para analisar este aspecto, existem as seguintes medidas que estão diretamente relacionadas com esse aspecto, tais como: CyC, CTC, CoC, NF e ComC. Como as diferentes estruturas mantém o número de *features* obrigatórios, OR e XOR a análise foi baseada apenas nas medidas CyC, CTC e ComC.

Figura 14 – Histograma para a medida CyC

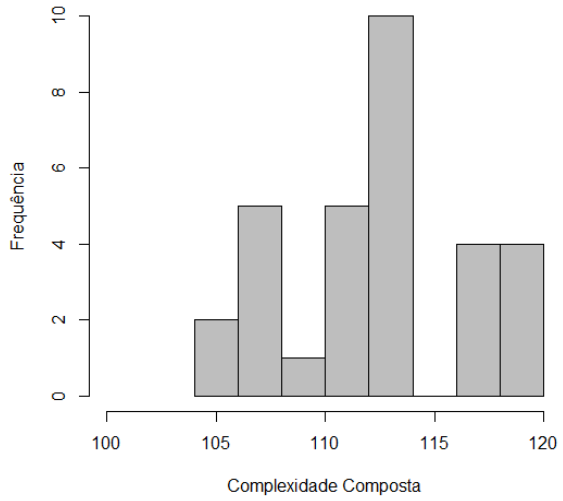


Fonte: Elaborada pelo autor.

A partir das Figuras 14 e 16 é possível verificar o alto grau de relacionamento entre as medidas CTC e CyC, uma vez que ambas estão diretamente relacionadas as *constraints*. Tendo isso em vista, as medidas das estruturas selecionadas devem apresentar o CTC e CyC proporcionalmente iguais.

A Figura 15 nos mostra grande concentração de modelos com complexidade entre 110 e 115, o que pode ser ocasionada pelas *features* que possuem *constraints* e devido a isso sua complexidade tende a aumentar.

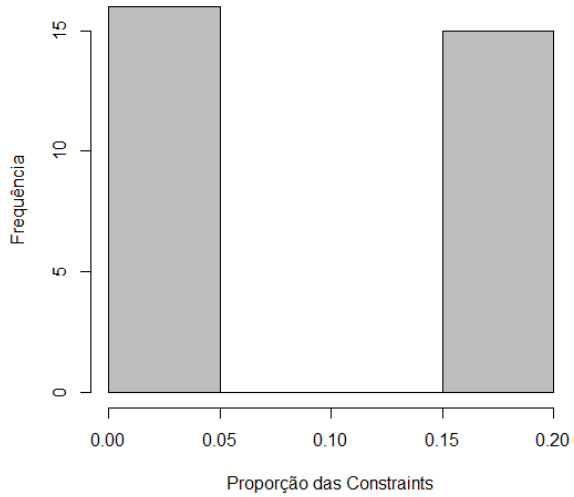
Figura 15 – Histograma para a medida ComC



Fonte: Elaborada pelo autor.

Nesta análise foi verificado para cada modelo de *features* a sua complexidade com base na medida ComC. Foi identificado que quanto menor o número de *constrains* menos complexo é o modelo de *feature*, tendo para o exemplo gerado como o número ótimo neste contexto o valor 0 para *constraints*. Após essa análise foi realizado uma busca por modelos que apresentassem os seguintes resultados de suas medidas CTC = 0, CyC = 0, CoC = 0 e ComC = 109.22, pois foram os resultados com menor complexidade para uma estrutura. Feito essa análise foi possível eliminar mais 11 modelos, tendo como restante os modelos A.2, A.4, A.5, A.8.

Figura 16 – Histograma para a medida CTC

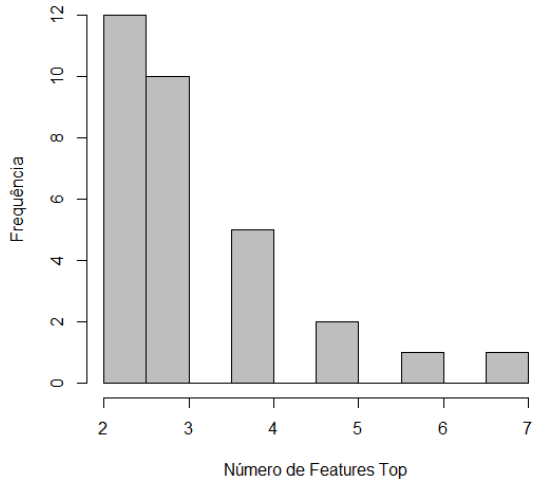


Fonte: Elaborada pelo autor.

6.3 Avaliação pela flexibilidade do modelo de *features*

Flexibilidade é a capacidade de um modelo de *features* responder a possíveis alterações internas ou externas que afetam a sua entrega de valor, de forma oportuna e rentável (ISO. IEC 25010, 2011). Para analisar este aspecto foram consideradas as medidas Nleaf e Ntop. De acordo com os trabalho de Bagheri, Gasevic (2011) e Berger e Guo (2014), a medida NLeaf tem relações com as medidas CyC, FoC, NVC, NF e NTop. Para esta análise foi utilizado apenas a medida NTop porque as outras medidas não têm qualquer variação entre as estruturas.

Figura 17 – Histograma para a medida NTop



Fonte: Elaborada pelo autor.

Realizando uma comparação entre a Figura 17 e 13 é possível verificar um desiquilíbrio a partir das *features* NTop em relação as *features* NLeaf, uma vez que o maior número de modelos tem 2 *features Top* e 6 *features Leaf*. Logo a partir dessa análise deve haver muitas exclusões, pois os melhores modelos tendem a estar mais próximo de 6, de acordo com a análise de profundidade.

Como a medida NTop tem um alto grau de relacionamento com a medida NLeaf foi realizada uma busca por estruturas que possuíam a menor diferença entre essas duas medidas. Com isso em mente, foi identificado que a menor diferença foi de apenas duas unidades que sérvio como número ótimo para todas as estruturas. Curiosamente as estruturas restantes apresentaram um crescimento de uma unidade entre eles para a medida NTop o que ocasionou uma variação de 2 até 5 desta medida. No final foi selecionado apenas o modelo A.2, porque ele era o único que obtinha o valor 5 na medida NTop que foi a menor diferença encontrada entre a medida NLeaf e NTop, sendo esta estrutura selecionada como a melhor estrutura gerada.

7 DISCUSSÃO

Os modelos de *features* são um dos artefatos gerados na fase de engenharia de domínio de uma Linha de Produto de Software (LPS). Tendo em vista a importância desses modelos para o gerenciamento de uma LPS, esse trabalho conseguiu identificar um processo de engenharia reversa e aplica-la em um contexto em que o usuário tenha a mínima interação para gerar estes modelos, sem a necessidade de conhecimento do contexto em que esse modelo será aplicado.

Para realização da construção das estruturas foi utilizado à combinação de sub-árvores apresentada nos cálculos proposto por Cayley (1881). Onde foi realizado o cálculo sobre o número de *features* e a partir dos resultados foram identificadas as combinações das sub-árovres necessárias para montar uma estrutura Central e Bicentral. No final do processo foi possível extrair um total de 34 modelos que foram validados a partir da linguagem FAMILIAR.

Para realizar a validação dos resultados do algoritmo proposto por este trabalho foi feito o mapeamento dos modelos para suas fórmulas proposicional utilizado o Quadro proposto por Benavides (2010) que foi ilustrado na Figura 2. Logo após foi feita a inserção na ferramenta FAMILIAR onde foi atribuída para um conjunto de variáveis da ferramenta a fórmula proposicional equivalente de cada modelo sintetizado. Por fim, foi realizada uma comparação de cada variável que continha a fórmula proposicional de um modelo com a fórmula extraída a partir da matriz de configuração para verificar se eram equivalentes. A partir dessa verificação foi constatado que apenas os modelos A.12, B.13 e B.14 não apresentavam fórmulas proposicionais equivalentes, sendo assim retirados da avaliação.

Com a avaliação realizada sobre os modelos de *features* também foi possível identificar modelos com estruturas iguais, pois apresentavam todas suas medidas iguais e devido a isso também foram excluídos da análise.

A partir dos resultados foi possível verificar que os critérios de avaliação causaram uma redução relevante sobre todos os modelos gerados a partir do processo de sintetização. Esse grande quantidade de exclusão se deve a necessidade de maior precisão das medidas aos seus números ótimos. Essa maior precisão se fazia necessária, pois os valores dos diferentes modelos gerados tendem a ser aproximados.

De acordo com a análise realizada sobre a manutenibilidade dos modelos de *features* sintetizados foi possível selecionar apenas o modelo A.2 devido a sua distribuição homogênea das *features* o que lhe garantiu uma profundidade equilibrada em relação ao

número de folhas que é a medida que possui maior relação, baixa complexidade por possuir poucas *features* agrupadas e não apresentar nenhum tipo de *constraint* e possuir uma maior flexibilidade devido ao seu maior número de *features top* que está relacionado também ao número de *features*.

7.1 Ameaças à Validade

De acordo com WOHLIN (2012), existem quatro tipos de ameaças à validade que são: validade da conclusão, validade interna, validade de construção e validade externa.

De acordo com o estudo sobre a descrição de cada tipo de ameaça foi realizado o enquadramento dos possíveis problemas encontrados no decorrer do trabalho em cada uma delas.

De início foi verificado a definição de validade da conclusão que de maneira geral refere-se ao número reduzido de amostras para execução do trabalho, o que pode reduzir a capacidade de revelar os padrões de dados. Tendo isso em mente, foi verificado que a sintetização de um modelo foi aplicado para apenas um exemplo o que pode ser considerado uma ameaça de conclusão.

A validade interna esta relacionada à casualidade dos dados, onde se deve verificar se os dados não estão direcionados a um resultado. A primeira ameaça relacionada a validade interna é a não validação da semântica do modelo de *features,* uma fez que a avaliação está preocupada apenas com a manutenibilidade do modelo que é relacionada com a sintaxe. A segunda ameaça está relacionada à corretude dos modelos gerados, uma fez que é utilizado engenharia reversa para a criação dos modelos e não é passado como entrada todos os possíveis produtos de uma LPS, não há garantia que os modelos gerados irão possuir todos os seus relacionamentos corretos.

A validade externa refere-se à qualidade e corretude dos dados extraídos de outros ambientes que não estão inclusos no trabalho realizado. Uma ameaça encontrada para essa classificação foi a utilização de um modelo de *features* extraído a partir do repositório SPLOT onde qualquer usuário pode submeter um modelo de *features* para o repositório.

Por fim a validade de construção está relacionada a generalização dos resultados que diz respeito a aplicabilidade do trabalho em diferentes contextos de uma mesma área. Não foram identificados ameaças nesse tipo de classificação.

8 CONSIDERAÇÕES FINAIS

Esta Seção é composta por considerações que devem ser levadas em conta a respeito do trabalho realizado. Inicialmente será apresentado a possíveis ameaças que podem comprometer a validade dos resultados deste trabalho. Logo após é feito um levantamento sobre os possíveis trabalhos futuros, apresentando indícios de extensão e melhoria do trabalho. Por fim, são descritas as conclusões a respeito da execução deste trabalho.

8.1 Trabalhos Futuros

A partir dos resultados foi possível identificar diversas melhorias e extensões a serem aplicadas, além de novas possibilidades de trabalhos. Algumas dessas extensões são descritas a seguir.

Uma melhoria que pode ser aplicado sobre este trabalho é a utilização de conhecimento ontológico sobre as *features*, onde essa *features* são agrupadas de acordo com sua semântica, utilizando para isso ferramentas de mineração de dados textuais em conjunto com algoritmos, tais como Smith-Waterman para realizar agrupamentos de segmentos do modelo de *features.* Com isso seria possível reorganizar as *features* nas estruturas de acordo com sua semântica. Essa abordagem já é realizada a partir do operador *ksynthesis* da linguagem FAMILIAR, contudo ela não permite a aplicação em estruturas predefinidas.

Outra melhoria seria a utilização de outras medidas relacionadas a manutenibilidade tais como a Taxa de Conectividade do Grafo para analisar a flexibilidade do modelo ou a Media de *Constraints* Referenciadas por *Features* para verificar a complexidade do modelo.

Também poderia ser realizado a implementação do algoritmo proposto por Czarnecki e Wasowski (2007) e depois verificar seu desempenho. Logo após realizar a implementação do algoritmo proposto por este trabalho, realizar a integração entre os dois algoritmos e verificar seu desempenho para diferentes tamanhos de entradas.

Outra aplicação seria adaptar o processo desenvolvido nesse trabalho em *features* com atributos adicionado alguns dos passos propostos por Bécan *et al*. (2015). A entrada para esse processo seria uma matriz tridimensional onde os eixos x e y representariam as *features* e seus produtos respectivamente e o eixo z seria composto pelos atributos dessas *features*.

Uma ultima contribuição seria a utilização de um mecanismo de pontuação para realizar as análises, onde cada atributo de qualidade possui um peso em relação a sua característica de qualidade. Então, são definidos três pontuações com valores alto, médio e

baixo, onde para cada medida é definido um valor ótimo em que os modelos que mais se aproximarem desse valor recebem pontuação máxima. No final das avaliações o melhor modelo de *feature* para representar uma LPS deve possuir a maior pontuação.

8.2 Conclusões

Este trabalho realizou a avaliação dos modelos de *features* gerados a partir de um algoritmo definido para realizar a construção de vários modelos de *features* com estruturas distintas, utilizando um processo de engenharia reversa que tem como entrada um conjunto de configurações dos produtos.

Todos os seus objetivos foram alcançados uma vez que o seu resultado final apresentou apenas um modelo de *feature* que foi considerado o melhor entre todos os modelos gerados para representar uma LPS de acordo com a análise.

Como previsto na Seção 5.2 tanto as estruturas Centrais como as Bicentrais apresentaram uma distribuição homogênea de suas *features* em relação a sua profundidade o que provocou uma remoção equilibrada dos dois tipos de estruturas para este critério de avaliação.

Outra descoberta é que as ferramentas voltadas para LPS, que foram analisadas neste trabalho, não possuem funcionalidades para dar suporte a este tipo de engenharia reversa para a construção de múltiplos modelos de *features* sem o conhecimento do domínio. O operador *ksynthesis* da linguagem FAMILIAR embora trabalhe com sintetização de modelos, se faz necessário um maior número de interações com o usuário para determinar partes da hierarquia do modelo gerando e a partir de heurísticas aplicadas sobre as *features* é gerado um modelo que é considerado o melhor para representar uma LPS. Contudo, não é aplicado nenhum tipo de avaliação sobre o modelo gerado pelo *ksynthesis* o que não garante a qualidade do modelo.

Um dos pontos positivos a partir da abordagem utilizada foi a possibilidade de se obter mais de um modelo como resultado final da avaliação, o que possibilita ao usuário final selecionar o modelo que mais se adegue a seu contexto.

Um ponto negativo é que o algoritmo não pode ser aplicado em modelos com grandes quantidades de *features*, pois ele é classificado como NP-completo. O objetivo deste trabalho é avaliar um conjunto de modelos de *features* que serão extraídos a partir das configurações dos produtos.

Com implementação deste trabalho em empresas ou no meio acadêmico será possível reduzir o tempo e esforço na criação de modelos de *features* e evitar que erros

manuais sejam inseridos, garantindo assim uma maior qualidade dos trabalhos realizados sobre a área de LPS.

REFERÊNCIAS

ABÍLIO, Ramon Simões. **Detecting code smells in software product lines**. 2014. 144 f. Dissertação (mestrado) – Departamento de Ciência da Computação. Universidade Federal de Lavras, Lavras, 2014.

ACHER, Mathieu *et al*. Familiar: A domain-specific language for large scale management of *feature* models. **Science of Computer Programming**, v. 78, n. 6, p. 657-681, 2013.

ACHER, Mathieu *et al*. Support for reverse engineering and maintaining *feature* models. *In*: **Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems.** ACM, 2013. p. 20.

AL-MSIE'DEEN, R. et al. Reverse Engineering *Feature* Models from Software Configurations using Formal Concept Analysis. In: **CLA 2014: Eleventh International Conference on Concept Lattices and Their Applications**. 2014. p. 95-106.

ALVES, Vander et al. Refactoring product lines. In: **Proceedings of the 5th international conference on Generative programming and component engineering**. ACM, 2006. p. 201-210.

BAGHERI, Ebrahim; GASEVIC, Dragan. Assessing the maintainability of software product line *feature* models using structural metrics. **Software Quality Journal**, v. 19, n. 3, p. 579-612, 2011.

BATORY, Don. ***Feature* models, grammars, and propositional fórmulas**. Springer Berlin Heidelberg, 2005.

BÉCAN, Guillaume *et al*. Automating the formalization of product comparison matrices. *In*: **Proceedings of the 29th ACM/IEEE international conference on Automated software engineering.** ACM, 2014. p. 433-444.

BÉCAN, Guillaume *et al*. Breathing Ontological Knowledge Into *Feature* Model Synthesis: An Empirical Study. **Empirical Software Engineering**, p. 51, 2015.

BÉCAN, Guillaume et al. Synthesis of attributed *feature* models from product descriptions. In: **Proceedings of the 19th International Conference on Software Product Line**. ACM, 2015. p. 1-10.

BENAVIDES, David *et al*. FAMA: Tooling a *Framework* for the Automated Analysis of *Feature* Models. **VaMoS**, v. 2007, p. 01, 2007.

BENAVIDES, David; SEGURA, Sergio; RUIZ-CORTÉS, Antonio. Automated analysis of *feature* models 20 years later: A literature review. **Information Systems**, v. 35, n. 6, p. 615-636, 2010.

BERGER, Thorsten *et al*. A survey of variability modeling in industrial practice. *In*: **Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems.** ACM, 2013. p. 7.

BERGER, Thorsten; GUO, Jianmei. Towards system analysis with variability model metrics. In: **Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems**. ACM, 2014. p. 23.

BEZERRA, Carla IM; ANDRADE, Rossana MC; MONTEIRO, José Maria S. Measures for Quality Evaluation of *Feature* Models. *In*: **Software Reuse for Dynamic Systems in the Cloud and Beyond.** Springer International Publishing, 2014. p. 282-297.

BÖCKLE, Günter; POHL, Klaus; VAN DER LINDEN, Frank. **Software Product Line Engineering Foundations, Principles and Techniques**. 1. ed. Springer Berlin Heidelberg, 2005.

BOSCH, Jan. **Software Product Lines: Going Beyond: 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings**. Springer Science & Business Media, 2010.

CAYLEY, Arthur. LVIII. On the analytical forms called trees.–Part II. **The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science**, v. 18, n. 121, p. 374-378, 1859.

CAYLEY, Arthur. On the analytical forms called trees, with application to the theory of chemical combinations. **Rep. Brit. Assoc. Advance. Sci**, v. 45, p. 257-305, 1875.

CAYLEY, Professor. On the analytical forms called trees. **American Journal of Mathematics**, v. 4, n. 1, p. 266-268, 1881.

CAPILLA, Rafael; BOSCH, Jan. The promise and challenge of runtime variability. **Computer**, v. 44, n. 12, p. 93-95, 2011.

CZARNECKI, Krzysztof; WASOWSKI, Andrzej. *Feature* diagrams and logics: There and back again. In: **Software Product Line Conference, 2007. SPLC 2007. 11th International**. IEEE, 2007. p. 23-34.

DEHMOUCH, Ikram. Towards an agile *feature* composition for a large scale software product lines. *In*: **Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on.** IEEE, 2014. p. 1-6.

EÉN, Niklas; SÖRENSSON, Niklas. An extensible SAT-solver. *In*: **Theory and applications of satisfiability testing**. Springer Berlin Heidelberg, 2004. p. 502-518.

ERIKSSON, Magnus; HAGGLUNDS, Alvis. An Introduction to Software Product Line Development. *In*: **Proceedings of Umeå's Seventh Student Conference in Computing Science, UMINF**. 2003. p. 26-37.

ETXEBERRIA, Leire; SAGARDUI, Goiuria; BELATEGI, Lorea. Quality aware software product line engineering. **Journal of the Brazilian Computer Society**, v. 14

FERNANDES, Paula; WERNER, Cláudia; MURTA, Leonardo Gresta Paulino. *Feature* Modeling for Context-Aware Software Product Lines. *In*: **SEKE.** 2008. p. 758-763.

FERNANDES, Paula; WERNER, Cláudia; TEIXEIRA, Eldânae. An Approach for *Feature* Modeling of Context-Aware Software Product Line. **J. UCS**, v. 17, n. 5, p. 807-829, 2011.

GERSTING, Judith L. **Fundamentos matemáticos para a ciência da computação: um tratamento moderno de matemática discreta**. Livros Técnicos e Científicos, 2004.

GHANAM, Yaser. **An agile *framework* for variability management in software product line engineering.** 2012. 264 f. Tese (doutorado) – Department of Computer Science University of Calgary, Calgary, 2012.

GHEYI, Rohit; MASSONI, Tiago; BORBA, Paulo. Automatically Checking *Feature* Model Refactorings. **J. UCS**, v. 17, n. 5, p. 684-711, 2011.

GRISS, Martin L.; FAVARO, John; D'ALESSANDRO, Massimo. Integrating *feature* modeling with the RSEB. *In*: **Software Reuse, 1998. Proceedings. Fifth International Conference on.** IEEE, 1998. p. 76-85.

HASLINGER, Evelyn Nicole; LOPEZ-HERREJON, Roberto E.; EGYED, Alexander. Reverse engineering *feature* models from programs' *feature* sets. *In*: **Reverse Engineering (WCRE), 2011 18th Working Conference on**. IEEE, 2011. p. 308-312.

HAUGEN, Oystein *et al*. Adding standardized variability to domain specific languages. *In*: **Software Product Line Conference, 2008. SPLC'08. 12th International**. IEEE, 2008. p. 139-148.

HEYMANS, Patrick; TRIGAUX, Jean-Christophe. Software product line: state of the art. **Relatório Técnico EPH3310300R0462/215315, Product Line Engineering of food Traceability software (PLENTY) Project, Institut d'Informatique, FUNDP, Namur**, 2003.

HOLANDA JUNIOR, João F.; **DyMMEr: Uma ferramenta para avaliação de qualidade do modelo de *features* baseada em menidas de Linhas de Produtos de Software Dinâmicas**. 2014. 60 f. Monografia (Graduação em Sistemas de Informação) - Centro de Tecnologia, Universidade Federal do Ceará, Fortaleza, 2014.

ISO, ISO. IEC 25010: 2011: Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models. **International Organization for Standardization**, 2011.

KANG, Kyo C. *et al*. ***Feature*-oriented domain analysis (FODA) feasibility study.** CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1990.

KANG, Kyo C. et al. FORM: A *feature*-oriented reuse method with domain-specific reference architectures. **Annals of Software Engineering**, v. 5, n. 1, p. 143-168, 1998.

KASTNER, Christian *et al*. *Feature*IDE: A tool *framework* for *feature*-oriented software development. *In*: **Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on**. IEEE, 2009. p. 611-614.

KRUEGER, Charles W. The BigLever Software Gears Systems and Software Product Line Lifecycle *Framework*. *In*: **SPLC Workshops**. 2010. p. 297.

LAUENROTH, Kim; METZGER, Andreas; POHL, Klaus. Quality assurance in the presence of variability. *In*: **Intentional Perspectives on Information Systems Engineering.** Springer Berlin Heidelberg, 2010. p. 319-333.

LIU, Jing; DEHLINGER, Josh; LUTZ, Robyn. Safety analysis of software product lines using state-based modeling. **Journal of Systems and Software**, v. 80, n. 11, p. 1879-1892, 2007.

LOPEZ-HERREJON, Roberto E. et al. An assessment of search-based techniques for reverse engineering *feature* models. **Journal of Systems and Software**, v. 103, p. 353-369, 2015.

LUCRÉDIO, Daniel *et al*. Software reuse: The Brazilian industry scenario. **Journal of Systems and Software**, v. 81, n. 6, p. 996-1013, 2008.

MENDONÇA, Marcılio. **Efficient reasoning techniques for large scale *feature* models**. 2009. Tese de Doutorado. University of Waterloo.

MENDONCA, Marcilio; BRANCO, Moises; COWAN, Donald. SPLOT: software product lines online tools. *In*: **Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications**. ACM, 2009. p. 761-762.

MENDONCA, Marcilio; WĄSOWSKI, Andrzej; CZARNECKI, Krzysztof. SAT-based analysis of *feature* models is easy. In: **Proceedings of the 13th International Software Product Line Conference**. Carnegie Mellon University, 2009. p. 231-240.

MONTAGUD, Sonia; ABRAHÃO, Silvia. Gathering current knowledge about quality evaluation in software product lines. *In*: **Proceedings of the 13th International Software Product Line Conference.** Carnegie Mellon University, 2009. p. 91-100.

PEREIRA, Francisco A. *et al*. Linhas de Produto de Software: Uma Tendência da Indústria. **V Encontro Regional de Informática Ceará-Piauí (ERCEMAPI 2011)**, Cap, v. 1, 2011.

POHL, Klaus; BÖCKLE, Günter; VAN DER LINDEN, Frank J. **Software product line engineering: foundations, principles and techniques**. Springer Science & Business Media, 2005.

SCHAEFER, Ina; STAMELOS, Ioannis (Ed.). **Software Reuse for Dynamic Systems in the Cloud and Beyond: 14th International Conference on Software Reuse, ICSR 2015, Miami, FL, USA, January 4-6, 2015. Proceedings**. Springer, 2014.

SHE, Steven *et al*. Efficient synthesis of *feature* models. **Information and Software Technology**, v. 56, n. 9, p. 1122-1143, 2014.

SHE, Steven *et al*. Reverse engineering *feature* models. *In*: **Software Engineering (ICSE), 2011 33rd International Conference on.** IEEE, 2011. p. 461-470.

SOMMERVILLE, I. **Engenharia de Software**. 9ª. ed. São Paulo: Pearson Education do Brasil, 2011.

SVENDSEN, Andreas; HAUGEN, Øystein; MØLLER-PEDERSEN, Birger. Analyzing variability: capturing semantic Ripple effects. *In*: **Modelling Foundations and Applications**. Springer Berlin Heidelberg, 2011. p. 253-269.

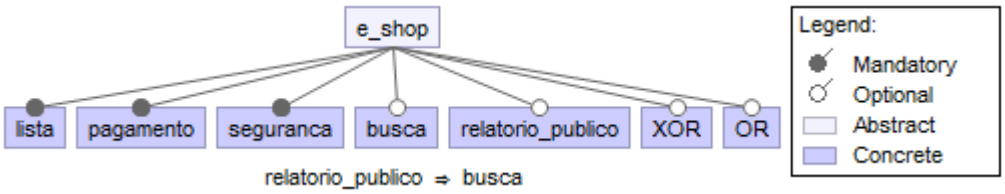
THÜM, Thomas *et al*. *Feature*ide: An extensible *framework* for *feature*-oriented software development. **Science of Computer Programming**, v. 79, p. 70-85, 2014.

WOHLIN, Claes et al. **Experimentation in software engineering**. Springer Science & Business Media, 2012.

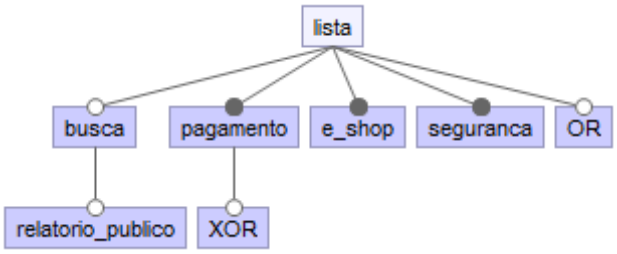
APÊNDICES

APÊNDICE A – Modelos de *Features* Centrais

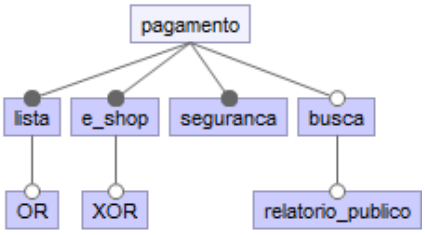
A.1)



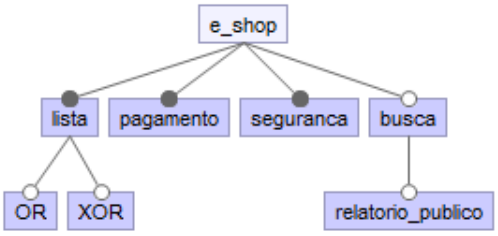
A.2)



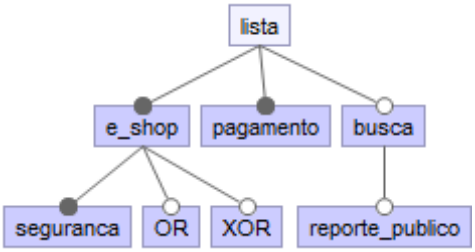
A.3)



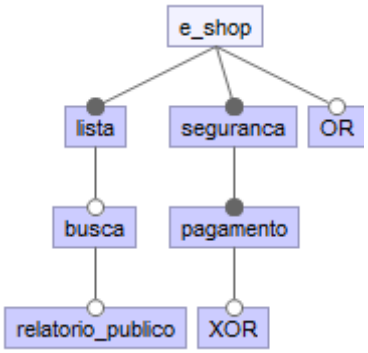
A.4)



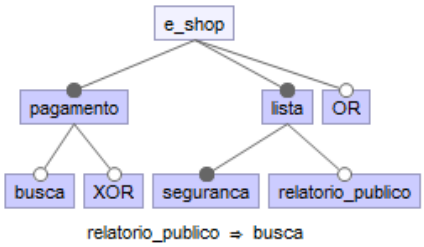
A.5)



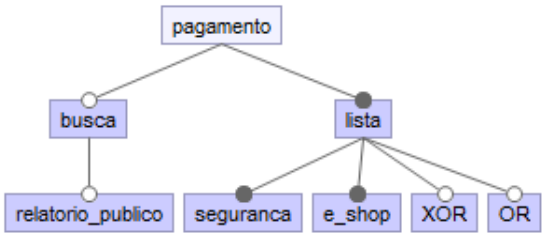
A.6)



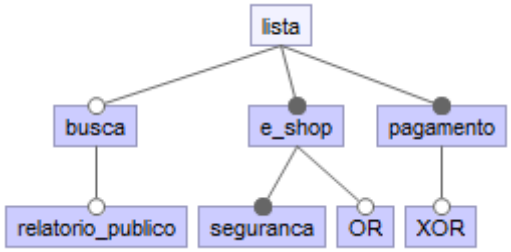
A.7)



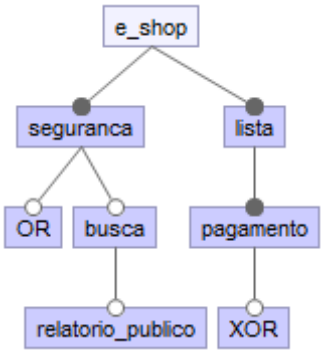
A.8)



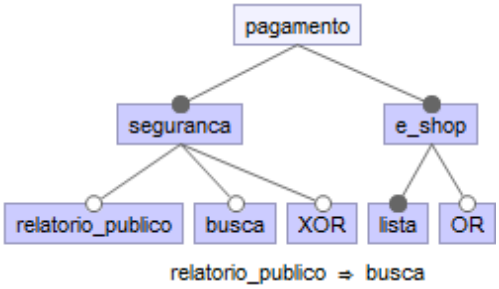
A.9)



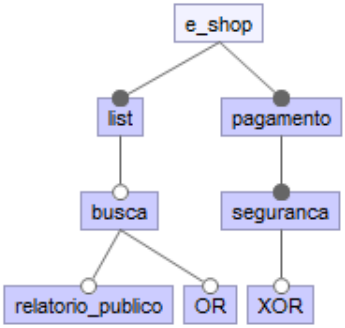
A.10)



A.11)

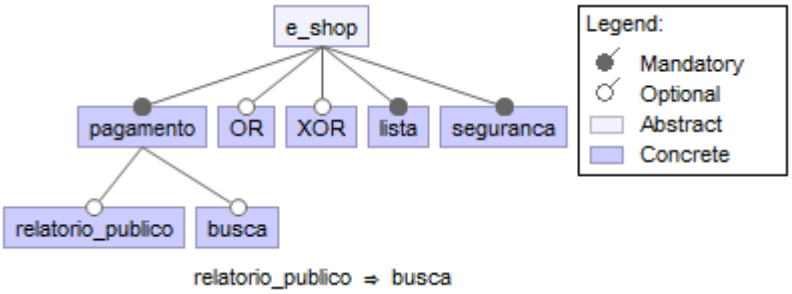


A.12)

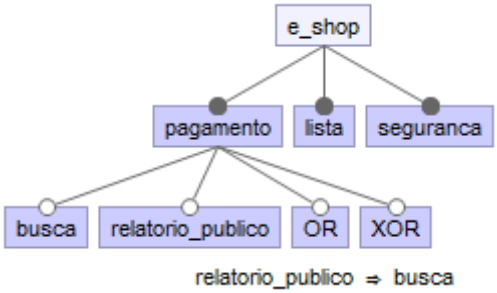


APÊNDICE B – Modelos de *Features* Bicentrais

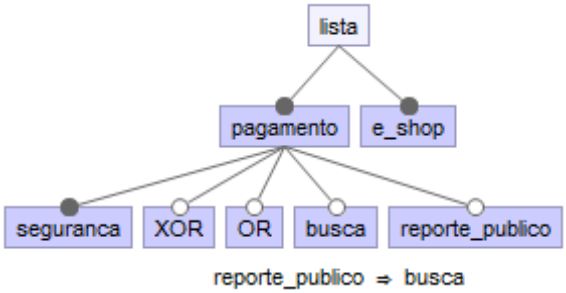
B.1)



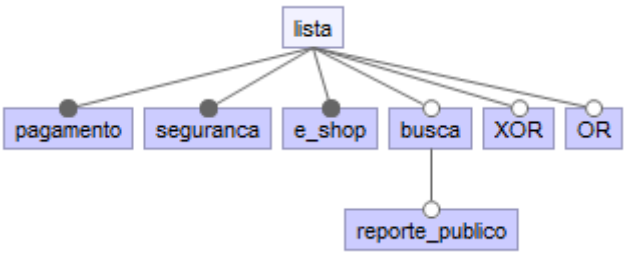
B.2)



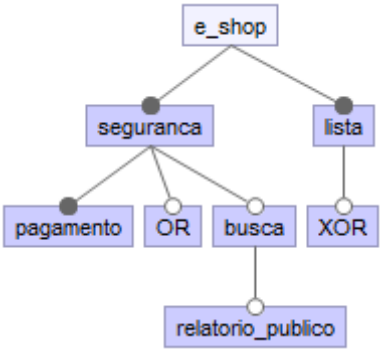
B.3)



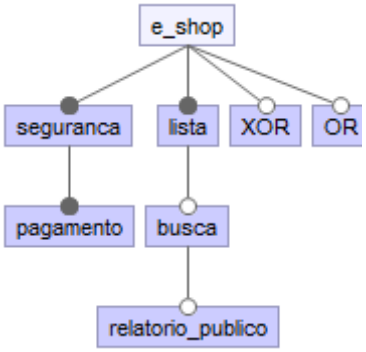
B.4)



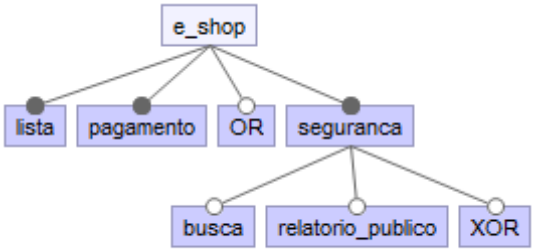
B.5)



B.6)

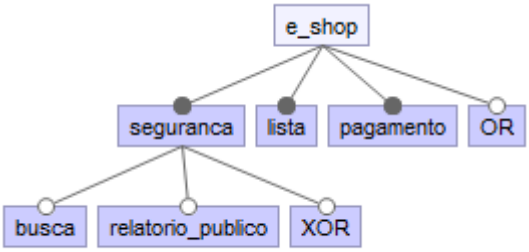


B.7)



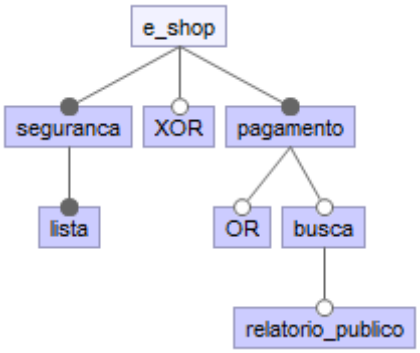
relatorio_publico ⇒ busca

B.8)

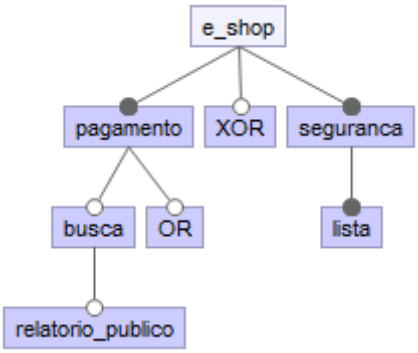


relatorio_publico ⇒ busca

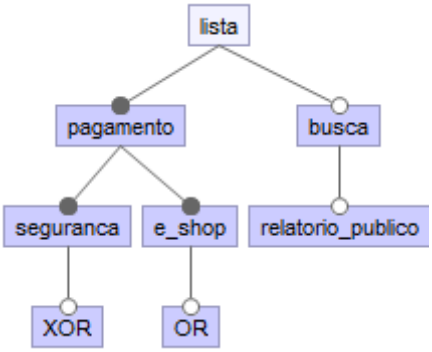
B.9)



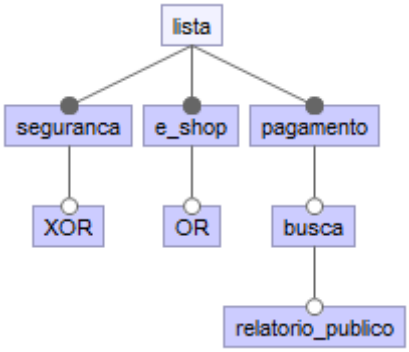
B.10)



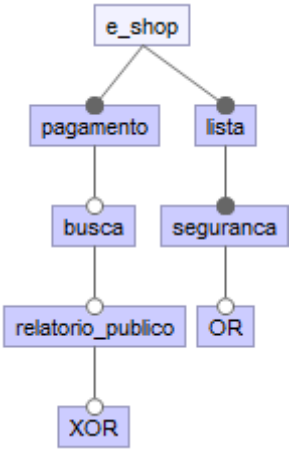
B.11)



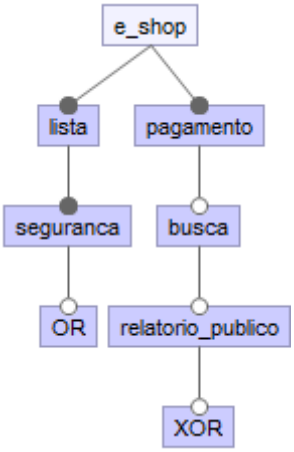
B.12)



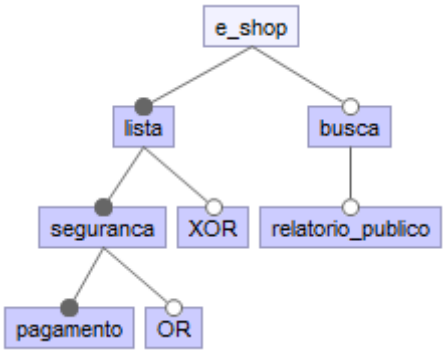
B.13)



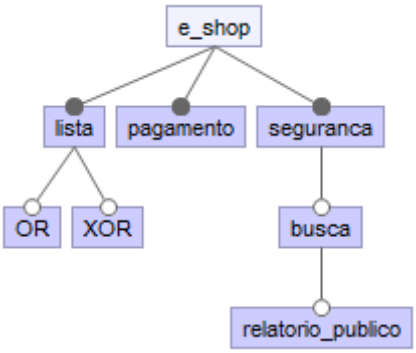
B.14)



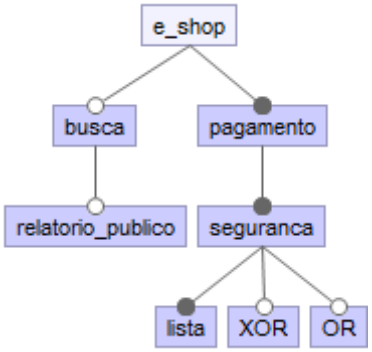
B.15)



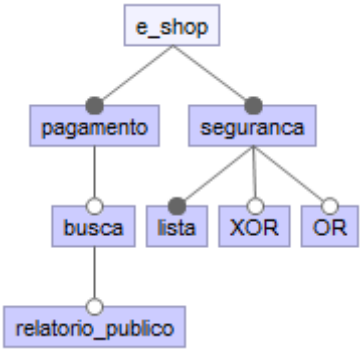
B.16)



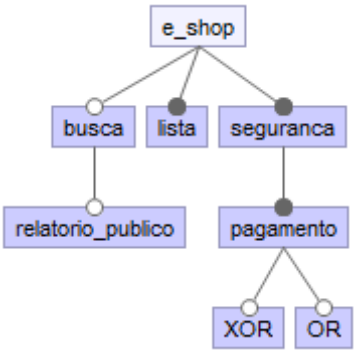
B.17)



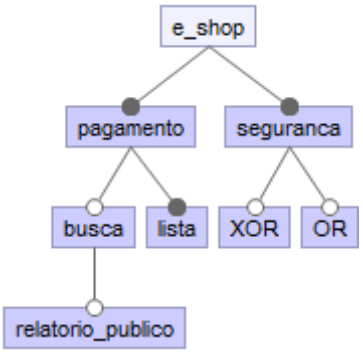
B.18)



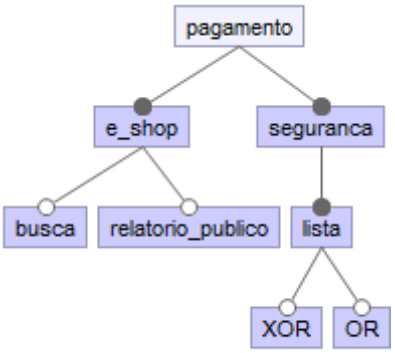
B.19)



B.20)

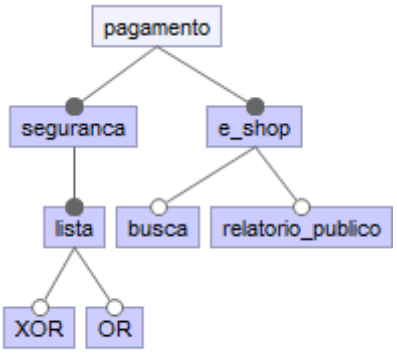


B.21)



relatorio_publico ⇒ busca

B.22)



relatorio_publico ⇒ busca

APÊNDICE C – Medidas das estruturas Centrais

Extração das medidas para as 11 estruturas Centrais que formaram uma representação válida de sua estrutura.

Metrics	NF	NA	NO	NLeaf	CogC	FoC	SCDF	MCDF	FEX	CyC	ComC
A.1	10	2	2	9	2	0.2	0	0	9	1	104.88
A.2	10	2	2	7	2	0.2	0	0	7	0	109.22
A.3	10	2	2	6	2	0.2	0	0	6	0	113.88
A.4	10	2	2	7	2	0.2	0	0	7	0	109.22
A.5	10	2	2	7	2	0.2	0	0	7	0	109.22
A.6	10	2	2	5	2	0.2	0	0	5	0	119.88
A.7	10	2	2	7	2	0.2	0	0	7	1	111.55
A.8	10	2	2	7	2	0.2	0	0	7	0	109.22
A.9	10	2	2	6	2	0.2	0	0	6	0	113.88
A.10	10	2	2	5	2	0.2	0	0	5	0	119.88
A.11	10	2	2	7	2	0.2	0	0	7	1	111.55

Metrics	CTC	CoC	DT	NTop	SHoF	MHoF	RNoF	NVC	NVF	RoV
A.1	0.2	0.9	2	7	2	2	6	18	4	0.18
A.2	0	0.9	3	5	2	2	6	18	4	0.18
A.3	0	0.9	3	4	2	2	6	18	4	0.18
A.4	0	0.9	3	4	2	2	6	18	4	0.18
A.5	0	0.9	3	3	2	2	6	18	4	0.18
A.6	0	0.9	4	3	2	2	6	18	4	0.18
A.7	0.2	0.9	3	3	2	2	6	18	4	0.18
A.8	0	0.9	3	2	2	2	6	18	4	0.18
A.9	0	0.9	3	2	2	2	6	18	4	0.18
A.10	0	0.9	4	2	2	2	6	18	4	0.18
A.11	0.2	0.9	3	2	2	2	6	18	4	0.18

APÊNDICE D – Medidas das estruturas Bicentrais

Extração das medidas para as 20 estruturas Bicentrais que formaram uma representação válida de sua estrutura.

Metrics	NF	NA	NO	NLeaf	CogC	FoC	SCDF	MCDF	FEX	CyC	ComC
B.1	10	2	2	8	2	0.2	0	0	8	1	107.55
B.2	10	2	2	8	2	0.2	0	0	8	1	107.55
B.3	10	2	2	8	2	0.2	0	0	8	1	107.55
B.4	10	2	2	8	2	0.2	0	0	8	0	105.88
B.5	10	2	2	6	2	0.2	0	0	6	0	113.88
B.6	10	2	2	6	2	0.2	0	0	6	0	113.88
B.7	10	2	2	8	2	0.2	0	0	8	1	107.55
B.8	10	2	2	8	2	0.2	0	0	8	1	107.55
B.9	10	2	2	6	2	0.2	0	0	6	0	113.88
B.10	10	2	2	6	2	0.2	0	0	6	0	113.88
B.11	10	2	2	5	2	0.2	0	0	5	0	119.88
B.12	10	2	2	5	2	0.2	0	0	5	0	119.88
B.15	10	2	2	6	2	0.2	0	0	6	0	113.88
B.16	10	2	2	6	2	0.2	0	0	6	0	113.88
B.17	10	2	2	6	2	0.2	0	0	6	0	113.88
B.18	10	2	2	6	2	0.2	0	0	6	0	113.88
B.19	10	2	2	6	2	0.2	0	0	6	0	113.88
B.20	10	2	2	6	2	0.2	0	0	6	0	113.88
B.21	10	2	2	6	2	0.2	0	0	6	1	116.88
B.22	10	2	2	6	2	0.2	0	0	6	1	116.88

Metrics	CTC	CoC	DT	NTop	SHoF	MHoF	RNoF	NVC	NVF	RoV
B.1	0.2	0.9	2	5	2	2	6	18	4	0.18
B.2	0.2	0.9	3	3	2	2	6	18	4	0.18
B.3	0.2	0.9	3	2	2	2	6	18	4	0.18
B.4	0	0.9	2	6	2	2	6	18	4	0.18
B.5	0	0.9	3	2	2	2	6	18	4	0.18
B.6	0	0.9	3	4	2	2	6	18	4	0.18
B.7	0.2	0.9	3	4	2	2	6	18	4	0.18
B.8	0.2	0.9	3	4	2	2	6	18	4	0.18
B.9	0	0.9	3	3	2	2	6	18	4	0.18
B.10	0	0.9	3	3	2	2	6	18	4	0.18
B.11	0	0.9	4	2	2	2	6	18	4	0.18
B.12	0	0.9	3	3	2	2	6	18	4	0.18
B.15	0	0.9	4	2	2	2	6	18	4	0.18
B.16	0	0.9	3	3	2	2	6	18	4	0.18
B.17	0	0.9	4	2	2	2	6	18	4	0.18
B.18	0	0.9	3	2	2	2	6	18	4	0.18
B.19	0	0.9	4	3	2	2	6	18	4	0.18
B.20	0	0.9	3	2	2	2	6	18	4	0.18
B.21	0.2	0.9	4	2	2	2	6	18	4	0.18
B.22	0.2	0.9	4	2	2	2	6	18	4	0.18