UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALISSON GASPAR CHIQUITTO

Gerenciamento de Variabilidades com CVL na Abordagem SyMPLES

Maringá
2015

ALISSON GASPAR CHIQUITTO

Gerenciamento de Variabilidades com CVL na Abordagem SyMPLES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Itana Maria de Souza Gimenes

Maringá
2015

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR, Brasil)

Chiquitto, Alisson Gaspar
C541g Gerenciamento de variabilidades com CVL na abordagem SyMPLES / Alisson Gaspar Chiquitto. -- Maringá, 2015.
123 f.: il.; color., figs.,fotos.

Orientadora: Profa. Dra. Itana Maria de Souza Gimenes.

Dissertação (Mestre em Ciência da Computação) - Universidade Estadual de Maringá. Centro de Tecnologia, Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação.

1. Sistemas embarcados. 2. Linha de Produto de Software. 3. Gerenciamento de variabilidade. 4. SyMPLES. 5. SMarty. I. Gimenes, Itana Maria de Souza. II. Universidade Estadual de Maringá. Centro de Tecnologia, Programa de Pós graduação em Ciência da Computação. III. Título.

21.ed. 005

FOLHA DE APROVAÇÃO

ALISSON GASPAR CHIQUITTO

Gerenciamento de variabilidades com CVL na abordagem SyMPLES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Profa. Dra. Itana Maria de Souza Gimenes
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Edson Alves de Oliveira Junior
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Avelino Francisco Zorzo
Pontifícia Universidade Católica do Rio Grande do Sul – FACIN/PUCRS

Aprovada em: 15 de junho de 2015.
Local da defesa: Sala 102, Bloco C56, *campus* da Universidade Estadual de Maringá.

DEDICATÓRIA

Dedico este trabalho às mulheres de minha vida.
À minha mãe, eterno exemplo de bondade, carinho e amor. Modelo de conquistas e superação de limites, por sempre me amar e lutar para que hoje eu chegasse à vitória, para que fosse o homem que sou.
À minha amada esposa Andréia. Minha companheira, que esta sempre ao meu lado de mãos dadas, impulsionando-me a sempre ser melhor.

AGRADECIMENTOS

Quando eu me recordo das dificuldades durante a minha trajetória do início do Mestrado até este momento em que finalizo este trabalho percebo que muitas histórias compõem esta caminhada, e que ajudam a ser quem eu sou. Em cada uma dessas histórias, me recordo de pessoas que foram importantes e me influenciaram a ser quem sou. Certamente sem estas pessoas eu não conseguiria chegar onde estou.

Esta dissertação é produto de uma pesquisa que envolveu a participação, direta e indireta, de inúmeras pessoas, as quais merecem meus sinceros agradecimentos.

Aos meus pais, Felipe e Aparecida, por terem me ensinado, através de palavras e exemplos, a importância dos estudos e da busca pelo conhecimento na minha vida. Agradeço pela formação que me deram, pelo amor que sempre demonstraram por mim, por sempre terem me incentivado a estudar, me apoiando em cada nova jornada de estudos e trabalho. Agradeço ao meu padrasto Adalto, pai de minha "irmãzinha" Anna Luiza, por apoiar minha mãe na minha educação.

Agradeço aos meus avós, principalmente aos que faleceram durante estes últimos dois anos, Miguel e Américo, porque suas histórias de dificuldades e superação servem de exemplo. Nunca esquecerei de suas palavras e momentos compartilhados.

À minha melhor amiga e esposa, Andréia, pelo amor e admiração, que me servem de estímulo e motivação. Te amo.

Agradeço à professora Itana Maria de Souza Gimenes, por orientar meus passos na elaboração deste trabalho e no caminho que a ela sucedeu. Agradeço-lhe, ainda, pelos desafios a mim apresentados e pela valiosa confiança em mim depositada para superá-los.

Ao professor Edson A. Oliveira Júnior por seu tempo dedicado a mim pelas importantes palavras e conhecimentos compartilhados servindo como orientação.

A todos os professores do Departamento de Informática da UEM, que contribuíram de forma direta ou indireta com este trabalho, minha formação e desafios lançados.

Agradeço ao Rogério Ferreira da Silva e Vanderson Fragal que sempre me apoiaram, fornecendo documentos e arquivos, para a continuidade da pesquisa da abordagem SyMPLES.

Não posso esquecer aqueles que sempre estiveram dispostos a responder meus emails de dúvidas. Um obrigado a Tayana Conte, Igor Steinmacher e Øystein Haugen.

Aos professores, Sueli Poppi Borba, Roni Francis Shigueta e Luiz Fernando Braga Lopes, pelas preciosas palavras dedicadas a mim em minha carta de recomendação para este mestrado.

À UNIPAR, pelo auxílio financeiro do programa de capacitação de professores e pela disponibilização de recursos físicos para esta pesquisa.

À Maria Inês Davanço, que ainda na pré-seleção dos candidatos demonstrou amizade, paciência e compreensão. Eu a admiro pelo excelente profissionalismo.

Aos meus companheiros de mestrado, pelos grupos de estudo e por compartilharem seus conhecimentos e experiências.

Um obrigado a todos os meus amigos, que tornaram os momentos difíceis em momentos prazerosos de bate-papos e diversão.

Enfim, agradeço a todos os familiares, amigos, professores e colegas pela ajuda e compreensão que recebi durante o perído desse mestrado, com demonstrações de solidariedade, apoio e superação.

"Só o que é bom dura tempo o bastante pra
se tornar inesquecível."
Chorão - Charlie Brown Jr.

Gerenciamento de Variabilidades com CVL na Abordagem SyMPLES

RESUMO

A abordagem *SysML-based Product Line Approach for Embedded Systems* (SyMPLES-SMarty) utiliza a linguagem *SysML* e *Stereotype-based Management of Variability* (SMarty) para especificação e representação de variabilidades. SyMPLES-SMarty foi proposta para apoiar o desenvolvimento de sistemas embarcados utilizando conceitos de Linha de Produto Software (LPS). SMarty é uma abordagem para gerenciamento de variabilidades em LPS baseada em *UML*. Esta dissertação propõe uma abordagem alternativa em que as variabilidades são gerenciadas e representadas pela *Common Variability Language* (CVL). CVL é uma linguagem para gerenciar variabilidades sobre modelos especificados usando um metamodelo baseado em *Meta-Object Facility* (*MOF*), tais como *UML* e *SysML*. A principal diferença entre SMarty e CVL é: SMarty é caracterizada como uma abordagem anotativa, enquanto CVL é caracterizada como composicional. Além disso, a CVL pode ser combinada com outras linguagens de modelagem, além da *UML*, tornando-a independente de Linguagens Específicas de Domínio (DSL). A avaliação da substituição proposta foi realizada por meio de um estudo experimental no qual participantes responderam um formulário de coleta de dados, a fim de encontrar a efetividade das abordagens (SyMPLES-SMarty e SyMPLES-CVL) e oferecer evidências sobre vantagens e limitações de cada abordagem comparada. Além disso, uma avaliação qualitativa também foi realizada para discutir e comparar características das abordagens SyMPLES-SMarty e SyMPLES-CVL. As principais contribuições desta dissertação são: (a) oferecer uma abordagem que gerencia e representa variabilidades com a linguagem CVL, e (b) oferecer resultados das comparações entre SyMPLES-SMarty e SyMPLES-CVL.

Palavras-chave: SyMPLES, SMarty, CVL, Linha de Produto de Software, Gerenciamento de Variabilidades.

Variabily Management with CVL in SyMPLES approach

ABSTRACT

The SysML-based Product Line Approach for Embedded Systems (SyMPLES-SMarty) approach is based on the SysML language and the Stereotype-based Management of Variability (SMarty) approach for management to specification and representation of variability. SyMPLES-SMarty has been proposed to support the development of embedded systems using concepts of Software Product Line (SPL). SMarty is an approach for managing variability in SPL based on UML. This dissertation proposes an alternative approach where variability is managed and represented by the Common Variability Language (CVL). CVL is a language for managing variability on models specified using a metamodel based on Meta-Object Facility (MOF), such as UML and SysML. The main difference between SMarty and CVL is: SMarty is characterized as a annotative approach, while CVL is characterized as compositional. Moreover, CVL can be combined with other modeling languages, in addition to UML, making it independent of Domain Specific Languages (DSL). The evaluation of the proposed alternative was carried out in an experimental study in which participants answered a data collection form, in order to find the effectiveness of the approaches (SyMPLES-SMarty and SyMPLES-CVL) and provide evidence about benefits and limitations of each approach compared. In addition, a qualitative evaluation is also done to discuss and compare characteristics of SyMPLES-SMarty and SyMPLES-CVL approaches. The main contributions of this dissertation are: (a) provide an approach that manages and represents variability with CVL language, and (b) provide results of comparisons between SyMPLES-SMarty and SyMPLES-CVL.

Keywords: SyMPLES, SMarty, CVL, Software Product Line, Variability Management.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ABS *Anti-lock braking system*

ATL *Atlas Transformation Language*

CASE *Computer-Aided Software Engineering*

CVL *Common Language Variability*

DSL Linguagens Específicas de Domínio

EMF *Eclipse Modeling Framework*

ES Engenharia de Software

GMF *Graphical Modeling Framework*

GPS *Global Positioning System*

GQM *Goal/Question/Metric*

IDE *Integrated Development Environment*

LP Linha de Produto

LPS Linha de Produto de Software

MDE *Model Driven Engineering*

MOF *Meta-Object Facility*

OCL *Object Constraint Language*

OMG *Object Management Group*

QVT *Meta Object Facility (MOF) 2.0 Query/View/Transformation*

ROI Retorno Sobre Investimento

SEI *Software Engineering Institute*

SMarty *Stereotype-based Management of Variability*

SPL *Software Product Line*

SyMPLES *SysML-based Product Line Approach for Embedded Systems*

SyMPLES-ProfileFB *SyMPLES Profile for Functional Blocks*

SyMPLES-ProfileVar *SyMPLES Profile for Representation of Variability*

SyMPLES-ProcessPL *SyMPLES Process for Product Lines*

SyMPLES-ProcessVar *SyMPLES Process for Identification of Variabilities*

SyMPLES-ProcessVarCvl *SyMPLES Process for Identification of Variabilities in CVL*

SysML *Systems Modeling Language*

TCL *Train Control Language*

TCLE Termo de Consentimento Livre e Esclarecido

UEM Universidade Estadual de Maringá

UML *Unified Modeling Language*

UNIPAR Universidade Paranaense

VANT Veículo Aéreo Não Tripulado

VClassifier *Variability Classifiers*

VSpec Especificação de Variabilidade

SUMÁRIO

1

Introdução

Observando a essência dos mais variados equipamentos eletrônicos, desde fornos de micro-ondas a controladores de freios *ABS*, pode-se notar que a maioria deles possui sistemas embarcados. Também chamado de sistema embutido, o computador deste tipo de sistema é completamente encapsulado e dedicado ao dispositivo que controla, e o software que eles executam são conhecidos como *software* embarcado (Lee e Seshia, 2011).

Segundo Berger (2002), diferentemente dos sistemas tradicionais, os sistemas embarcados possuem características como restrições de tempo, restrições de consumo de energia, restrições de desempenho e são dedicados a tarefas específicas. Essas características fazem com que os atributos de qualidade sejam críticos, principalmente os relacionados aos aspectos temporais, pois dependendo da aplicação podem colocar vidas humanas em risco (Wong et al., 2010).

O uso de sistemas embarcados está crescendo exponencialmente. Atualmente 90% dos dispositivos computacionais se encontram em sistemas embarcados e não em computadores pessoais (3TU.Federatie, 2015). O principal motivador desse crescimento é a competição entre os fabricantes para oferecer uma maior variedade de opções e personalizações aos clientes, o que tem levado à uma diminuição no ciclo de vida de alguns produtos que incluem os sistemas embarcados. Esse fato é constatado e impulsionado pela demanda cada vez maior de consumo das pessoas, que tem levado a uma necessidade por parte da indústria de inovar nos lançamentos de forma mais ágil para se manter competitiva.

Na Engenharia de Software, alguns paradigmas podem ser utilizados para auxiliar o processo de desenvolvimento de sistemas embarcados, como Engenharia Guiada por Modelos (*Model Driven Engineering* - MDE) e Linha de Produto de Software (LPS). MDE apoia a geração de aplicações por meio da transformação de modelos, que podem

estar no mesmo ou em diferentes níveis de abstração (Czarnecki et al., 2005), enquanto LPS permite o reuso sistemático de artefatos comuns derivados de um mesmo domínio para a construção de aplicações individuais (van der Linden et al., 2007).

Uma Linha de Produto de Software é um grupo de sistemas que dividem um conjunto comum de características satisfazendo as necessidades específicas de um segmento de mercado e que são desenvolvidos a partir de um conjunto comum de recursos. A adoção de LPS traz benefícios em diversos aspectos do processo de desenvolvimento de software. Uma nova aplicação consiste de componentes que já foram utilizados e testados em outros produtos, isso faz com que o número de falhas encontrados em uma aplicação seja menor do que a quantidade de falhas encontradas em produtos desenvolvidos individualmente (van der Linden et al., 2007). Vários relatos de experiência indicam que a abordagem de LPS é uma opção satisfatória de reuso sistemático, conforme os benefícios reportados pelas empresas que dela utilizam, conforme o *Hall of Fame* do SEI (2014). Tida como uma atividade crucial para a abordagem de LPS, o gerenciamento de variabilidades é alvo de diversos estudos que objetivam apresentar novas propostas de abordagens que facilitem o desenvolvimento das atividades de LPS na indústria (Chen et al., 2009).

A abordagem *SysML-based Product Line Approach for Embedded Systems* (SyMPLES) (Silva, 2012; Silva et al., 2013) foi proposta para facilitar o gerenciamento de variabilidades em LPS e apoiar o desenvolvimento de sistemas embarcados a partir de modelos de alto nível. SyMPLES é baseada em SysML, pois esta linguagem permite expressar conceitos importantes para a engenharia de sistemas que não podem ser diretamente representados em linguagens de propósito geral como a UML. A abordagem SyMPLES utiliza *Stereotype-based Management of Variability* (SMarty), uma abordagem com características anotativas, para a identificação e representação de variabilidades em modelos SysML. A abordagem SMarty (OliveiraJr et al., 2010) permite o gerenciamento de variabilidade por meio de um perfil UML e de um processo composto por atividades e diretrizes bem definidas para identificar, delimitar e representar variabilidades em modelos UML de uma LPS utilizando os estereótipos propostos em seu perfil (Marcolino, 2014).

Apesar da consolidação da abordagem SMarty nos últimos anos, com sua adoção por diversos projetos/trabalhos, como em Contieri Junior et al. (2011), Oizumi et al. (2012), Fragal (2013), OliveiraJr et al. (2013b), (Rodrigues et al., 2014), (Scheidt et al., 2014), (Falvo Junior et al., 2014) e da sua avaliação experimental (Marcolino, 2014), ela apresenta dois principais problemas: (a) é dependente do metamodelo da UML e (b) a representação do estereótipo «variability» é feita por meio de notas, o que necessita de esforço adicional por parte das ferramentas.

A *Common Variability Language* (CVL) (Haugen, 2012) possui características composicionais e é uma linguagem independente de domínio para especificar e resolver variabilidades. Ela apoia a especificação e resolução de variabilidade em qualquer linguagem definida usando um metamodelo baseado em *Meta-Object Facility* (*MOF*). Desta maneira, a CVL pode ser combinada com outras linguagens de modelagem existentes no mercado, como a UML e SysML. Isso torna a CVL uma linguagem genérica baseada em MOF, que pode ser aplicada em contextos diferentes (Haugen, 2012). Isto porque a CVL não adiciona anotações ou conceitos de variabilidade aos modelos originais da LPS, mantendo-os livres de alterações.

Neste contexto, este trabalho de mestrado visa propor uma versão da abordagem SyMPLES que utiliza a linguagem CVL para o gerenciamento de variabilidades, em vez da abordagem SMarty. Os objetivos específicos são: (a) adaptar a abordagem SyMPLES para o uso de CVL; (b) realizar estudos experimentais como um meio de avaliar a substituição proposta; e, (c) fornecer evidências sobre as vantagens e limitações do uso de SMarty e CVL a partir dos estudos realizados.

Esta dissertação apresenta mais cinco capítulos. O Capítulo 2 apresenta uma revisão bibliográfica com fundamentação teórica sobre LPS, a abordagem *SyMPLES*, a abordagem SMarty e a Linguagem CVL. No Capítulo 3 é descrita a abordagem SyMPLES-CVL, proposta por este trabalho. No Capítulo 4 é apresentado os resultados de um estudo experimental com o objetivo de avaliar e comparar as abordagens SyMPLES-SMarty e SyMPLES-CVL. No Capítulo 5 é apresentado uma avaliação qualitativa comparando as abordagens SyMPLES-SMarty e SyMPLES-CVL através de critérios que enfatizam as diferenças e complementando os resultados do Capítulo 4. Por fim, no Capítulo 6 tem-se a conclusão e trabalhos futuros, seguida das referências bibliográficas.

2

Revisão Teórica

Neste capítulo serão abordados os conceitos de LPS, necessários para compreensão do trabalho, juntamente com pesquisas que serviram de embasamento. Dessa forma, na Seção 2.1 são apresentados os conceitos básicos sobre LPS. Na Seção 2.2 são apresentados os conceitos básicos de Gerenciamento de Variabilidades em LPS. A abordagem SMarty, juntamente com seus conceitos, é apresentada na Seção 2.2.1. Na Seção 2.2.2 são abordados de forma resumida os conceitos sobre a linguagem CVL, bem como suas características básicas e elementos principais. Na Seção 2.3 é realizada a introdução dos conceitos básicos referentes as abordagens anotativas e composicionais. Na Seção 2.4 é apresentado de forma geral a abordagem SyMPLES. E por fim, na Seção 2.5 são apresentadas as considerações finais.

2.1 Linha de Produto de Software

A adoção de LPS traz benefícios em diversos aspectos do processo de desenvolvimento de software. Uma nova aplicação consiste de componentes que já foram utilizados e testados em outros produtos, isso faz com que o número de defeitos encontrados em uma aplicação seja menor do que a quantidade de falhas encontradas em produtos desenvolvidos individualmente (van der Linden et al., 2007).

De acordo com van der Linden et al. (2007), uma LPS é organizada em duas etapas: a engenharia de domínio e a engenharia de aplicação. Na engenharia de domínio (desenvolvimento com reuso) é criada uma infraestrutura de referência para o desenvolvimento de uma família de produtos. A infraestrutura de uma LPS engloba os

artefatos relevantes ao longo do ciclo de desenvolvimento de software, incluindo requisitos, definição da arquitetura, implementação e teste. A principal distinção entre LPS e outras abordagens de reuso, é que seus artefatos possuem uma especificação de variabilidade. Na engenharia de aplicação (desenvolvimento com reuso), os produtos definidos pela linha de produto são efetivamente construídos. A engenharia de aplicação é guiada pela infraestrutura da LPS que possui grande parte da funcionalidade requerida para um novo produto.

2.2 Gerenciamento de Variabilidades

Dentre os principais aspectos que devem ser gerenciados em uma LPS, estão as diferenças entre seus produtos, conhecidas como variabilidade (OliveiraJr et al., 2010). Variabilidade é o termo chave da abordagem de LPS e de acordo com Bosch (2004): "Variabilidade é a habilidade de um software ou artefato ser alterado, adaptado ou configurado para uso em um contexto particular". Em vez de especificar cada sistema individualmente, uma LPS considera a variação existente entre os sistemas individuais e assim especifica uma família de produtos. As variabilidades são explicitamente definidas, representadas, implementadas e evoluídas. Assim, precisa ser gerenciada ao longo do ciclo de vida da LPS.

Para representar a variabilidade, diversas abordagens vêm sendo propostas ao longo dos anos (Chen et al., 2009; Galster et al., 2014). As formas de representação mais comuns descrevem a variabilidade com base em uma notação gráfica, cujos principais elementos são mostrados na Figura - 2.1. O significado dos elementos desta notação é descrito a seguir (van der Linden et al., 2007; Pohl et al., 2005):

- Ponto de variação (*variation point*): descreve o ponto onde existem as diferenças (variações) nos produtos individuais. De acordo com Jacobson et al. (1997), "um ponto de variação identifica um ou mais locais nos quais a variação pode ocorrer". Dessa forma, o ponto de variação pode ser representado em artefatos genéricos e em diferentes níveis de abstração. Basicamente, o ponto de variação responde a pergunta: O que varia em uma LPS?

- Variante (*Variant*): representa os possíveis elementos por meio dos quais um ponto de variação pode ser resolvido. Basicamente, uma variante responde a pergunta: Como uma variabilidade ou ponto de variação varia em uma LPS?

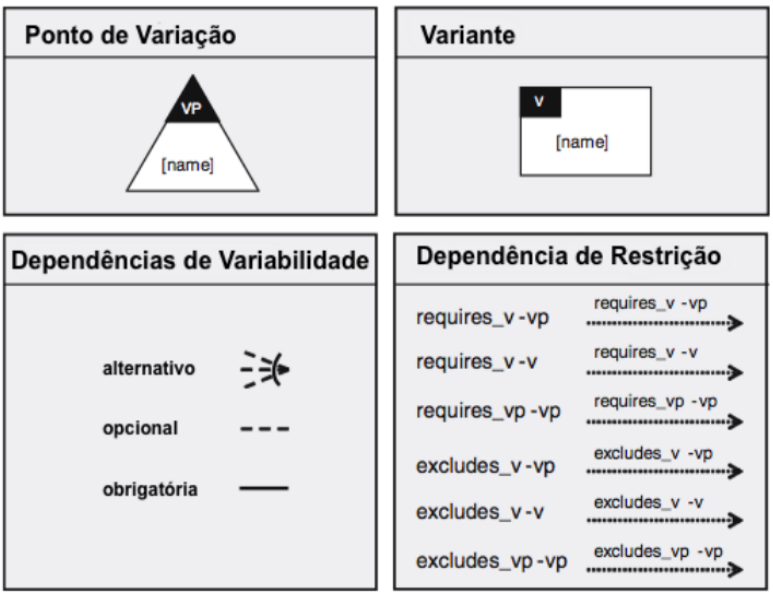

Figura 2.1: Notação gráfica para os modelos de variabilidade (van der Linden et al., 2007)

- Dependências de variabilidade (*Variability dependencies*): são utilizadas como base para definir as diferentes escolhas possíveis (variantes) para resolver um ponto de variação. A notação pode incluir a cardinalidade que determina quantas variantes podem ser selecionadas simultaneamente em um ponto de variação.

- Restrições de dependência (*Constraint dependencies*): descrevem as dependências que podem existir no momento de selecionar as variantes de determinado ponto de variação. Apresentam-se, basicamente, de duas formas: *requires*, quando a seleção de uma variante específica requer a seleção de outra variante; e *excludes*, quando a seleção de uma variante específica proíbe a seleção de outra variante.

O modelo de variabilidades por si só não pode representar todo o significado da variabilidade em uma LPS. Portanto, é necessária a visão tradicional da representação de requisitos, projeto, casos de teste, etc., para que possa ser estabelecida a relação entre estes artefatos e o modelo de variabilidades. Um exemplo deste relacionamento pode ser visualizado na Figura - 2.2, que mostra a relação entre o modelo de variabilidades e o modelo de classes (van der Linden et al., 2007).

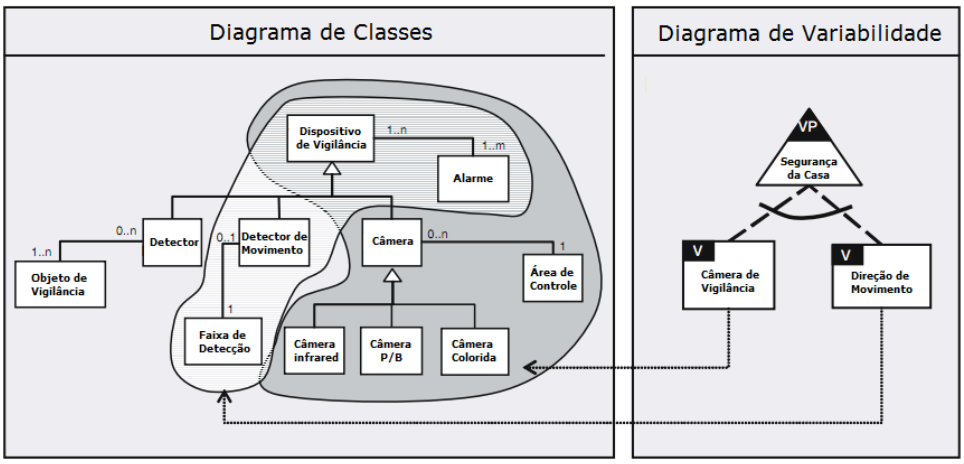

Figura 2.2: Relação entre um modelo de variabilidades e um modelo de classes (van der Linden et al., 2007)

2.2.1 A abordagem SMarty

A maioria das abordagens para gerenciamento de variabilidades em LPS representa seus modelos por meio de diagramas estruturais e comportamentais de Linguagens Específicas de Domínio (DSL), como por exemplo a UML. Isso faz com que seja necessário utilizar um modelo próprio das abordagens para representar as variabilidades, como é o caso da UML, que não define um mecanismo específico para expressar detalhadamente pontos de variação e variantes em uma arquitetura de *software*. Nesse contexto, os dois modelos, UML e de variabilidades, precisam estar relacionados dentro da infraestrutura de LPS para que seja possível derivar determinado produto, como pode ser visto na Figura - 2.2. A abordagem SMarty (OliveiraJr et al., 2010) permite expressar os conceitos de variabilidade em modelos UML, bem como identificar e delimitar tais variabilidades por meio de estereótipos UML. Ela é composta por um processo, o *SMartyProcess*, e um perfil UML 2, o *SMartyProfile*. *SMartyProcess* é um processo sistemático que guia o usuário na identificação, delimitação, representação, rastreamento das variabilidades e análise da configuração de produtos de uma LPS. O processo é apoiado por um conjunto de diretrizes que guiam o usuário na aplicação do *SMartyProfile* para representar variabilidades. *SMartyProfile* contém um conjunto de estereótipos e meta-atributos. Utiliza a notação UML e um mecanismo de *profiling*, para fornecer uma extensão do metamodelo padrão da UML que permita a representação gráfica dos conceitos de variabilidade. O *SMartyProfile* representa o relacionamento dos quatro principais conceitos definidos pelo gerenciamento de variabilidades: variabilidade, ponto de variação, variante e restrições de variantes. Embora a variabilidade possa ser especificada em diferentes artefatos e níveis de abstração,

a abordagem SMarty considera somente os modelos UML que resultam das atividades de desenvolvimento da LPS. O *SMartyProfile* é composto dos seguintes estereótipos e respectivos meta-atributos:

- «variability» representa o conceito de Variabilidade e é uma extensão da metaclasse `UML Comment`. Isso significa que este estereótipo pode ser aplicado somente em notas UML. Possui os seguintes meta-atributos:
  - `name`, o nome pelo qual uma variabilidade é referenciada;
  - `minSelection`, representa o número mínimo de variantes a serem selecionadas para resolver um ponto de variação ou variabilidade;
  - `maxSelection`, representa o máximo de variantes a serem selecionadas para resolver um ponto de variação ou variabilidade;
  - `bindingTime`, é o momento no qual uma variabilidade deve ser resolvida. Esse tempo é representado pela classe de enumeração `BindingTime`;
  - `allowsAddingVar`, indica se é possível incluir novas variantes após uma variabilidade ser resolvida;
  - `variants`, representa a coleção de instâncias associada à variabilidade;
  - `realizes`, representa a coleção de variabilidades de modelos de menor nível que realiza a variabilidade;
- «variationPoint» representa o conceito de Ponto de Variação e é uma extensão das metaclasses `UML Actor`, `UseCase`, `Interface` e `Class`. Isso significa que este estereótipo pode ser aplicado somente a atores, casos de uso, interfaces e classes. Este estereótipo possui os seguintes meta-atributos:
  - `numberOfVariants`, indica o número de variantes associadas que podem ser selecionadas para resolver este ponto de variação;
  - `bindingTime`, o momento no qual um ponto de variação deve ser resolvido. Esse tempo é representado pela classe de enumeração *BindingTime*;
  - `variants`, representa a coleção de instâncias das variantes associadas a este ponto de variação;
  - `variabilities`, representa a coleção de variabilidades com as quais esse ponto de variação está associado;

- «variant» representa o conceito Variante e é uma extensão abstrata das metaclasses `UML Actor`, `UseCase`, `Interface` e `Class`. Por ser abstrato, esse estereótipo não pode ser aplicado em nenhum elemento UML, porém, suas especializações não abstratas podem ser aplicadas em atores, casos de uso, interfaces e classes. Esse estereótipo é especializado em outros quatro estereótipos não abstratos, sendo eles: «mandatory», «optional», «alternative_OR» e «alternative_XOR». O estereótipo «variant» possui os seguintes meta-atributos:
  - `rootVP`, representa o ponto de variação ao qual está associado;
  - `variabilities`, que é a coleção de variabilidades com as quais esta variante está associada;
- «mandatory» representa uma variante obrigatória que está presente em todos os produtos de uma LPS;
- «optional» representa uma variante que pode ser escolhida para resolver uma variabilidade ou ponto de variação;
- «alternative_OR» representa uma variante que faz parte de um grupo de variantes inclusivas. Diferentes combinações das variantes inclusivas podem resolver pontos de variação de diferentes maneiras, gerando assim, produtos distintos;
- «alternative_XOR» representa uma variante que faz parte de um grupo de variantes exclusivas. Isso significa que somente uma variante do grupo pode ser selecionada para resolver um ponto de variação;
- «mutex» representa o conceito de restrição "Exclusão Mútua" e é um relacionamento mutuamente exclusivo entre variantes. Significa que para uma variante ser selecionada, a variante relacionada não pode ser selecionada;
- «requires» representa o conceito de restrição "Complemento" e é um relacionamento entre variantes, no qual a variante escolhida requer a escolha da variante relacionada;
- «variable» é uma extensão da metaclasse `UML Component`. Este estereótipo indica que um componente é formado por um conjunto de classes com variabilidades explícitas. Este estereótipo possui o meta-atributo *classSet*, que é a coleção de instâncias das classes variáveis que formam o componente.

2.2.2 A linguagem CVL

Inspirada nos modelos de características (Combemale et al., 2012), a *Common Variability Language* (CVL) é uma linguagem independente de domínio para especificação e resolução de variabilidades sobre instâncias de qualquer linguagem baseada no meta-modelo MOF (OMG, 2013). O propósito geral da linguagem CVL é introduzir variabilidades em modelos existentes sem modificá-los. A CVL tem uma arquitetura dividida em camadas (Bak et al., 2010). Na Figura - 2.3 são representadas as camadas e como elas se relacionam, e são descritas a seguir:

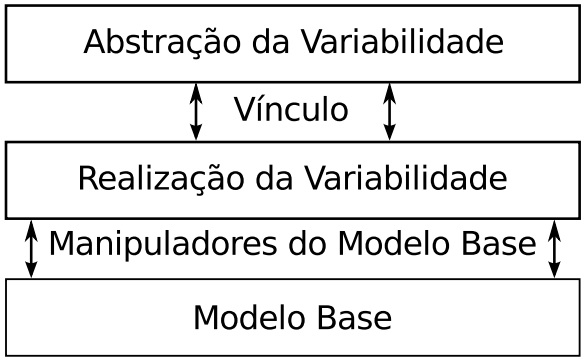

Figura 2.3: Arquitetura em camadas da CVL

- Modelo Base: contém o modelo no qual a CVL introduz variabilidade. A CVL assume que o Modelo Base é uma estrutura gráfica e deve ser uma instância de qualquer metamodelo definido via MOF, tais como as linguagens UML e SysML.
- Manipuladores do Modelo Base: referencia os elementos do Modelo Base. Manipuladores de Modelos Base fornecem uma interface entre a CVL e os Modelos Base.
- Realização da Variabilidade: fornecem construtores que terão um impacto direto no Modelo Base por meio da definição de pontos de variação. Existem 4 tipos de pontos de variação: existência (*existence*), substituição (*substitution*), atribuição de valor (*value assignment*) e caixa preta (*opaque variation point*) que serão vistos com mais detalhes posteriormente na Seção 2.2.2.3.
- Vínculo: conecta as camadas Realização da Variabilidade e Abstração da Variabilidade. É um mapeamento 1:1 que separa o efeito direto que as especificações de variabilidade (VSpec) têm sobre o Modelo Base.
- Abstração da Variabilidade: fornece construtores para especificação e resolução da variabilidade abstrata. A especificação da variabilidade (*VSpecs*) não tem efeito

direto no Modelo Base. Em vez disso, ele apresenta variabilidade em termos de escolhas (*choices*) em decisões binárias e variáveis (*variables*). Este assunto será discutido na Seção 2.2.2.1.

Na Figura - 2.4 temos a CVL combinada com outras linguagens. O Modelo de Variabilidade e os Modelos de Resolução são definidos em CVL, enquanto que o Modelo Base e os Modelos Resolvidos podem ser definidos em qualquer linguagem MOF. Isto é representado na figura com a utilização de cores: os elementos em cor rosa são elementos pertencentes a CVL, e os elementos em cor azul não pertencem.

O Modelo de Variabilidade é definido sobre o Modelo Base, representando e gerenciando as variabilidades do Modelo Base, mas sem adicionar conceitos de variabilidade ou fazer nenhuma alteração sobre o Modelo Base. Este é o motivo no qual a CVL é definida como uma linguagem genérica, pois pode ser aplicada em muitos contextos diferentes.

Os Modelos de Resolução são definidos com um conjunto de valores (resoluções) que serão utilizados para resolver o Modelo de Variabilidade.

O elemento na cor roxa (Execução CVL) representa a o processo de Materialização da CVL, que é responsável pela derivação de produtos. Este processo recebe como entrada o Modelo de Variabilidade, Modelo Base e os Modelos de Resolução. O resultado do processo são os Modelos Resolvidos, a partir dos quais cada Modelo de Resolução resultará em um Modelo Resolvido.

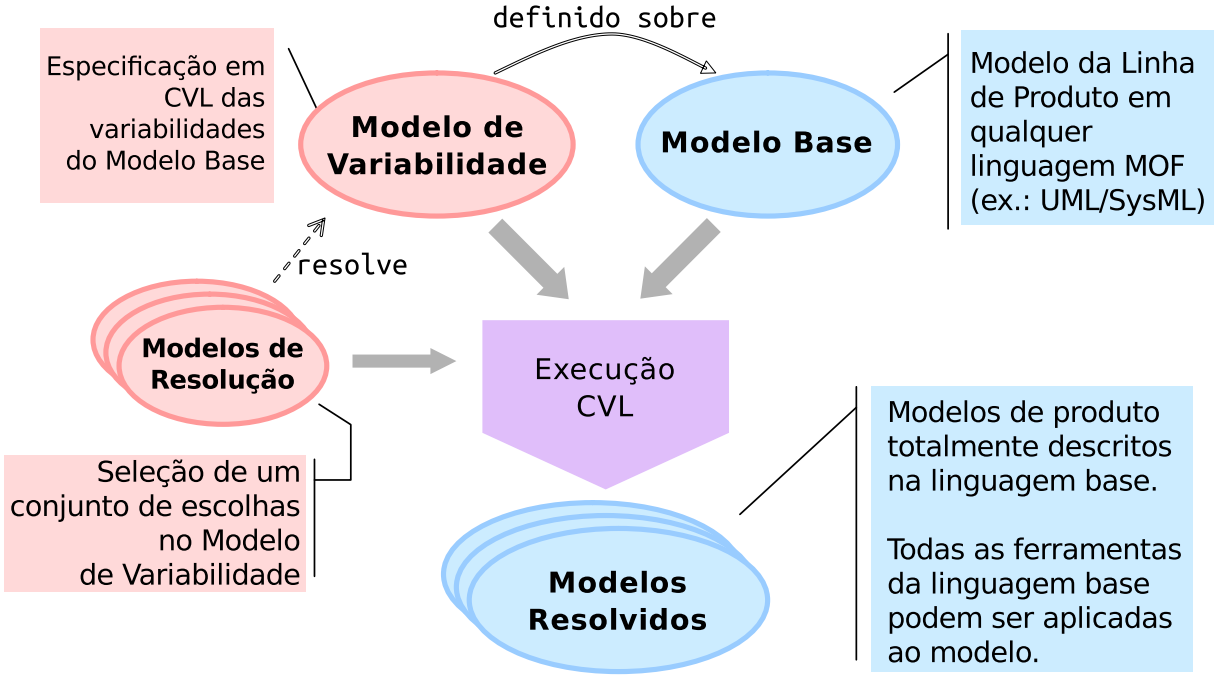

Figura 2.4: CVL combinada com outras linguagens MOF (Haugen, 2012)

A solicitação de proposta também deixa claro que a CVL deve ser executável de tal forma que os modelos resolvidos podem ser produzidos automaticamente a partir do modelo de variabilidade, os modelos de resolução e o Modelo Base. Isto é ilustrado na Figura - 2.4 pelo elemento Execução CVL (Materialização), no qual é um processo de transformação do Modelo Base em Modelo Resolvido pela aplicação dos conceitos de variabilidade contidos no Modelo de Variabilidade juntamente com as resoluções contidas nos Modelos de Resolução.

2.2.2.1 Abstração da Variabilidade

A camada Abstração da Variabilidade fornece construtores para especificação da variabilidade. Ela isola os componentes lógicos da CVL de partes que manipulam o Modelo Base. O principal construtor de variabilidade é a Especificação de Variabilidade (*VSpec*) que pode representar uma escolha binária, um valor de parâmetro, ou uma especificação de elemento que pode ser instanciado várias vezes. *VSpecs* podem ser organizados em árvores, chamadas de Árvores *VSpec*. Restrições podem ser associadas com os nós das árvores *VSpec* para definir restrições sobre resoluções de variabilidade válidas além daquelas implicadas pela árvore. Na Figura - 2.5 temos um exemplo de uma Árvore *VSpec* representando variabilidades de uma impressora. Nesta figura, os *VSpecs* (`Printer`, `HighSpeed` e `EmergencyPower`) estão representados por retângulos com cantos arredondados e a restrição está representada por um paralelogramo.

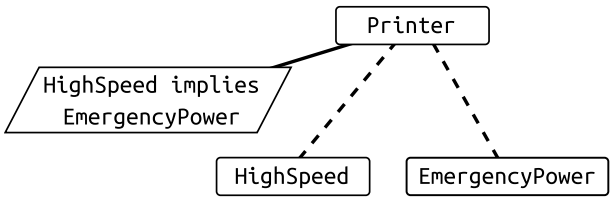

Figura 2.5: Exemplo de Árvore VSpec - Adaptado de Haugen (2012)

Um *VSpec* é uma indicação abstrata de variabilidade no Modelo Base. As particularidades de uma variabilidade como quais elementos do Modelo Base estão envolvidos e como eles são afetados não é especificado, tornando-os abstratos (Haugen, 2012). Em CVL, o efeito dos *VSpecs* sobre o Modelo Base é especificado pela ligação dos Pontos de Variação com os *VSpecs*. Pontos de Variação da linguagem CVL serão discutidos na Seção 2.2.2.3.

2.2.2.1.1 Tipos de VSpec

Podemos destacar três tipos de *VSpecs*: Escolhas (*choices*), Variáveis (*variables*) e Classificadores de Variabilidade (*variability classifiers* ou *VClassifier*).

A Figura - 2.6 mostra uma árvore *VSpec* de uma linha de produto representando um escritório que pode conter uma impressora e um *scanner*. Os nós exibidos como retângulos com cantos arredondados (`Office`, `Printer`, `Scanner`, `GreenMode`, `Type`, `Color`, `BW`, `Speed`, `High`, `Low` e `Turbo`) representam os *VSpecs* do tipo Escolha.

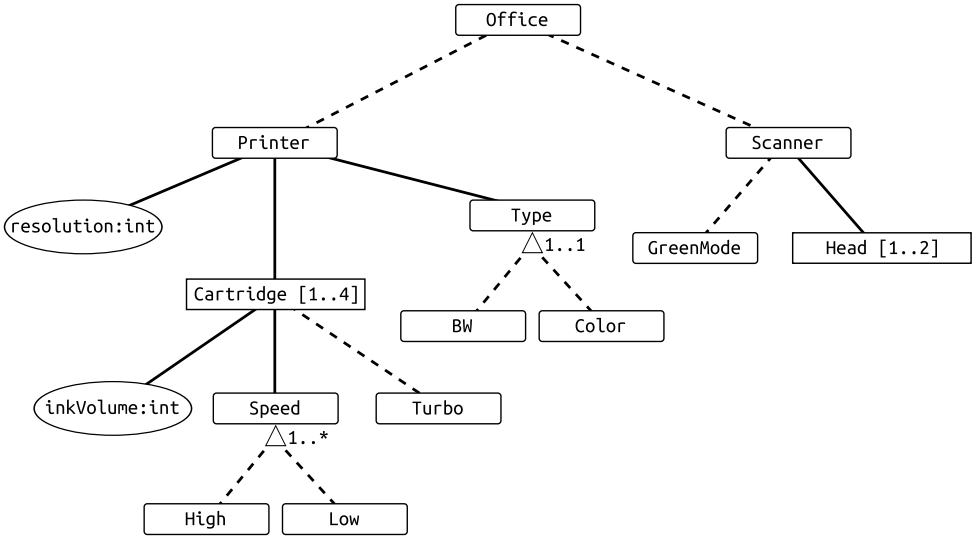

Figura 2.6: Árvore VSpec representando um modelo de variabilidade da CVL - Adaptado de Haugen (2012)

Uma Escolha é um *VSpec* cuja resolução requer uma decisão positiva ou negativa (verdadeiro ou falso). Por exemplo, o fato da existência de uma Escolha chamada `Scanner` na árvore indica que durante o processo de resolução será decidido positivo ou negativo para `Scanner`, e que esta decisão terá algum efeito sobre o Modelo Base, mas o efeito é desconhecido. O efeito pode ser, por exemplo, se um determinado elemento será excluído/mantido ou se uma substituição será feita.

Uma váriavel é um *VSpec* cuja resolução envolve o fornecimento de um valor de um tipo específico. Este valor então será aplicado a algum objeto no Modelo Base. Na Figura - 2.6, as variáveis (`resolution` e `inkVolume`) são representadas por elipses.

O terceiro tipo de *VSpec* é o classificador de variabilidade, abreviado por *VClassifier*. Um *VClassifier* é um tipo de *VSpec* cuja resolução significa a criação de instâncias e então o fornecimento de resoluções para os *VSpecs* de sua sub-árvore. Na Figura - 2.6 existem dois *VClassifiers* (`Cartridge` e `Head`) representados por retângulos. Para cada instância criada de `Cartridge` durante o processo de resolução, deverá ser resolvido `inkVolume`,

`Speed`, `High`, `Low` e `Turbo`. Assim, duas instâncias de `Cartridge` poderão ter resoluções distintas para estes *VSpecs*. Como nas Escolhas e Variáveis, nesta camada o efeito dos *VClassifiers* sobre o Modelo Base também é desconhecido.

Cada *VClassifier* tem uma multiplicidade de instâncias indicando a quantidade de instâncias que devem ser criadas durante a resolução. A multiplicidade de `Cartridge` é `[1..4]`, indicando que no mínimo 1 e no máximo 4 instâncias deverão ser criadas de `Cartridge`.

2.2.2.1.2 Árvores VSpec

VSpecs podem ser organizados em estruturas de árvore, onde a relação pai-filho organiza o processo de resolução através da imposição de estrutura e lógica para resoluções que são permitidas.

Uma sub-árvore sob um nó representa *VSpecs* "subordinados" no sentido que a resolução de um nó impõe certas restrições nas resoluções dos seus nós subordinados. Além das restrições impostas implicitamente pela estrutura de árvore, também é possível especificar restrições explicítas, como temos por exemplo na Figura - 2.5, `HighSpeed` exige/requer `EmergencyPower`. As restrições explicítas da linguagem CVL tem como foco as árvores *VSpec* e serão discutidas na Seção 2.2.2.2.

- **Implicação de resolução negativa** - Uma Escolha de resolução negativa requer uma resolução negativa para os seus *VSpecs* do tipo Escolha e nenhuma resolução para os outros tipos. Por exemplo, na Figura - 2.6, se decidimos "falso" para `Printer` então devemos decidir "falso" para `Type`, não podemos ter nenhuma instância de `Cartridge`, e não podemos definir um valor para `resolution`, e assim por diante recursivamente em direção as folhas da árvore;

- **Implicação de resolução positiva** - Cada Escolha possui um atributo booleano *isImpliedByParent* que quando verdadeiro (*true*) indica que se seu pai (a) é resolvido positivamente ou (b) é a raiz da árvore, então esta escolha deve ser resolvida positivamente (verdadeiro). Uma resolução para um *VSpec* que não seja do tipo Escolha é sempre considerada positiva por definição.

  A regra geral é: (a) se um *VSpec* do tipo Escolha que possui subordinados for resolvido positivamente ou (b) existem resoluções de *VClassifiers*, então suas sub-escolhas com $isImpliedByParent = true$ devem ser resolvidas positivamente, suas sub-variáveis devem receber um valor do tipo apropriado e seus sub-*VClassifiers* devem ser instanciados de acordo com sua multiplicidade de instâncias.

Em árvores *VSpec*, um valor *true* para *isImpliedByParent* é indicado por uma linha sólida ligando um nó filho com seu nó pai. E um valor $isImpliedByParent = false$ é indicado por meio de uma linha tracejada. Para um *VSpec* que não seja do tipo Escolha, a linha é sempre sólida. É importante ressaltar que para um *VSpec* Escolha que esteja posicionada na raiz de uma árvore o valor do atributo *isImpliedByParent* é sempre *true*.

De acordo com as regras acima, na Figura - 2.6, as Escolhas `Office`, `Type`, e `Speed` possuem o valor *true* no atributo *isImpliedByParent* e as outras Escolhas possuem o valor *false*. Em uma possível resolução, se decidirmos *true* para `Printer` então devemos decidir `True` para `Type`, fornecer um valor inteiro para `resolution`, e criar entre 1 e 4 instâncias de `Cartridge`. Para cada instância de `Cartridge` devemos decidir *true* para `Speed` e fornecer um valor inteiro para `inkVolume`. Similarmente, se decidirmos *True* para `Scanner` então devemos criar entre 1 e 2 instâncias de `Head`.

- **Multiplicidade de grupo** - Um *VSpec* pode ter uma multiplicidade de grupo, especificando quantas sub-escolhas devem ser resolvidas positivamente. Nas árvores *VSpec* as multiplicidades de grupos são exibidas como pequenos triângulos localizados abaixo de um *VSpec* Escolha: se `Type` é resolvido positivamente então apenas uma (`[1..1]`) escolha entre `BW` e `Color` deve ser resolvido positivamente. Similarmente, para cada instância de `Cartridge`, `Speed` deve ser resolvido positivamente e então entre 1 e 2 (`[1..2]`) Escolhas `Low` e `High` deve ser resolvida positivamente.

- **Multiplicidade de instância** - Cada *VClassifier* possui uma multiplicidade de instância especificando quantas instâncias deste *VSpec* podem ser criados. Isto é especificado como um intervalo usando dois valores, indicando uma multiplicidade mínima e uma máxima. Na Figura - 2.6, `Cartridge` e `Head` são *VClassifiers*, e possuem multiplicidade de instância `[1..4]` e `[1..2]`, respectivamente.

2.2.2.1.3 Árvores de Resolução VSpec

VSpecs são resolvidos pelos *VSpecs* de Resolução. Para os três tipos de *VSpecs* citados na Seção 2.2.2.1.1, temos três tipos de *VSpecs* de Resolução: (a) resoluções de escolhas resolvem *VSpecs* do tipo Escolha; (b) atribuição de valores resolvem variáveis; e (c) *VInstances* resolvem *VClassifiers*. Cada resolução de *VSpec* resolve exatamente um *VSpec* do tipo apropriado. Uma exceção são os *VClassifiers*, que podem ter várias instâncias, deste modo várias resoluções (*VInstances*).

Na Figura - 2.7 existe um exemplo de Árvore de Resolução *VSpec*, onde cada nó na árvore é uma resolução *VSpec* correspondente a Árvore *VSpec* da Figura - 2.6. Na Figura - 2.7, (a) *Printer = verdadeiro* é uma resolução de Escolha resolvendo `Printer` positivamente; (b) *resolution = 1600* é uma atribuição de valor resolvendo `resolution`; e (c) `c1:Cartridge` e `c2:Cartridge` são instâncias resolvendo `Cartridge`. Observe que existem as resoluções *Low = verdadeiro* e *Low = falso* resolvendo o elemento `Low`, mas pertencem a duas instâncias distintas de `Cartridge`.

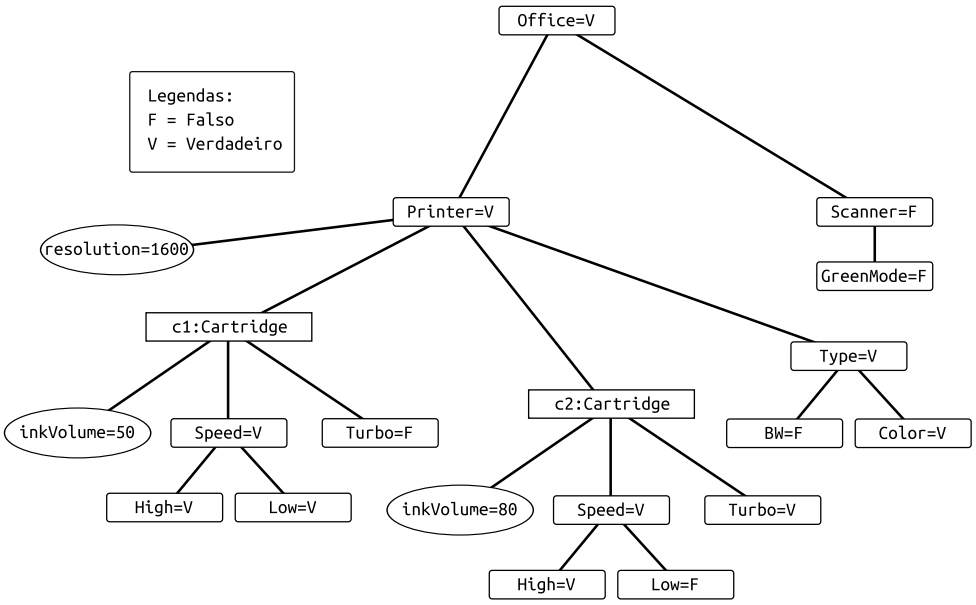

Figura 2.7: Árvore de Resolução VSpec - Adaptado de Haugen (2012)

A árvore de resoluções *VSpecs* devem respeitar as regras citadas na Seção 2.2.2.1.2. Consequentemente, na Figura - 2.7, `Printer` é resolvido positivamente, assim é atribuido o valor inteiro 1600 para `Type`, e criadas duas instâncias de `Cartridge` (`c1` e `c2`), respeitando a multiplicidade de instâncias impostas por `Cartridge`. Como `Scanner` é resolvido negativamente, então `GreenMode` é resolvido negativamente e nenhuma instância de `Head` foi criada, apesar de sua multiplicidade de instância ser `[1..2]`.

2.2.2.2 Restrições

Em CVL, as restrições são usadas para expressar relações entre *VSpecs* que não podem ser diretamente declaradas pelas relações hierárquicas da árvore *VSpec*. As restrições são expressas por meio de um subconjunto da *Object Constraint Language* (OCL) (OMG, 2014b). Além disso, CVL permite o uso de outras linguagens de restrições, incluindo a OCL completa.

As próximas seções apresentam exemplos do uso da linguagem básica de restrições definida pela CVL sobre as árvores *VSpec*. Serão abordadas as restrições proposicionais e aritméticas, as expressões de caminho e a quantificação implícita permita pela CVL.

2.2.2.2.1 Restrições proposicionais

A Figura - 2.8 mostra a especificação de variabilidade de um *pool* de impressões que fornecem um serviço de `fax` e um serviço de impressão (`printer`). Ambos, `fax` e `printer` são opcionais na especificação. Porém o serviço de fax requer o serviço de impressão para imprimir os documentos que forem entregues por fax. Esta dependência é expressa por uma restrição proposicional (no paralelogramo) onde *implies* significa requer/implica. Se existe `fax`, então `printer` deve estar presente.

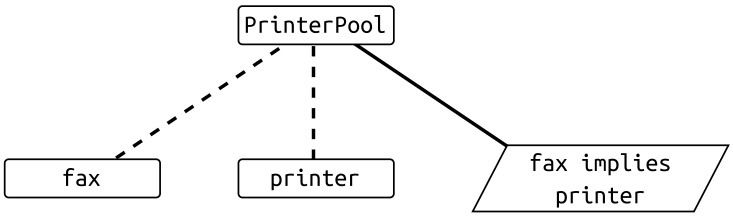

Figura 2.8: Restrição proposicional simples - Adaptado de Haugen (2012)

A Figura - 2.9 mostra uma especificação de variabilidade de um *pool* de impressões que oferece serviços opcionais de `fax`, impressões (`printer`), digitalizações (`scanner`) e cópias (`copier`). Como no último exemplo, `printer` oferece parte da funcionalidade `fax`, imprimindo os documentos que chegam. Além disso, o *pool* de impressões pode copiar documentos se `printer` e `scanner` estiverem presentes. As duas restrições são exibidas no paralelogramo, combinadas pelo operador `and`. A primeira implicação é a mesma da Figura - 2.8 (`fax implies printer`). A segunda especifica que o serviço de cópias requer que `printer` e `scanner` estejam presentes. Como não há ligações entre o paralelogramo e os *VSpecs* a restrição é global. Neste exemplo, ligando a restrição ao `PrinterPool` teriamos o mesmo efeito.

2.2.2.2.2 Restrições aritméticas

As restrições CVL podem envolver expressões para restringir valores de variáveis. A Figura - 2.10 mostra a especificação de variabilidade de um *pool* de impressões com 2 variáveis: velocidade mínima (`minSpeed`) e velocidade padrão de impressão (`speed`). A relação entre `minSpeed` e `speed` é definida por uma restrição: `speed` é igual a `minSpeed` mais uma constante (`300`).

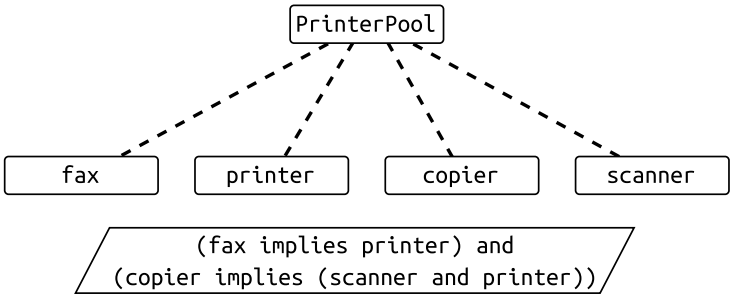

Figura 2.9: Restrição proposicional - Adaptado de Haugen (2012)

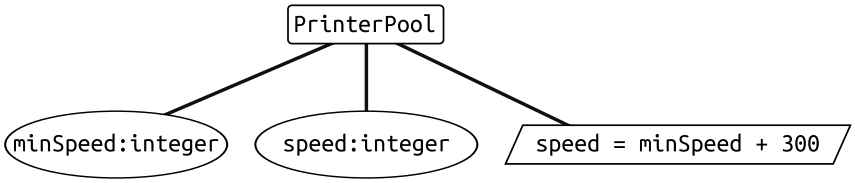

Figura 2.10: Restrição aritmética - Adaptado de Haugen (2012)

2.2.2.2.3 Expressões de caminho

Expressões de caminho são usadas para navegar ao longo da hierarquia da árvore *Vspec*. A mesma restrição exibida na Figura - 2.8 é exibida na Figura - 2.11, `fax` requer `printer`, só que usando nomes completamente qualificados. Expressões de caminho são úteis para remover a ambiguidade entre *Vspecs* com o mesmo nome. Por exemplo, se ambos, `fax` e `printer` tiver uma escolha dependente chamada `color`, poderíamos distingui-los como `fax.color` e `printer.color`.

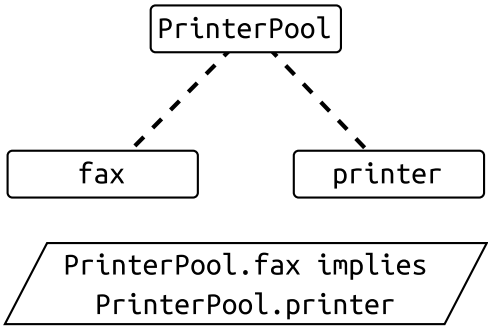

Figura 2.11: Expressão de caminho - Adaptado de Haugen (2012)

2.2.2.2.4 Quantificação implícita

A Figura - 2.12 mostra uma especificação de variabilidade de um *pool* de impressões contendo múltiplas impressoras. No exemplo, uma impressora (`printer`) pode imprimir

documentos em cores (`color`). Se uma impressora é capaz de fazer impressões coloridas, então ela deve conter uma peça para impressões coloridas (`colorPrinterHead`). Esta intenção é expressa por meio de uma restrição proposicional no paralelogramo. A restrição é especificada no contexto do *VClassifier* `Printer`. Embora a restrição diz respeito a um *VClassifier*, não há um quantificador explícito. Assim, está implícito que a restrição é válida para todas as instâncias de `Printer`.

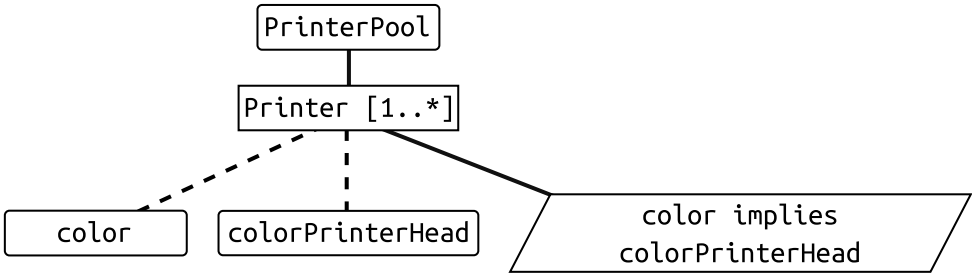

Figura 2.12: Quantificação implícita - Adaptado de Haugen (2012)

2.2.2.3 Realização da Variabilidade

A camada de Realização da Variabilidade fornece construtores para a especificação de pontos de variação sobre o Modelo Base, como mostrado na Figura - 2.3. Um Ponto de Variação CVL é uma modificação aplicada ao Modelo Base durante o processo de transformação do Modelo Base em um Modelo Resolvido, chamado de Materialização. Os Pontos de Variação CVL referem-se a elementos do Modelo Base por meio dos Manipuladores do Modelo Base e também se referem aos *VSpecs* para definir como o Ponto de Variação CVL será resolvida. Estas referências formam uma ligação entre as camadas Abstração da Variabilidade e Realização da Variabilidade. Esse relacionamento é ilustrado na Figura - 2.3.

Existem 4 maneiras dos pontos de variação definir modificações sobre o Modelo Base:

- **Existência** (*existence*) - uma indicação de existência de um objeto, link ou valor no Modelo Base. A existência é um termo genérico para certos tipos de Pontos de Variação CVL cuja aplicação (negativa) implica a exclusão de elementos do Modelo Base. Eles são utilizados para expressar os elementos opcionais no Modelo Base.

- **Substituição** (*substitution*) - uma indicação que um único objeto ou fragmento de um modelo pode ser substituído por outro. **Substituição de objetos** envolve dois objetos e significa redirecionar todas as ligações (*links*) do primeiro objeto para o segundo, e em seguida apagar o primeiro objeto. **Substituição de Fragmento**

envolve a identificação de um *fragmento de colocação* no Modelo Base através de *elementos de fronteira*, criando um "buraco" conceitual para ser preenchido por um *fragmento de substituição* do mesmo tipo;

- **Atribuição de valor** (*value assignment*) - uma indicação que um valor pode ser atribuído a um *slot* específico de um objeto de um Modelo Base; e
- **Caixa Preta** (*opaque variation point*) - um Ponto de Variação CVL cujo impacto sobre o Modelo Base não esta definido na CVL, mas pelo usuário. O impacto é especificado usando linguagens de transformação de modelos, por exemplo a linguagem QVT (OMG, 2015a).

Cada ponto de variação faz referência a um *VSpec* por meio de um vínculo. Um vínculo oferece uma ligação entre a Realização da Variabilidade e a Abstração da Variabilidade. Cada ponto de variação é ligado a um único *VSpec* por meio de um vínculo.

Um exemplo (adaptado de Haugen (2012)) que mostra a variabilidade de um *Scanner* onde aplica-se um *ChoiceVariationPoint*, *SlotAssignmentVariationPoint* e um *SubstitutionVariationPoint* é apresentado na Figura - 2.13. O Modelo Base do `Scanner` é dado em UML usando classes `Feeder`, `Turner`, `Quality`, `BW`, `Color` e `Scanner`. Um *scanner* é descrito por uma estrutura composta das partes `f:Feeder`, `q:Quality` e `t:Turner`.

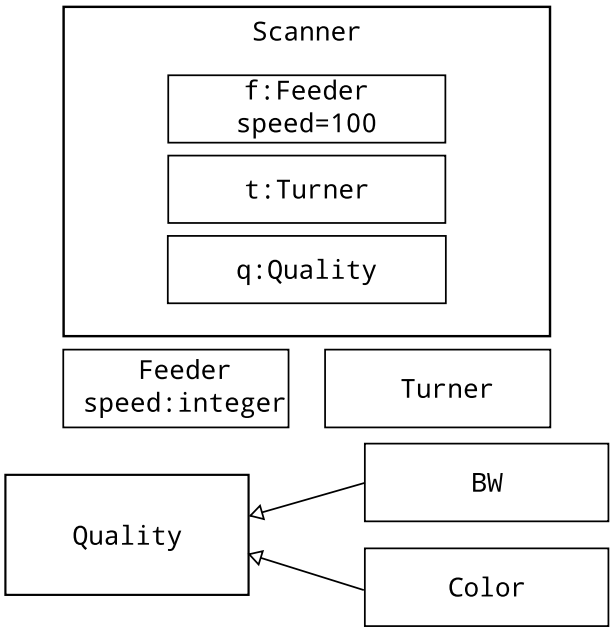

Figura 2.13: Modelo Base da CVL - Diagrama de Classes UML - Adaptado de Haugen (2012)

O *scanner* contém uma parte que faz a leitura do papel (`Feeder`), uma parte que pode virar o papel para ser escaneado em ambos os lados (`Turner`) e uma unidade óptica que determina a qualidade da digitalização (`Quality`). Existem dependências (especializações) das classes `BW` e `Color` para `Quality` indicando que `BW` ou `Color` pode ser utilizado no lugar de `Quality`. `Feeder` contém um atributo (`speed`) que define a velocidade das leituras feitas pelo *scanner*.

O Modelo de Variabilidades do *Scanner* é definido na Figura - 2.14.

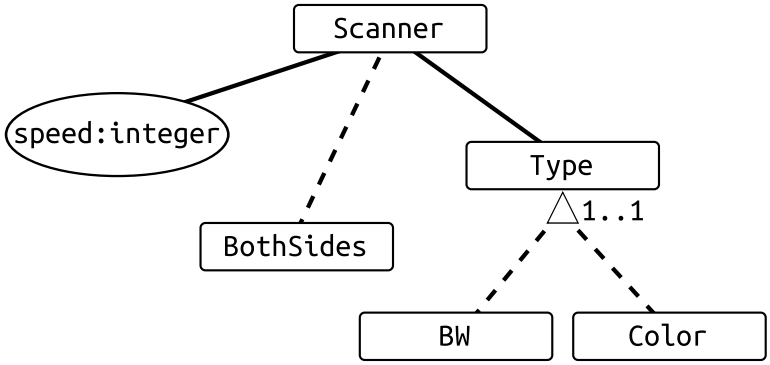

Figura 2.14: Modelo de Variabilidade da CVL - Árvore VSpec - Adaptado de Haugen (2012)

A variabilidade de `Scanner` informa que a velocidade de leitura é dada por um valor inteiro e que é opcional existir a leitura dos dois lados do papel. Além disso, o `Scanner` pode fazer digitalizações em cores ou apenas preto-branco, mas não ambos.

Na Figura - 2.15 ilustramos 3 tipos de pontos de variação:

- O *ParametricSlotAssignment* é ligado à variável `speed` na Árvore *VSpec*. O ponto de variação recebe o valor da resolução de `speed` e o atribui no Modelo Base;
- O *ObjectExistence* esta ligado à Escolha `BothSides`. Ele descreve que quando esta escolha é verdadeira, então `t:Turner` deve estar presente no modelo resolvido, enquanto que quando a escolha de *BothSides* é falsa o `t:Turner` será removido durante a resolução.
- A Escolha `Type` define que o *scanner* deve ser colorido (`Color`) ou preto-branco (`BW`). Quando `Color` é escolhido, é aplicado o *ObjectSubstitution* ligado a `Color`, onde a classe `Quality` é substituido pela classe `Color` no Modelo Base. O efeito da substituição indicada é que qualquer propriedade do tipo `Quality` no Modelo Base será do tipo `Color` no modelo resolvido.

  Quando `BW` é escolhido, do mesmo modo, o *ObjectSubstitution* correspondente ligado à `BW` e referenciando a classe `BW`, irá substituir a classe `Quality` pela classe `BW`.

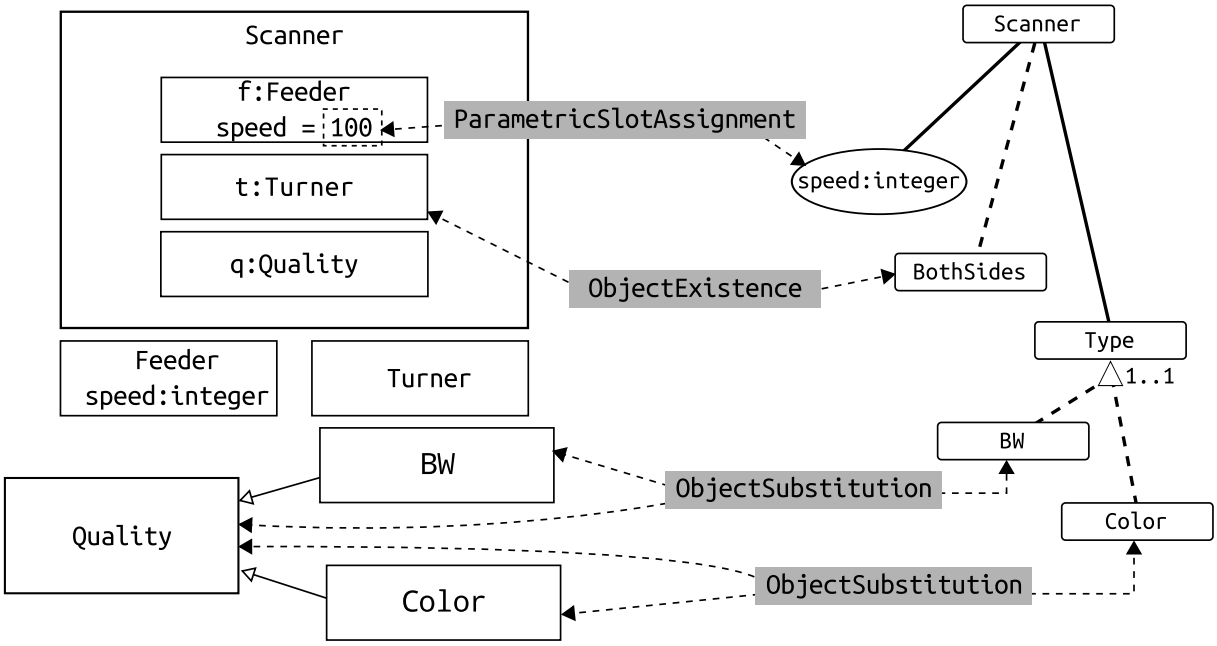

Figura 2.15: Modelo Base e Modelo de Variabilidade com Pontos de Variação - Adaptado de Haugen (2012)

Um possível produto materializado da LPS é exibido na Figura - 2.16. O produto exibido na figura faz digitalizações coloridas em velocidade 200 e não é capaz de virar a página automaticamente para digitalizar os dois lados das folhas.

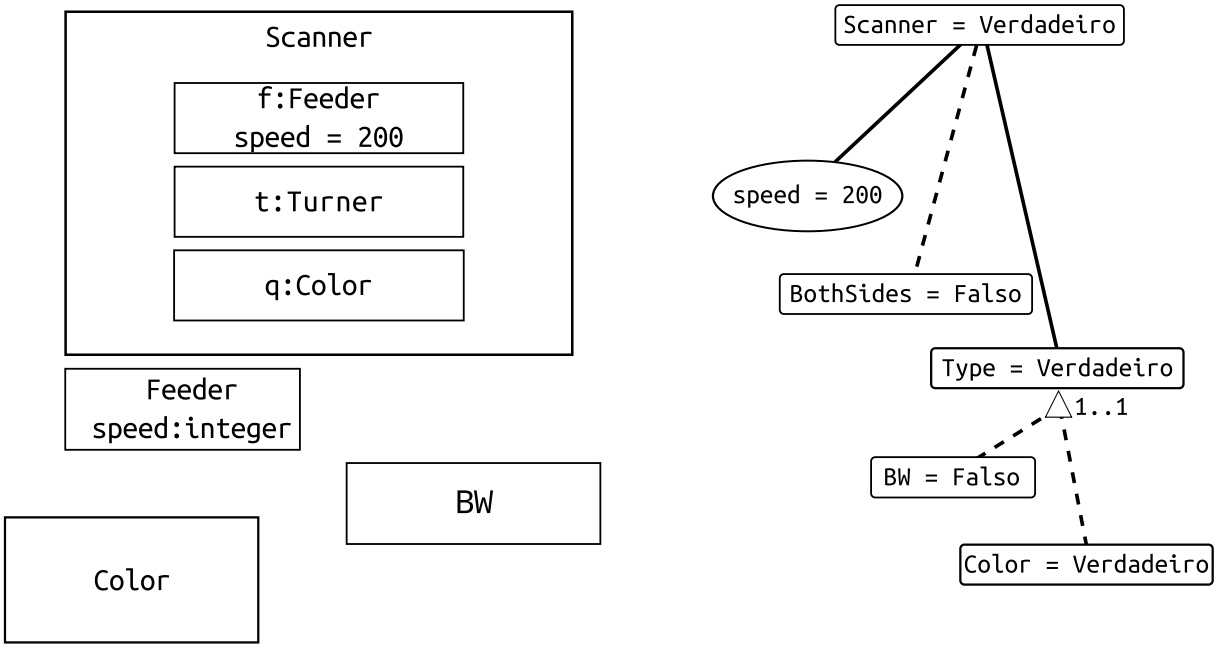

Figura 2.16: Modelo Materializado (esq) e Modelo de Resolução (dir) - Adaptado de Haugen (2012)

2.3 Abordagens Anotativas e Composicionais

Os termos anotativo e composicional evidenciam a natureza de representação de variabilidades das abordagens. Abordagens anotativas fazem anotações para representar as variabilidades, enquanto as composicionais compõem (criam) novos modelos para a representação das variabilidades. De acordo com Kästner et al. (2008), existem várias abordagens para a implementação de LPS, sendo que a maior parte delas pode ser agrupada como anotativas ou como composicionais.

2.3.1 Abordagens Anotativas

As abordagens anotativas representam os conceitos de variabilidades em forma de anotações em modelos de alto nível ou em forma de fragmentos em códigos fonte da LPS (Kästner e Apel, 2008). Neste tipo de abordagem, o processo de derivação de produtos deve interpretar e processar estas anotações afim de gerar um novo produto a partir das características selecionadas.

A abordagem SMarty é caracterizada como uma abordagem anotativa, pois adiciona estereótipos (anotações) do perfil *SMartyProfile* aos modelos UML/SysML, criando um mapeamento entre os conceitos de gerenciamento de variabilidades e os elementos dos modelos.

Na Figura - 2.17 temos um exemplo da aplicação da abordagem anotativa SMarty. Os elementos `Gerenciar Acelerometro A1` e `Gerenciar Acelerometro A2` estão anotados com o estereótipo «alternative_OR», indicando que estes elementos fazem parte de um grupo de variantes inclusivas. O elemento `Configurar_Acelerometro` por sua vez está anotado com o estereótipo «mandatory», indicando que este elemento é obrigatório.

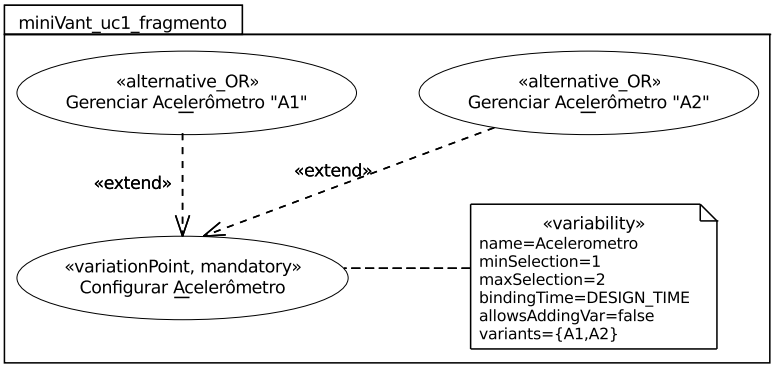

Figura 2.17: Fragmento de LPS *mini-VANT* com variabilidades representadas com SMarty

2.3.2 Abordagens Composicionais

Segundo Kästner e Apel (2008), as abordagens composicionais implementam características em unidades distintas (fisicamente separadas). Sendo assim, para modelos SysML, a representação de variabilidades é feita em modelos separados do modelo SysML. Durante o processo de derivação de instâncias (ou produtos) da LPS, é necessário que tanto o modelo com os elementos da LPS quanto o modelo que contém os conceitos de variabilidades sejam interpretados e processados.

A linguagem CVL pertence ao grupo das abordagens composicionais, pois não modifica os modelos UML/SysML e compõe novos modelos para representar as variabilidades da LPS.

A Figura - 2.18 exibe um exemplo da aplicação da linguagem CVL. Os elementos `Gerenciar Acelerometro A1` e `Gerenciar Acelerometro A2` estão ligados aos Pontos de Variação CVL do tipo Existência, que por sua vez estão ligados aos VSpecs `Acelerometro1` e `Acelerometro2`. O Diagrama de Caso de Uso é definido como um Modelo Base da CVL, enquanto que na parte superior da figura encontra-se o Modelo de Variabilidades da CVL.

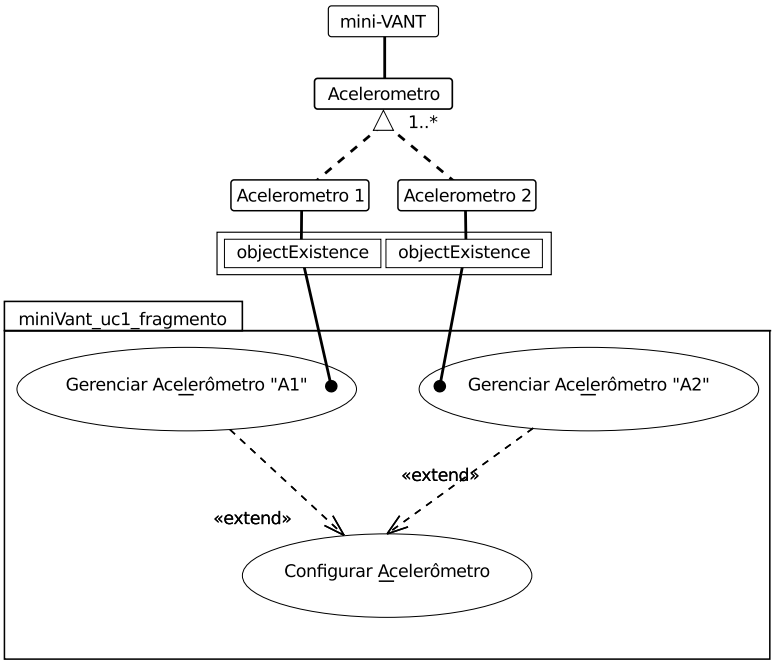

Figura 2.18: Fragmento de LPS *mini-VANT* com variabilidades representadas com SMarty

2.4 A abordagem SyMPLES-SMarty

A abordagem SyMPLES-SMarty, originalmente chamada de SyMPLES por Silva (2012), tem como base o método *OOSEM* (Lykins et al., 2000) usado na engenharia de sistemas e utiliza SMarty (OliveiraJr et al., 2010) para gerenciamento de variabilidades. SyMPLES reúne duas técnicas que têm sido utilizadas para lidar com a crescente complexidade da atual geração de sistemas embarcados: LPS e os conceitos de orientação a objetos aplicados na construção de modelos de alto nível. SyMPLES é composta de dois perfis e dois processos. Os dois perfis do SyMPLES são extensões da linguagem SysML, e são criadas por meio do mecanismo de *profiling*. Eles são demonstrados a seguir:

- *SyMPLES Profile for Functional Blocks* (SyMPLES-ProfileFB) é um perfil para blocos funcionais, formado por um grupo de estereótipos cujo objetivo é fornecer uma semântica adicional aos blocos SysML.
- *SyMPLES Profile for Representation of Variability* (SyMPLES-ProfileVar) é um perfil para representação de variabilidades, baseado no perfil UML definido na abordagem SMarty;

Cada um dos processos do SyMPLES é formado por um grupo de atividades e um conjunto de diretrizes para cada atividade que sistematiza a execução do processo conforme descrito a seguir:

- *SyMPLES Process for Product Lines* (SyMPLES-ProcessPL) é um processo de desenvolvimento de LPS que define um conjunto de atividades genéricas, independente de ferramentas, que auxiliam a criação dos artefatos de uma LPS para sistemas embarcados baseada em SysML;
- *SyMPLES Process for Identification of Variabilities* (SyMPLES-ProcessVar) é um processo para representação das variabilidades de uma LPS, baseado no processo definido na abordagem SMarty. O SyMPLES-ProcessVar é executado paralelamente ao processo SyMPLES-ProcessPL; seu objetivo é auxiliar o usuário na identificação, delimitação e representação das variabilidades, além da configuração de produtos de uma LPS para sistemas embarcados baseada em SysML;

A abordagem SyMPLES tem como objetivo definir uma LPS para sistemas embarcados que permita a geração de produtos a partir de modelos. Com isso, é possível expressar os conceitos relacionados à estrutura, comportamento, definição e resolução de variabilidades em SysML (Fragal, 2013).

A seguir são exibidos com detalhes o SyMPLES-ProfileFB e o SyMPLES-Process-PL. Ambos são de grande importância pois fazem parte da versão alternativa do SyMPLES-SMarty, proposta desta dissertação.

2.4.1 O perfil SyMPLES-ProfileFB

Nesta seção é descrito o perfil SysML de blocos funcionais SyMPLES-ProfileFB. Contudo, é necessária a introdução de alguns conceitos sobre a relação entre a abordagem de blocos funcionais e a abordagem orientada a objetos, representada pela linguagem SysML, na modelagem de sistemas embarcados. Para ilustrar a utilização do SyMPLES-ProfileFB, será usado como exemplo o modelo da LPS para o bloco de Sensores do *mini-VANT* definido por (Buschmann et al., 2004).

2.4.1.1 Relação entre blocos funcionais e SysML

Nas disciplinas de engenharia, as linguagens para implementação de sistemas frequentemente são baseadas em blocos funcionais (BFs), como por exemplo: as linguagens para controladores programáveis (LCPs) e *Matlab/Simulink* (Mathworks, 2007). Ferramentas disponíveis na indústria para este tipo de aplicação também utilizam blocos funcionais como unidade de organização. Apesar de algumas diferenças, o conceito de bloco funcional é o mesmo em grande parte destas linguagens e ferramentas (Heverhagen, 2003).

Os elementos básicos de uma linguagem de blocos funcionais são: blocos, portas e linhas. Um bloco representa uma entidade; blocos podem conter outros blocos, representando o conceito de subsistema. Uma linha liga dois ou mais blocos, conectando a porta de um bloco de origem a uma ou mais portas no bloco destino. As linhas representam a interação das entidades do modelo em termos de transmissão de dados e controle de fluxo (Sjostedt et al., 2008). A Figura - 2.19 mostra um exemplo de sistema que representa um loop simples formado por oito blocos conectados por linhas, utilizando blocos funcionais da linguagem *Matlab/Simulink*.

Nas linguagens de blocos funcionais, o comportamento de um bloco é pré-definido. Por exemplo: na Figura - 2.19, os blocos `G1` e `G2` são do tipo `Ganho`. A saída de um bloco desse tipo é igual à entrada multiplicada por um parâmetro definido pelo desenvolvedor, no caso desse exemplo, `G1` tem um ganho de `3x` e o `G2` de `5x`.

Apesar da popularidade das linguagens de blocos funcionais na engenharia de sistemas, tal abordagem não permite expressar os requisitos do sistema, pois a modelagem se inicia com a construção da solução para o problema que está sendo considerado. Por outro lado, o uso de tecnologias orientadas a objeto é amplamente aceito neste domínio (Heverhagen,

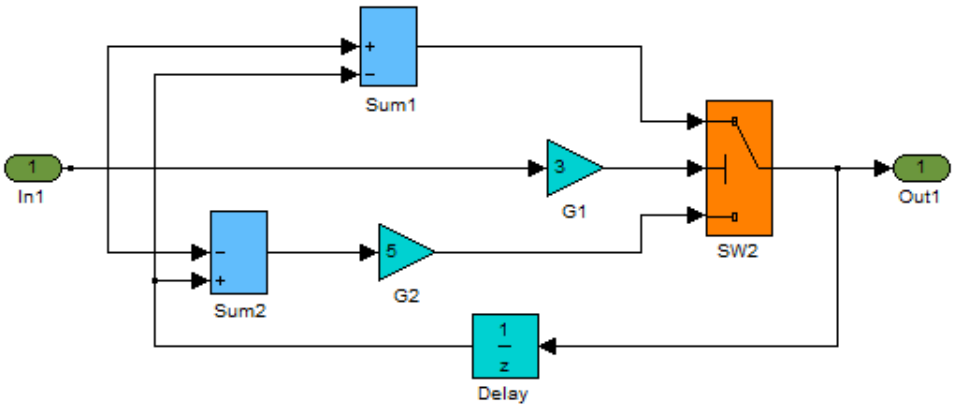

Figura 2.19: Exemplo de sistema como um modelo Simulink (Mathworks, 2007)

2003). Linguagens como SysML são utilizadas para criar modelos nas fases de requisitos, análise, projeto, validação e testes. Neste caso, torna-se vantajoso combinar SysML com os conceitos de uma linguagem orientada a blocos funcionais, pois é possível modelar um sistema desde as fases iniciais do projeto, como especificação de requisitos e análise, até a fase de implementação, por meio da utilização da abordagem de blocos funcionais.

SysML é uma linguagem de propósito geral para a engenharia de sistemas em que um bloco não possui comportamento pré-determinado. Tal linguagem utiliza os diagramas estruturais para descrever o relacionamento entre os elementos do sistema, sendo que os diagramas de Definição de Blocos e Diagrama Interno do Blocos são os mais adequados para representar modelos de blocos funcionais (Sjostedt et al., 2008). O Diagrama Interno de Blocos tem como principal propósito representar os relacionamentos entre os elementos que formam um bloco complexo, e identificar como tais elementos colaboram entre si. Duas sub-partes podem conter portas e estar ligadas por meio de conectores dedicados. Esta estrutura é bastante semelhante aos modelos de blocos funcionais.

Contudo, estabelecer a relação um-para-um entre um bloco funcional e um bloco SysML não é suficiente para especificar um sistema. Diversos parâmetros precisam ser considerados nesse processo de mapeamento. Anotações podem ser necessárias para identificar que um determinado elemento criado na linguagem SysML corresponde a um determinado bloco funcional com comportamento pré-estabelecido. Essa correspondência pode ser realizada com a utilização de um conjunto de estereótipos, utilizados como tags, que podem ser aplicados a blocos SysML, com o objetivo de fornecer informação adicional para determinar uma semântica precisa.

O perfil SyMPLES-ProfileFB contém os estereótipos para permitir o mapeamento entre elementos da linguagem SysML e as principais classes de blocos funcionais. A Figura - 2.20 mostra o perfil com os estereótipos de blocos funcionais criados.

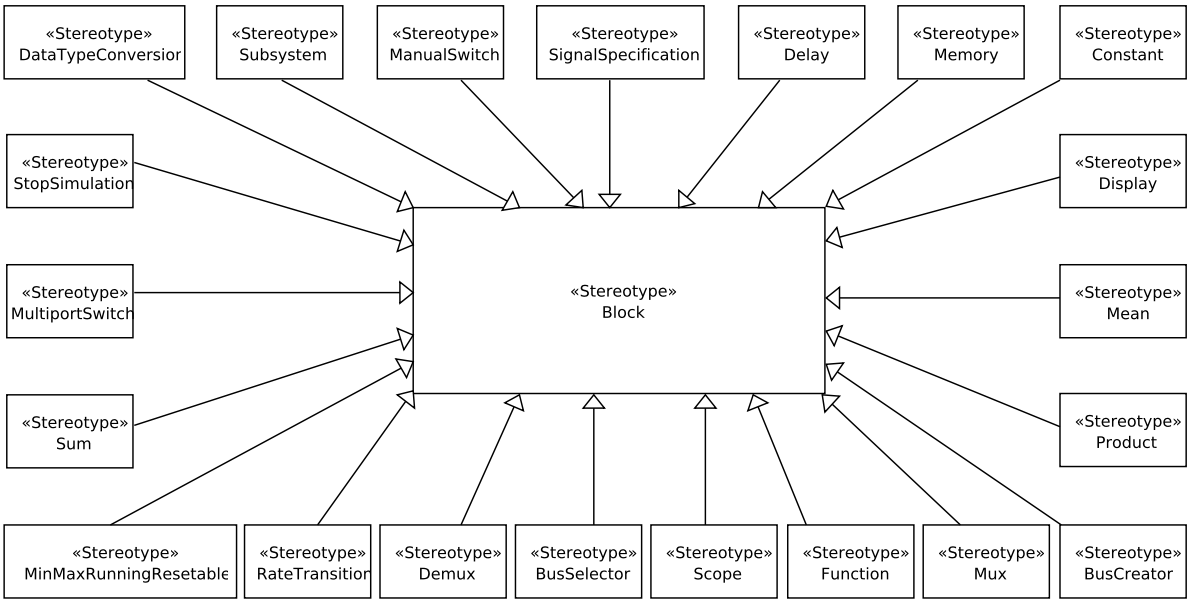

Figura 2.20: Perfil para os Blocos Funcionais

A Figura - 2.20 mostra o estereótipo «Block» ao qual estão associados vários estereótipos. Tais estereótipos foram criados com base nos blocos funcionais nativos da linguagem *Simulink*, e representam as principais classes de blocos existentes tanto em *Simulink* quanto em outras linguagens de blocos funcionais. No Apêndice A são descritos os estereótipos que compõem o perfil SyMPLES-ProfileFB.

Na estrutura interna do bloco Sensores do *mini-VANT* (Figura - 2.22) foram definidos dois pontos de variação:

- o **tipo de GPS** que será utilizado na aeronave, sendo as suas possíveis variantes os elementos `GPS Receiver A`, `GPS Receiver B` ou `GPS Receiver C`; e
- a **combinação de acelerômetros** que será utilizada, sendo as suas possíveis variantes os elementos `Acce1` e/ou `Acce2`.

A Figura - 2.21 mostra o modelo de características para a estrutura interna do bloco Sensores do *mini-VANT*, em que pode ser visualizado um elemento obrigatório, além dos dois pontos de variação citados anteriormente. Em tal figura é utilizada a notação da ferramenta *pure::variants* para o modelo de características.

A Figura - 2.22 mostra o modelo Simulink da LP para o bloco Sensores do *mini-VANT*. Os blocos mostrados na cor escura não fazem parte da lógica da aplicação, mas são necessários para representar e auxiliar na resolução das variabilidades, conforme definido no modelo de características da Figura - 2.21. O bloco `Multiport Switch` e o valor

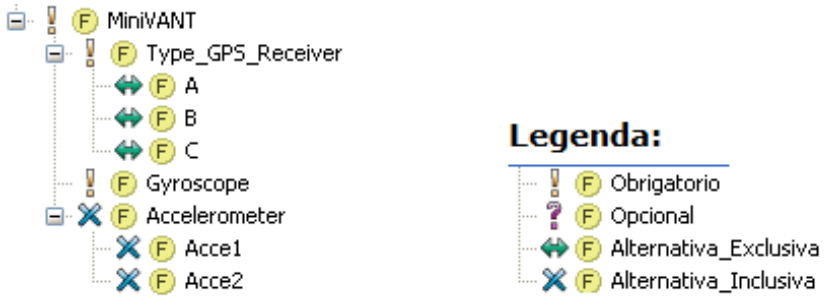

Figura 2.21: Modelo de características para o bloco Sensores do mini-VANT (Silva, 2012)

definido na constante `VAR GPS` definem qual dos três blocos do tipo `GPS Receiver` será selecionado (característica alternativa exclusiva). Analogamente, as constantes `A1` e `A2` determinam se os blocos `Accelerometer1` e `Accelerometer1` serão ou não habilitados (característica alternativa inclusiva), após o processo de resolução das variabilidades.

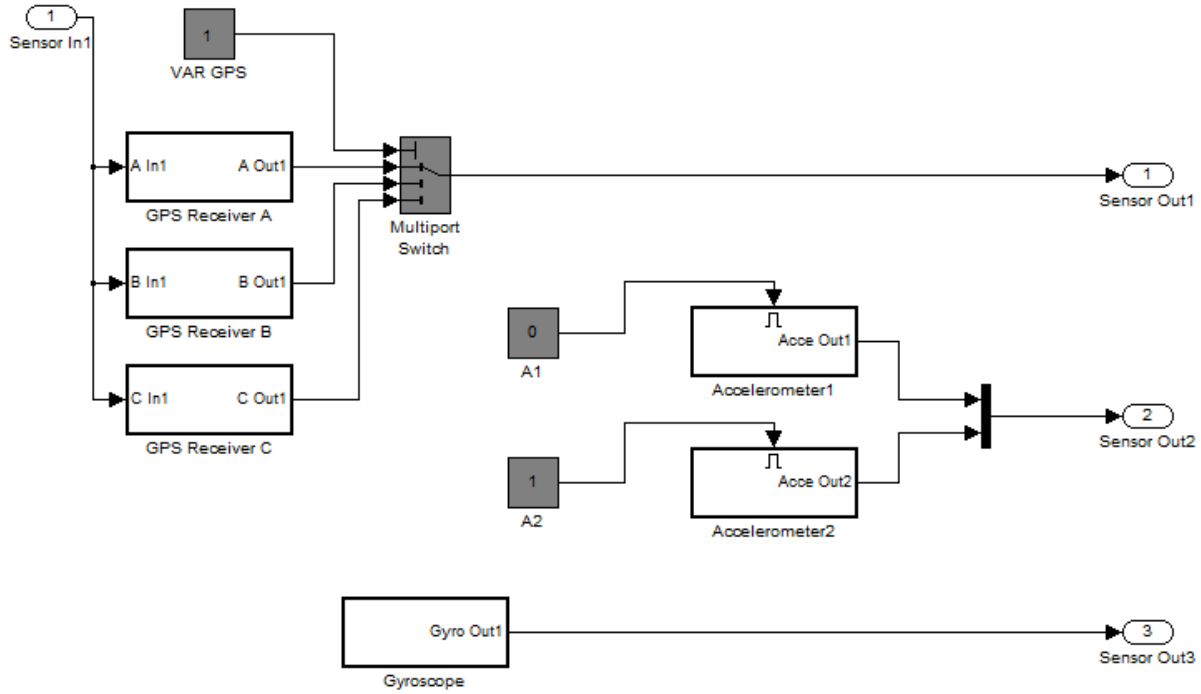

Figura 2.22: Modelo Simulink para a estrutura interna do bloco Sensores do mini-VANT (Silva, 2012)

A Figura - 2.23 mostra o diagrama interno do bloco Sensores do *mini-VANT* modelado em SysML. Foram utilizados os estereótipos do perfil SyMPLES-ProfileFB para fazer a relação com os blocos funcionais correspondentes. Assim como no modelo *Simulink*

apresentado na Figura - 2.22, ao se derivar uma instância específica, o modelo SysML deve ser modificado para desativar as características não selecionadas para o produto.

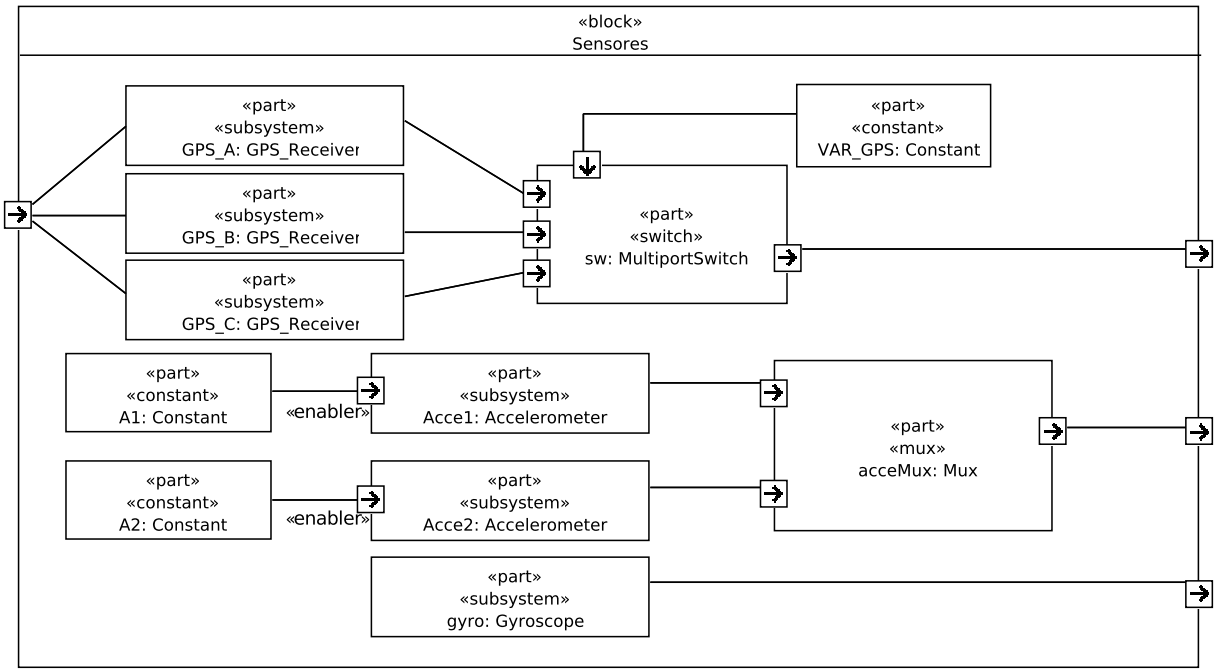

Figura 2.23: Representação em SysML da LPS do *mini-VANT*, com a utilização do perfil de blocos funcionais SyMPLES-ProfileFB (Silva, 2012)

Na Figura - 2.23, a característica alternativa exclusiva `Type_GPS_Receiver` é representada com a utilização de um bloco marcado com o estereótipo «Switch». Um bloco do tipo *Switch* seleciona apenas uma de suas entradas como valor de saída, de acordo com o valor de uma determinada variável. No exemplo do *mini-VANT* (Figura - 2.22), o valor estabelecido para o bloco `VAR_GPS` define qual dos três blocos `GPS_Receiver` será habilitado para o sistema.

Analogamente, a característica alternativa inclusiva `Accelerometer` é representada com a utilização de um bloco marcado com o estereótipo «Mux». Um bloco do tipo *Mux* realiza a combinação do valor de suas entradas em um único valor de saída, pois uma porta de saída que recebe mais de um valor de entrada representa um modelo *Simulink* inválido. Os blocos `Acce1` e `Acce2` possuem uma porta marcada com o estereótipo «Enabler», identificando que este bloco será habilitado ou desabilitado de acordo com o valor atribuído para a variável correspondente que está ligada à porta de entrada do bloco. No exemplo do *mini-VANT* (Figura - 2.22), o bloco `Acce1` está desabilitado, devido ao valor 0 (zero) atribuído ao bloco do tipo constante `A1`; e o bloco `Acce2` está habilitado, devido ao valor 1 (um) atribuído ao bloco do tipo constante `A2`.

2.4.2 O processo SyMPLES-ProcessPL

As atividades do processo SyMPLES-ProcessPL são baseadas nas atividades genéricas definidas pela abordagem *OOSEM* (Lykins et al., 2000). *OOSEM* utiliza os conceitos de orientação a objetos, em conjunto com os métodos tradicionais da engenharia de sistemas, para auxiliar a criação de projetos de sistemas complexos.

Bassi et al. (2011) definem uma metodologia iterativa, baseada na abordagem *OOSEM*, para auxiliar no processo de desenvolvimento de sistemas complexos. O processo SyMPLES-ProcessPL foi baseado em tal metodologia, porém foram incorporadas novas atividades para auxiliar a construção dos artefatos de uma LPS.

As próximas seções descrevem as atividades e o conjunto de diretrizes definidos para o processo SyMPLES-ProcessPL.

2.4.2.1 Análise de requisitos

O objetivo dessa atividade é realizar a análise tanto do ambiente do sistema como das necessidades dos usuários, para gerar uma lista de requisitos e o diagrama de casos de uso do sistema. O diagrama de casos de uso representa o escopo da aplicação por meio da identificação dos casos de uso e também os atores que interagem com o sistema.

A atividade de análise de requisitos é realizada com o apoio das seguintes diretrizes:

D.1.1.1 listar os requisitos (funcionais e/ou não-funcionais) do sistema;

D.1.1.2 definir os atores e casos de uso;

D.1.1.3 detalhar os casos de uso;

D.1.1.4 estruturar o modelo de casos de uso, com a utilização de relações como *include* e *extends*;

Conforme visualizado na Figura - 2.24, na análise de requisitos do *mini-VANT* foram identificados dois atores: o `Operador de Controle` dispara os casos de uso `Controlar Modo de Navegação`, `Gerenciar Piloto Automático` e `Gerenciar Sistema de Controle`; e o ator `Projetista da Missão` dispara os casos de uso `Configurar GPS` e `Configurar Acelerômetro`.

2.4.2.2 Refinamento dos Requisitos

Esta atividade recebe como entrada a lista de requisitos e o diagrama de casos de uso produzidos na atividade anterior e gera como saída um diagrama de requisitos SysML. O

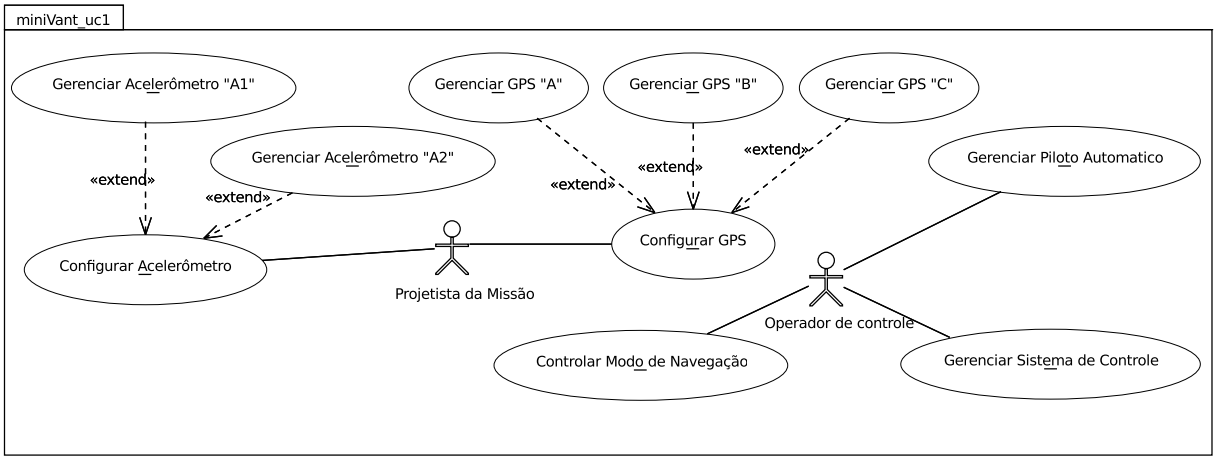

Figura 2.24: Diagrama de casos de uso para o *mini-VANT* (Silva, 2012)

objetivo do diagrama de requisitos é especificar as funcionalidades que um sistema deve implementar e as condições que ele deve satisfazer. Em um processo de refinamento, os requisitos podem ser decompostos em sub-requisitos, de forma a organizá-los em uma estrutura hierárquica. Cada requisito deve possuir um identificador único e um texto que contenha a sua descrição.

A atividade de refinamento dos requisitos pode ser realizada de acordo com as seguintes diretrizes:

D.1.2.1 refinamento da lista de requisitos construída na atividade anterior;

D.1.2.2 estruturação dos requisitos de forma hierárquica;

D.1.2.3 elaboração do diagrama de requisitos, utilizando os relacionamentos do tipo *dependency*, *deriveReqt* e *containment*;

A Figura - 2.25 mostra o diagrama de requisitos produzido para o *mini-VANT*. A partir do refinamento de uma lista de requisitos, tais requisitos são relacionados e dispostos de forma hierárquica em um diagrama. Na Figura - 2.25, foram utilizados os seguintes relacionamentos entre requisitos:

- *dependency*, estabelece uma relação de dependência entre dois requisitos; na Figura - 2.25 pode ser observado que o requisito `Controlar Modo de Navegação` possui uma relação de dependência com o requisito `Gerenciar Estação de Controle`;
- *deriveReqt*, utilizado no sentido de derivação, quando um requisito base dá origem a um sub-requisito; na Figura - 2.25 o requisito `Gerenciar Sensores` é derivado do requisito `Gerenciar Sistema de Controle do VANT`;

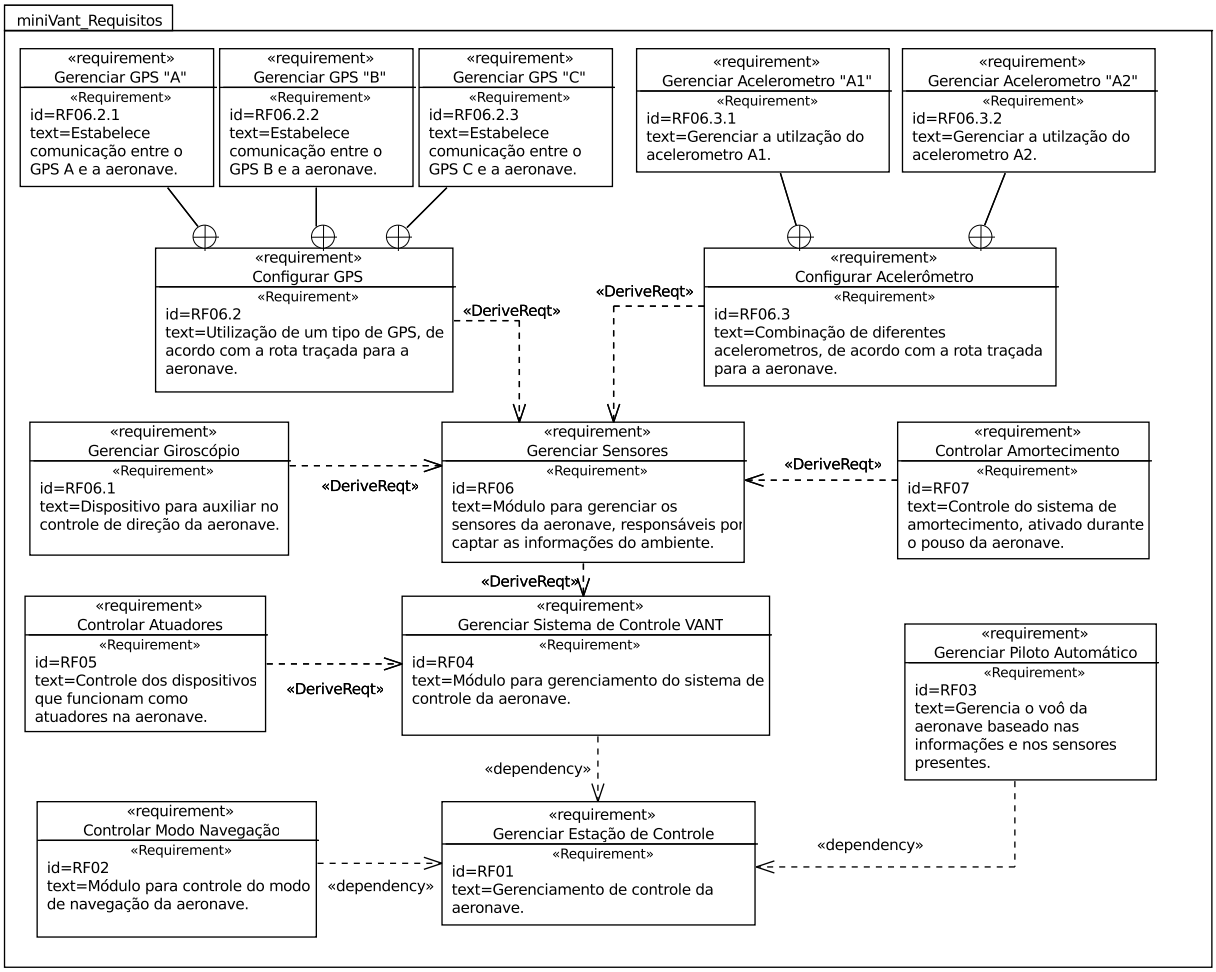

Figura 2.25: Diagrama de Requisitos para o *mini-VANT* (Silva, 2012)

- *containment*, é utilizado no sentido de partição, quando um requisito complexo pode ser particionado em requisitos mais simples, sem alterar o seu significado original; na Figura - 2.25 o requisito `Configurar GPS` é particionado entre os requisitos `Gerenciar GPS "A"`, `Gerenciar GPS "B"` e `Gerenciar GPS "C"`;

2.4.2.3 Definição do Modelo de Características

Esta atividade tem como objetivo identificar as características externamente visíveis dos produtos que comporão a LPS e organizá-los em um modelo de características. As características de um produto podem ser identificadas e classificadas a partir dos requisitos do sistema, sejam estes funcionais ou não-funcionais. Também, é possível definir uma característica a partir de técnicas de implementação que serão utilizadas, tecnologias utilizadas no domínio e ambientes operacionais (Lee et al., 2002).

A atividade de definição do modelo de características pode ser realizada a partir das seguintes diretrizes:

D.1.3.1 identificar os requisitos funcionais e não-funcionais comuns a todos os produtos como características obrigatórias;

D.1.3.2 definir as características opcionais a partir da lista de requisitos funcionais e não-funcionais;

D.1.3.3 definir as características alternativas (inclusivas e exclusivas) a partir da lista de requisitos funcionais e não-funcionais;

D.1.3.4 com base nas tecnologias utilizadas, técnicas de implementação e ambiente operacional, definir as características obrigatórias, opcionais e alternativas;

D.1.3.5 estruturar o modelo de características, de acordo com as características identificadas;

A Figura - 2.26 exibe o modelo de características completo do *mini-VANT*. Tais características foram criadas com base apenas nos requisitos funcionais do *mini-VANT*, portanto, não foram considerados os requisitos não-funcionais, tecnologias de utilização, técnicas de implementação e o ambiente operacional. A partir do diagrama de requisitos apresentado na Figura - 2.25, foram determinadas as características obrigatórias, opcionais, alternativas inclusivas e alternativas exclusivas que compõem o modelo de características do *mini-VANT*.

2.4.2.4 Definição da Arquitetura

O objetivo desta atividade é particionar o sistema em blocos e definir como estes interagem para satisfazer os requisitos do sistema. Os blocos são abstrações que implementam os requisitos do sistema, sem necessariamente impor restrições relacionadas às tecnologias de implementação.

A atividade de definição da arquitetura pode ser realizada a partir das seguintes diretrizes:

D.1.4.1 a partir dos requisitos do sistema, definir os blocos que irão implementar tais requisitos;

D.1.4.2 identificar o relacionamento entre os blocos, por meio da criação de portas lógicas e do diagrama de definição de blocos;

Figura 2.26: Modelo de Características completo do *mini-VANT* (Silva, 2012)

D.1.4.3 estruturar o diagrama de definição de blocos, identificando as associações de generalização/especialização, composição e agregação;

A Figura - 2.27 mostra o diagrama de blocos do *mini-VANT* em que foram definidos os blocos que satisfazem os requisitos do sistema. Tais blocos foram marcados com os estereótipos do perfil SyMPLES-ProfileFB, fazendo a correspondência com os blocos equivalentes na linguagem *Simulink*. A delimitação das variabilidades com uso da CVL será explicada com detalhes na Seção 3.2.

2.4.2.5 Refinamento da Arquitetura

Esta atividade tem como objetivo definir a estrutura interna dos blocos, por meio da criação dos diagramas internos do bloco. A atividade pode ser realizada utilizando as seguintes diretrizes:

D.1.5.1 identificar as partes (equivalentes a objetos) necessárias para realizar a funcionalidade definida para cada bloco;

D.1.5.2 interligar e estruturar as partes identificadas por meio da criação do diagrama de definição do bloco;

Como exemplo de artefato produzido por esta atividade, pode-se citar o diagrama interno do bloco Sensores do *mini-VANT*, exibido na Figura - 2.23. Neste diagrama, as partes (objetos) necessárias para implementação da funcionalidade definida para o bloco foram identificadas e interligadas por meio de portas e conectores.

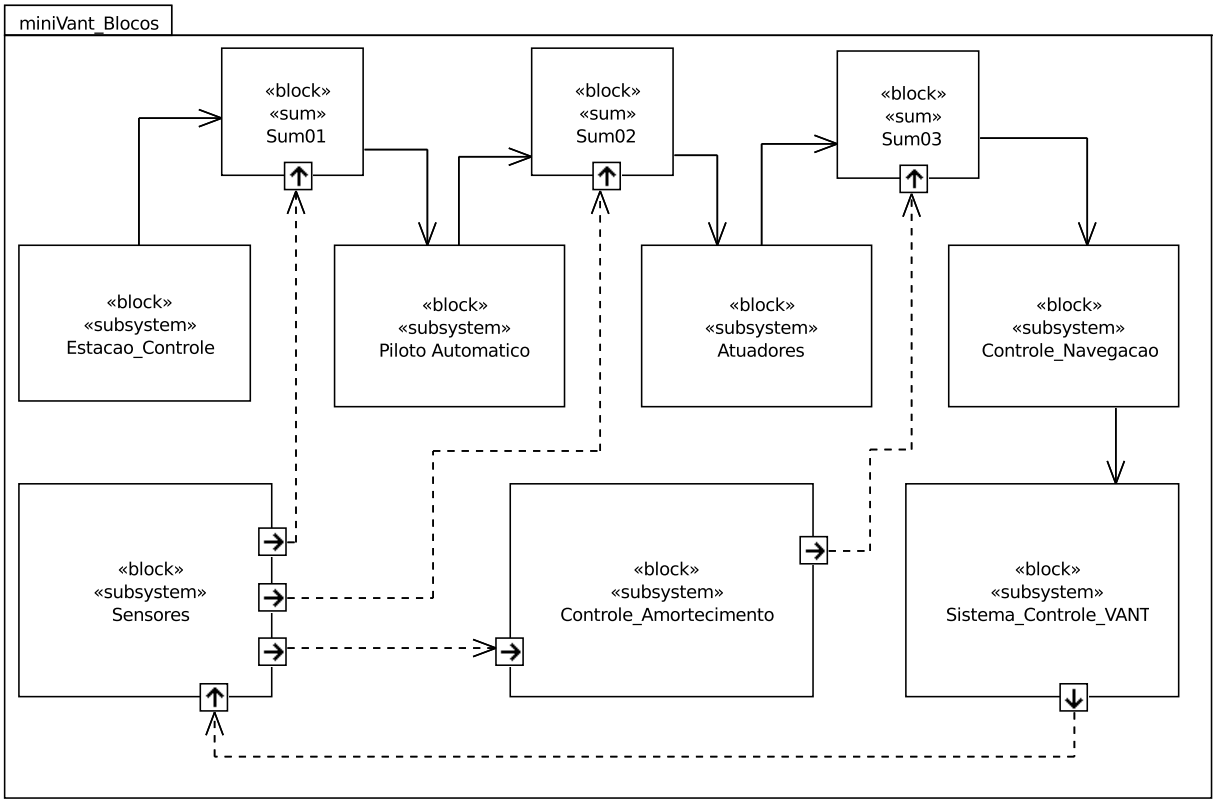

Figura 2.27: Diagrama de Definição de Blocos para o *mini-VANT* (Silva, 2012)

2.4.2.6 Mapeamento dos elementos da arquitetura do sistema aos requisitos

Esta atividade realiza a alocação entre os blocos criados durante a definição da arquitetura (Seção 2.4.2.5) e os requisitos definidos no diagrama de requisitos (Seção 2.4.2.2). Dessa forma, é possível gerar relatórios de rastreabilidade de requisitos a partir da arquitetura definida, e visualizar os impactos de possíveis alterações nos requisitos. Em SysML, opta-se pela utilização do diagrama de requisitos para realizar o mapeamento aos demais elementos da arquitetura, pois ao contrário do modelo de casos de uso, o diagrama de requisitos possibilita a representação de requisitos não-funcionais, e a consequente relação destes com os demais modelos da arquitetura.

Esta atividade pode ser realizada utilizando a seguinte diretriz:

D.1.6.1 no diagrama de requisitos, utilizar os relacionamentos *deriveReqt*, *refine*, *satisfy* e *trace* para identificar os blocos que estão relacionados um determinado requisito;

A Figura - 2.28 exibe o Diagrama de Requisitos do *mini-VANT*, em que é possível visualizar os elementos da arquitetura da aeronave relacionados com os requisitos correspondentes.

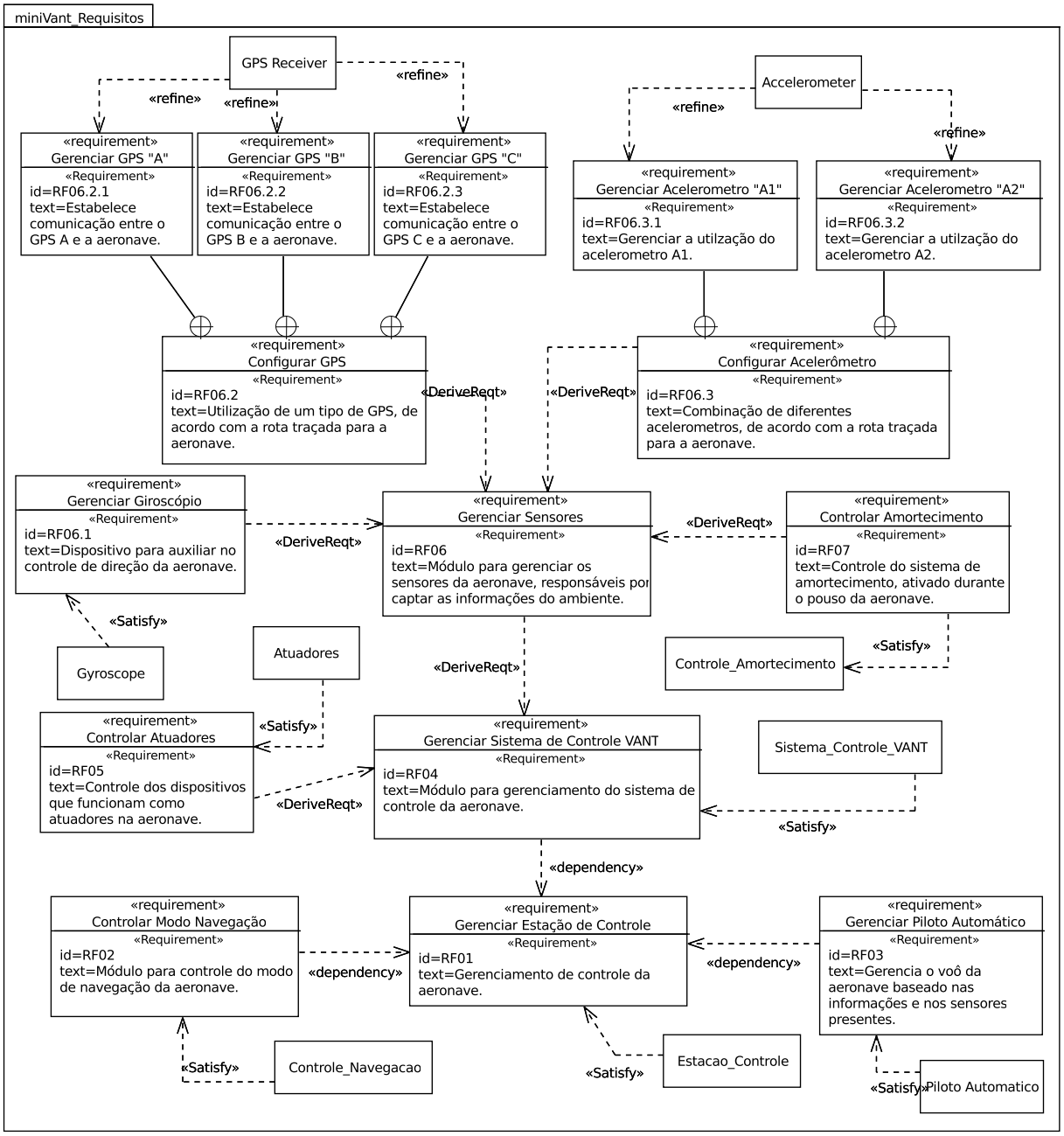

Figura 2.28: Diagrama de Requisitos do *mini-VANT* com os elementos da arquitetura mapeados

SyMPLES ainda conta com uma extensão que se apoia em técnicas MDE para auxiliar nas atividades da engenharia de aplicação, principalmente na transformação de modelos SysML, gerados pela ferramenta Papyrus (Eclipse Foundation, 2014), para a plataforma Simulink (Mathworks, 2011). O processo de transformação, definido por Fragal (2013), utiliza o framework EMF (Eclipse Foundation, 2008) em conjunto com a linguagem ATL (Obeo, 2015) usada para definir as regras de transformação. Através dos estereótipos

definidos pelo perfil SyMPLES-ProfileFB, é possível realizar o mapeamento entre os elementos SysML e as principais classes de blocos funcionais da plataforma Simulink, dessa forma possibilitando a transformação de modelos SysML para modelos Simulink.

2.5 Considerações finais

Este capítulo apresentou os conceitos relacionados à LPS, gerenciamento e representação de variabilidades, necessários para o entendimento da proposta desta dissertação. LPS é uma abordagem promissora e atual para o desenvolvimento de software, pois desenvolve um conjunto de produtos de um domínio específico com base em uma estrutura comum, a partir da qual diferentes produtos são derivados, considerando as variações necessárias para a geração de cada um deles. O gerenciamento de variabilidade é a atividade responsável por definir, representar, implementar e evoluir as variabilidades da LPS. Nas LPS baseadas em SysML, além dos modelos SysML é necessário relacionar à infraestrutura de LPS um modelo específico para representar as variabilidades. Nesse cenário, foi apresentada a linguagem CVL, que permite a expressão dos conceitos de variabilidade em modelos baseados em MOF, tais como UML e SysML. As técnicas de LPS são usadas amplamente no domínio de sistemas embarcados, como a indústria automotiva, aeronáutica e eletrônica.

O próximo capítulo apresentará a extensão da abordagem SyMPLES que é desenvolvida neste trabalho de mestrado, sobre o desenvolvimento de LPS para sistemas embarcados baseado em SysML.

3

Abordagem SyMPLES-CVL

Este capítulo apresenta SyMPLES-CVL, uma alternativa à abordagem SyMPLES-SMarty, proposta para o desenvolvimento de LPS baseadas em SysML para o domínio de sistemas embarcados. A SyMPLES-CVL utiliza a linguagem CVL para a identificação e representação das variabilidades das LPS. Inicialmente são exibidas as características de SyMPLES-CVL, juntamente com suas semelhanças e diferenças de SyMPLES-SMarty. Em seguida, é apresentado o *SyMPLES Process for Identification of Variabilities in CVL* (SyMPLES-ProcessVarCvl) que é executado simultaneamente ao SyMPLES-ProcessPL, e define um conjunto de atividades e diretrizes para auxiliar na identificação e delimitação de variabilidades com CVL. Por fim, é apresentada uma atividade alternativa ao Processo CVL de configuração de produtos e as considerações finais.

3.1 Apresentação Geral de SyMPLES-CVL

A abordagem SyMPLES-SMarty (Fragal, 2013; Silva, 2012; Silva et al., 2013), originalmente chamada de SyMPLES, contém dois perfis e dois processos, conforme descritos na Seção 2.4.

SyMPLES-CVL é composta de apenas um perfil; e dois processos conforme descritos a seguir.

PERFIL:

1. *SyMPLES Profile for Functional Blocks* (**SyMPLES-ProfileFB**). Originalmente este perfil é um conjunto de estereótipos com o objetivo de fornecer uma semântica adicional aos blocos SysML. Na versão proposta este perfil se manteve inalterado.

O SyMPLES-ProfileVar é um perfil para representação de variabilidades baseado em um perfil definido em SMarty. Devido ao fato da substituição do SMarty por CVL, este perfil não foi mais necessário.

PROCESSOS:

1. *SyMPLES Process for Product Lines* (**SyMPLES-ProcessPL**) é um processo que define atividades genéricas que auxiliam a criação dos artefatos de uma LPS. Este processo se manteve sem alterações porque as atividades definidas por ele não se relacionam com o SMarty;

2. *SyMPLES Process for Identification of Variabilities in CVL* (**SyMPLES-ProcessVarCvl**) é um processo para a representação das variabilidades de uma LPS, baseado no processo SyMPLES-ProcessVar, original. O SyMPLES-ProcessVarCvl, portanto substitui o SyMPLES-ProcessVar. Ele é executado simultaneamente ao processo SyMPLES-ProcessPL cujo objetivo é auxiliar o usuário na identificação, delimitação e representação das variabilidades com a linguagem CVL, além da configuração de produtos de uma LPS para Sistemas Embarcados baseada em SysML;

Desta maneira, o SyMPLES-CVL contém o perfil SyMPLES-ProfileFB, e os processos SyMPLES-ProcessPL e SyMPLES-ProcessVarCvl, responsáveis para a construção e definição de LPS de Sistemas Embarcados.

A Figura - 3.1 apresenta um diagrama ilustrando a interação entre o processo SyMPLES-ProcessPL, representado pelas atividades do retângulo do lado esquerdo e o processo SyMPLES-ProcessVarCvl, representado pelas atividades do retângulo à direita. Tais processos são executados simultaneamente durante o desenvolvimento de LPS. As atividades e diretrizes que compõem o SyMPLES-ProcessPL foram descritos na Seção 2.4.2. O SyMPLES-ProcessVarCvl é descrito a seguir.

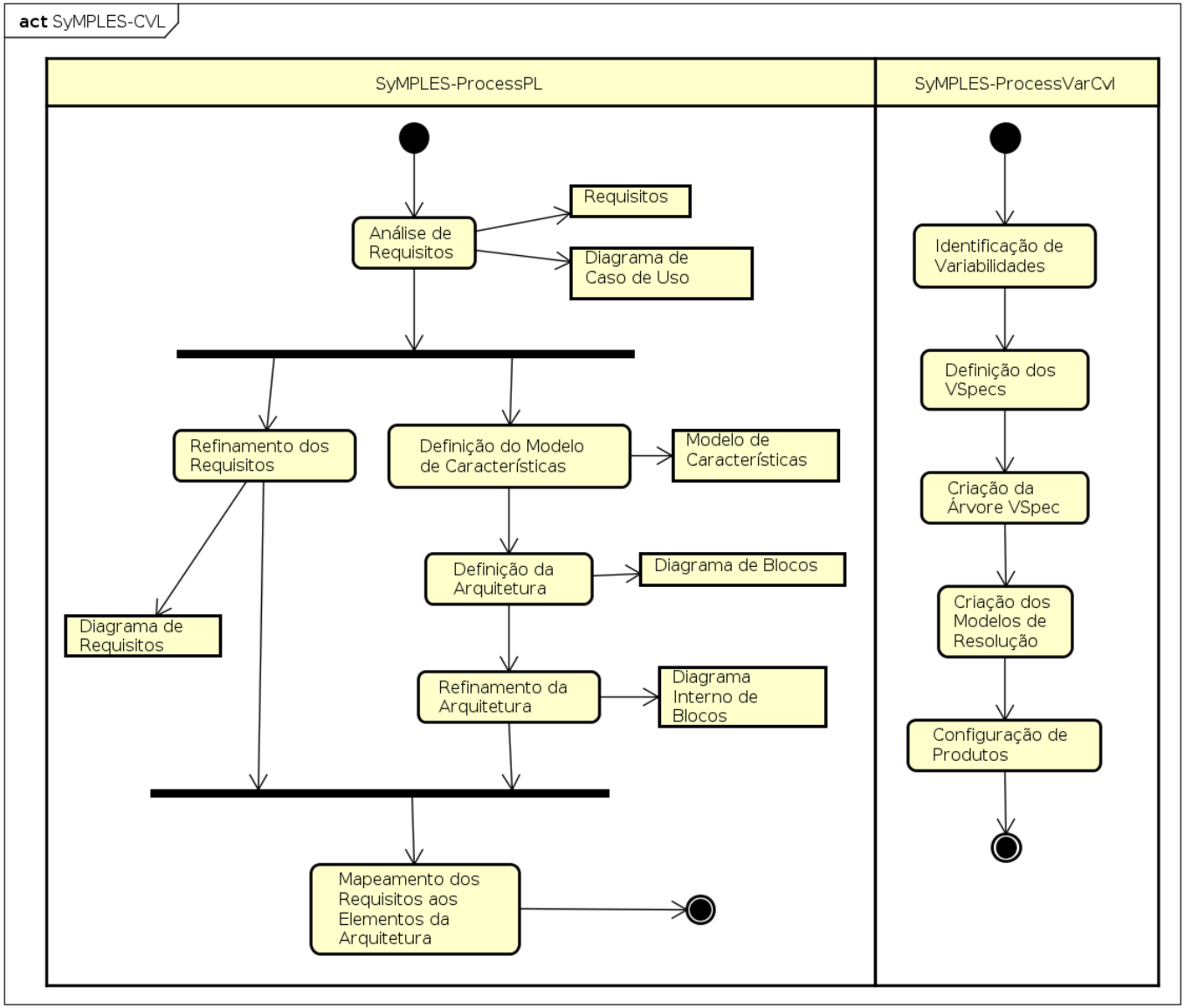

Figura 3.1: Interação entre os processos SyMPLES-ProcessPL e SyMPLES-ProcessVarCvl

3.2 O processo SyMPLES-ProcessVarCvl

O processo SyMPLES-ProcessPL é executado simultaneamente com o processo SyMPLES-ProcessVarCvl, com o objetivo de identificar e delimitar as variabilidades de uma LPS baseada em SysML para a criação de Modelos de Variabilidade CVL, bem como para sistematizar um processo de resolução das variabilidades, em CVL chamado de Materialização.

Para efeito de exemplificação da aplicação das atividades e diretrizes SyMPLES-ProcessVarCvl, foram identificadas e delimitadas as variabilidades dos modelos SysML exibidos na Seção 2.4. As variabilidades foram identificadas com base no Modelo de Características completo do *mini-VANT* exibido na Figura - 2.26.

- Na Figura - 3.2 é apresentado um exemplo de uma LPS com as variabilidades identificadas e delimitadas com o SyMPLES-ProcessVarCvl. A parte inferior mostra o mesmo Diagrama de Casos de Uso da Figura - 2.24, e na parte superior é exibido um Modelo de Variabilidades da linguagem CVL.

- Na Figura - 3.3 temos um Diagrama de Requisitos com as variabilidades representadas em CVL. A parte inferior é o mesmo diagrama exibido na Figura - 2.25, e na parte superior temos o Modelo de Variabilidades CVL.

- A Figura - 3.5 exibe o Diagrama de Requisitos do *mini-VANT* com os elementos da arquitetura mapeados, e inclui o Modelo de Variabilidades CVL para representar as variabilidades. A versão sem variabilidades representadas é exibida na Figura - 2.28.

- A Figura - 3.4 exibe o Diagrama de Definição de Blocos e o seu Modelo de Variabilidades CVL correspondente.

Os Modelos de Variabilidades CVL correspondentes a Figura - 3.2, Figura - 3.3 e Figura - 3.5 definem que: (a) os elementos Gerenciar `Acelerometro A1` e `Acelerometro A2` estão contidos em um grupo de variantes inclusivas; (b) os elementos Gerenciar `GPS A`, `GPS B` e `GPS C` fazem parte de um grupo de variantes exclusivas; e, (c) o elemento `Gerenciar Piloto Automatico` é uma variante opcional.

Na Figura - 3.4 todos os blocos foram identificados como variantes obrigatórias, com exceção do Bloco `Piloto Automatico`, que segundo o Modelo de Características do `mini-VANT`, é uma variante opcional. O Diagrama de Definição de Blocos equivalente é exibido na Figura - 2.27.

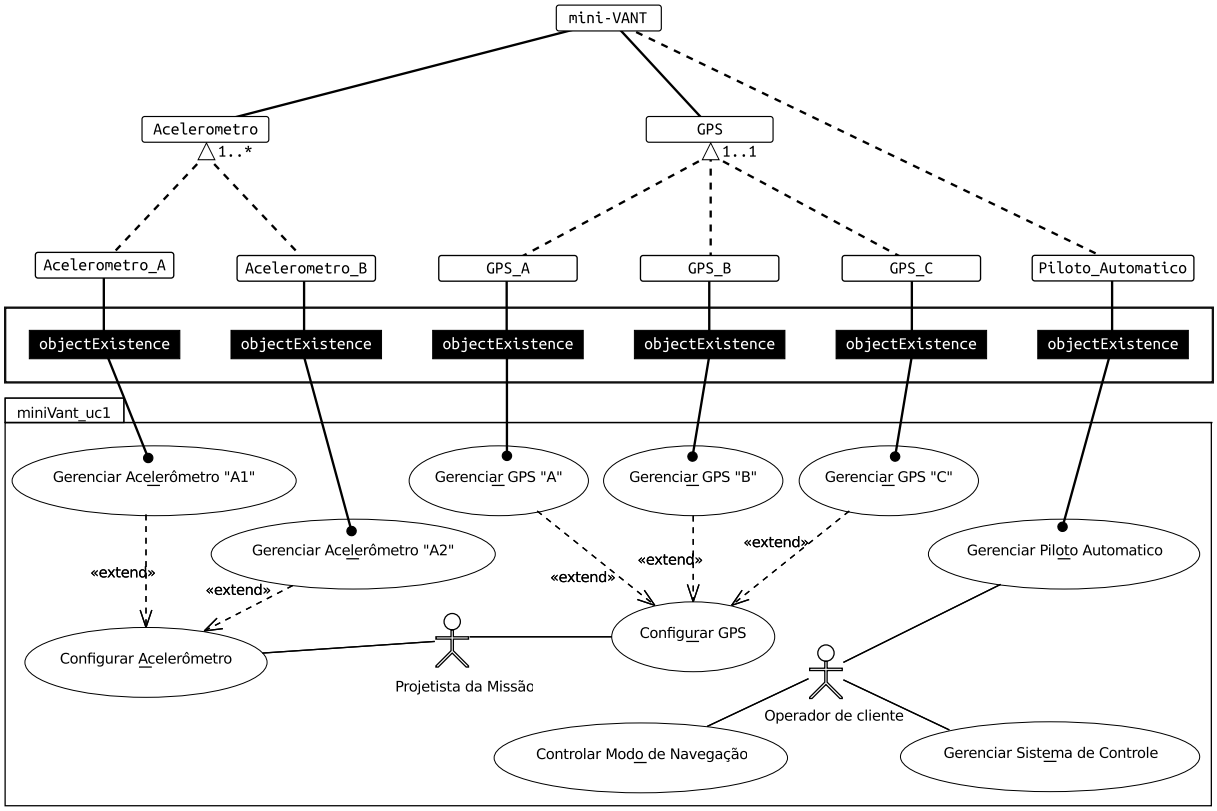

Figura 3.2: Diagrama de Casos de Uso para o *mini-VANT* com variabilidades representadas em CVL

As atividades e diretrizes do SyMPLES-ProcessVarCvl são descritas nas seções seguintes.

3.2.1 Identificação de Variabilidades

A cada interação entre o processo SyMPLES-ProcessPL e o SyMPLES-ProcessVarCvl, a atividade de identificação de variabilidades recebe como entrada o modelo de características, casos de uso, de requisitos, de definição de blocos e interno de blocos. Esta atividade tem como objetivo iniciar a criação de um Modelo de Variabilidades da linguagem CVL para cada modelo SysML que foi utilizado como entrada. Assim, nenhuma anotação ou conceito de variabilidade é adicionado aos modelos SysML utilizados como entrada. Estes modelos de entrada, na definição da CVL, são chamados de Modelos Base.

Na definição da linguagem CVL (Haugen, 2012), Pontos de Variação CVL são especificações de variabilidade no Modelo Base e fazem parte do Modelo de Variabilidade. Eles definem modificações específicas que serão aplicadas no Modelo Base durante o processo de Materialização. Localizados na camada de Realização da Variabilidade da CVL, os pontos

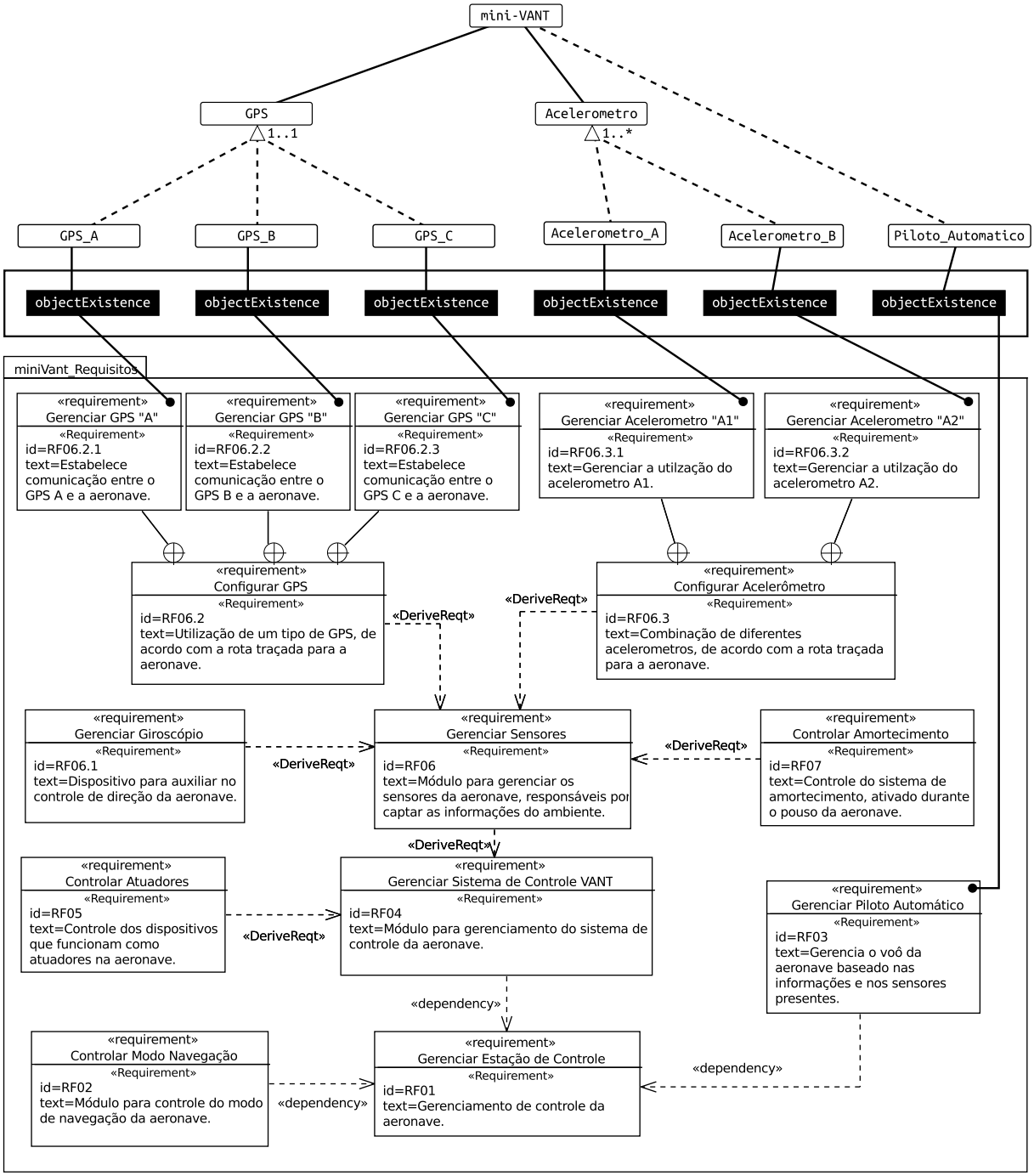

Figura 3.3: Diagrama de Requisitos para o *mini-VANT* com variabilidades representadas em CVL

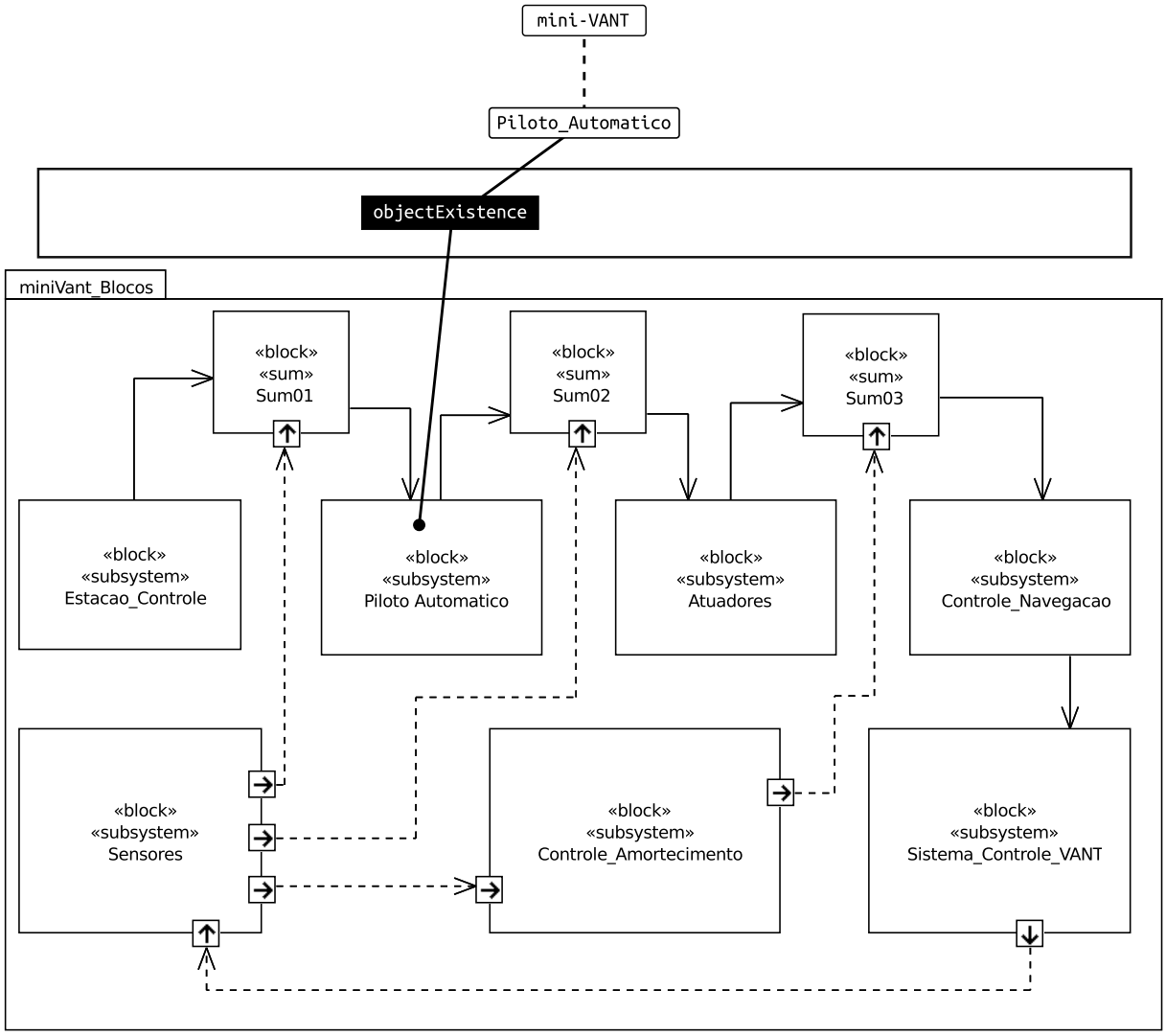

Figura 3.4: Diagrama de Definição de Blocos para o *mini-VANT* com variabilidades representadas em CVL

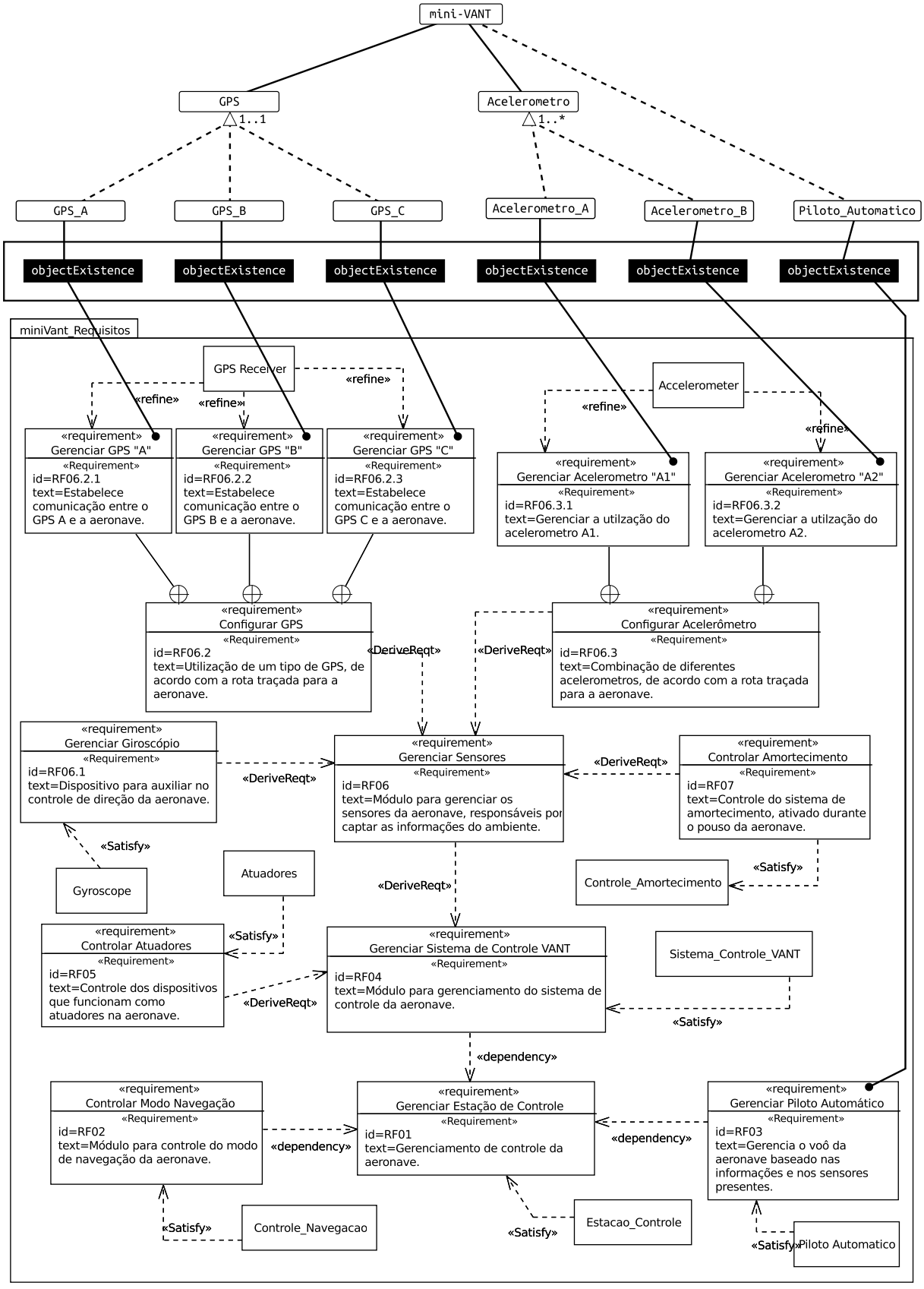

Figura 3.5: Diagrama de Requisitos do *mini-VANT* com os elementos da arquitetura mapeados e variabilidades representadas em CVL

de variação fazem referência aos elementos do Modelo Base por meio dos "Manipuladores do Modelo Base", e ao mesmo tempo estão ligados aos *VSpecs* da Camada de Abstração de Variabilidade. A ligação de um Ponto de Variação CVL a um *VSpec* significa que a aplicação do Ponto de Variação CVL no Modelo Base durante a Materialização depende da resolução do *VSpec*.

As diretrizes fornecidas para apoio desta atividade são descritas nas seções a seguir.

3.2.1.1 Identificação dos Pontos de Variação CVL Existência de Objeto

Os Pontos de Variação CVL *ObjectExistence* indicam que um objeto do Modelo Base pode ou não existir no modelo materializado. Este tipo de ponto de variação deve estar ligado a um *VSpec* do tipo Escolha. Durante a resolução, quando este Ponto de Variação CVL é resolvido, o objeto identificado no Modelo Base poderá ser excluído.

Todas as variantes opcionais, inclusivas ou exclusivas identificadas da diretriz **D.2.1.1** até a diretriz **D.2.1.8** deverão ser marcadas como Pontos de Variação CVL do tipo `ObjectExistence` no seu respectivo Modelo de Variabilidade.

D.2.1.1 os elementos de modelos de casos de uso relacionados aos mecanismos de extensão e de pontos de extensão indicam pontos de variação com variantes associadas, as quais podem ser inclusivas ou exclusivas. Assim, os casos de uso base representam pontos de variação, enquanto os casos de uso estendidos, representam variantes. No diagrama de casos de uso da Figura - 3.2, por exemplo, os casos de uso `Gerenciar Acelerômetro "A1"` e `Gerenciar Acelerômetro "A2"` representam variantes inclusivas associadas ao ponto de variação `Configurar Acelerômetro` (caso de uso estendido);

D.2.1.2 elementos de modelos de casos de uso relacionados com a associação «include» ou associados a atores indicam variantes obrigatórias ou opcionais;

D.2.1.3 pontos de variação e suas variantes em modelos de requisitos podem ser identificadas nos seguintes relacionamentos:

a) relação ***containment***, os requisitos mais gerais são os pontos de variação, enquanto os requisitos contidos são as suas variantes, as quais podem ser inclusivas ou exclusivas; na Figura - 3.3 a relação de contenção dos requisitos `Gerenciar GPS "A"`, `Gerenciar GPS "B"` e `Gerenciar GPS "C"` representam variantes exclusivas associadas ao ponto de variação `Configurar GPS`;

b) **derivação** (relação *deriveReqt*), os requisitos derivados indicam variantes obrigatórias ou opcionais; na Figura - 3.3, o requisito `Controlar Atuadores` é uma variante obrigatória do requisito `Gerenciar Sistema de Controle do VANT`;

c) **dependência** (relação *dependency*), os requisitos dependentes indicam variantes obrigatórias ou opcionais; na Figura - 3.3, o requisito `Controlar Modo de Navegação` é uma variante obrigatória;

D.2.1.4 pontos de variação e suas variantes em modelos de blocos (diagrama de definição de blocos) são identificadas nos seguintes relacionamentos:

a) **generalização**: os classificadores mais gerais são os pontos de variação, enquanto os mais específicos são as variantes;

b) **realização de interface**: os "*suppliers*" (especificações) são os pontos de variação e as implementações (clientes) são as variantes;

c) **agregação**: as instâncias tipadas com losangos não preenchidos são os pontos de variação e as instâncias associadas são as variantes; e

d) **composição**: as instâncias tipadas com losangos preenchidos são os pontos de variação e as instâncias associadas são as variantes.

D.2.1.5 os elementos de modelos de blocos (diagramas de blocos), relacionados às associações nas quais os seus atributos *aggregationKind* possuem valor *none* (não representam nem agregação nem composição) indicam variantes obrigatórias ou opcionais; na Figura - 3.4, todas as relações entre blocos são variantes obrigatórias.

D.2.1.6 se um bloco ou parte marcado com o estereótipo «Subsystem» possuir uma porta de entrada marcada com o estereótipo «enabler», tal bloco será identificado como uma variante opcional. Isso acontece, pois um bloco que possui uma porta de entrada do tipo *enabler* tem a sua funcionalidade desabilitada quando o valor recebido em tal porta é igual a zero;

D.2.1.7 partes ou blocos que possuem uma porta de entrada do tipo *enabler*, quando ligados a um bloco marcado com o estereótipo «mux» sugerem variantes alternativas inclusivas, pois um bloco do tipo *mux* combina o valor de suas entradas em um único valor de saída; na Figura - 2.23, as partes `Acce1` e `Acce2` foram identificadas como variantes inclusivas, pois possuem uma porta do tipo *enabler* e estão ligadas a um bloco do tipo *mux*.

D.2.1.8 partes ou blocos ligados a um bloco marcado com o estereótipo «switch» sugerem variantes alternativas exclusivas, pois um bloco do tipo *switch* seleciona como saída apenas uma de suas entradas, de acordo com o valor de uma constante; na Figura - 2.23 as partes `GPS_A`, `GPS_B` e `GPS_C` são do tipo `GPS_Receiver`, e foram identificadas como variantes alternativas exclusivas pois possuem as características citadas anteriormente.

3.2.1.2 Identificação dos Pontos de Variação CVL Substituição de Objetos

Um *ObjectSubstitution* é um Ponto de Variação CVL que deve estar ligado a um *VSpec* do tipo Escolha, e especifica se um objeto no Modelo Base, chamado de Objeto de Fixação (*replacement*), pode ser substituido por outro, chamado de Objeto de Substituição (*placement*). Os objetos de fixação e substituição são especificados via Manipuladores de Objetos CVL e identificam elementos no Modelo Base. Quando o *VSpec* é resolvido positivamente este ponto de variação é aplicado. Assim todas as ligações apontadas para o Objeto de Fixação são redirecionadas para o Objeto de Substituição.

Todas as variantes exclusivas identificadas da diretriz **D.2.2.1** até a diretriz **D.2.2.2**, e que não foram marcadas com outros tipos de Pontos de Variação CVL poderão ser marcadas como Pontos de Variação CVL do tipo *ObjectSubstitution* no seu respectivo Modelo de Variabilidade.

D.2.2.1 em modelos de blocos (diagrama de definição de blocos), objetos de fixação e os objetos de substituição são identificadas nos seguintes relacionamentos:

a) **generalização**, os classificadores mais gerais são os objetos de fixação, enquanto os mais específicos são os objetos de substituição;

b) **realização de interface**, os "*suppliers*" (especificações) são os objetos de fixação e as implementações (clientes) são os objetos de substituição;

D.2.2.2 partes ou blocos ligados a um bloco marcado com o estereótipo «switch» sugerem variantes alternativas exclusivas, pois um bloco do tipo *switch* seleciona como saída apenas uma de suas entradas, de acordo com o valor de uma constante; assim o bloco marcado com «switch» pode ser o objeto de fixação enquanto que as suas entradas são os objetos de substituição;

3.2.1.3 Identificação dos Pontos de Variação CVL Atribuição de Valor Paramétrico

Os Pontos de Variação CVL *ParametricSlotAssignment* especificam que um valor pode ser atribuído a um *slot* de um objeto no Modelo Base. O objeto é identificado por meio de

um Manipulador de Objetos apontando para o Modelo Base, e o *slot* é identificado pelo seu nome, armazenado no atributo *slotIdentifier*. O valor a ser atribuído é especificado explicitamente. Durante o processo de materialização do produto, quando o Ponto de Variação CVL é aplicado, o valor especificado é inserido no *slot* do Modelo Base.

Todas as variantes identificadas da diretriz **D.2.3.1** até a diretriz **D.2.3.2** deverão ser marcadas como Pontos de Variação CVL do tipo `ParametricSlotAssignment` no Modelo de Variabilidade.

D.2.3.1 blocos marcados com o estereótipo «constant» possuem um parâmetro, no qual sugerem um *slot* para atribuição de valores. Na Figura - 2.23, aos blocos `VAR_GPS`, `A1` e `A2` foram atribuídos pontos de variação *ParametricSlotAssignment* pois possuem o estereótipo «constant».

D.2.3.2 atributos de tipos primitivos, como *int*, *string* ou *boolean*, e que possuem valores iniciais sugerem um *slot* para atribuição de valores;

3.2.1.4 Identificação dos Pontos de Variação CVL Repetíveis

Um Ponto de Variação CVL Repetível pode ser aplicado várias vezes durante a Materialização. Este tipo de ponto de variação deve estar ligado a um *VSpec* do tipo *VClassifier* e é aplicada uma vez para cada instância criada pela materialização.

Todos os pontos de variação identificados pelas próximas diretrizes devem ser marcados com Pontos de Variação Repetíveis no seu respectivo Modelo de Variabilidade.

D.2.4.1 em modelos de blocos (diagrama de definição de blocos), pontos de variação são identificadas nos seguintes relacionamentos:

a) **agregação**, se a multiplicidade do relacionamento permitir mais de uma instância do elemento "parte", então o elemento "parte" pode ser marcado com o Ponto de Variação CVL Repetível; e

b) **composição**, se o elemento "todo" aceitar várias instâncias do elemento "parte", então o elemento "parte" pode ser marcado com o Ponto de Variação CVL Repetível.

3.2.1.5 Identificação de restrições entre Pontos de Variação CVL

D.2.5.1 uma variante `X` que, ao ser selecionada para fazer parte de um produto, exige a presença de outra determinada variante `Y` deve ter seus relacionamentos de dependência marcadas com a restrição `X implies Y` na *CVL Tree*;

D.2.5.2 uma variante `X` que, ao ser selecionada para fazer parte de um produto, exige a exclusão de outra determinada variante `Y` deve ter seus relacionamentos de dependência marcadas com a restrição `X implies not Y` na *CVL Tree*;

3.2.2 Definição dos VSpecs

Esta atividade tem como objetivo definir os *VSpecs* a partir dos modelos utilizados como entrada e dos Pontos de Variação CVL definidos na Seção 3.2.1.

Conforme dito na Seção 2.2.2, a arquitetura da CVL contém uma camada denominada Abstração da Variabilidade que fornece construtores para a especificação e resolução de variabilidades em um nível abstrato, ou seja, sem especificar a natureza da variabilidade do Modelo Base. O conceito central é o de uma especificação de variabilidade, que abreviamos como *VSpecs*. Tecnicamente, *VSpecs* são similares às características dos Modelos de Características, mas podem ser considerados Especificadores Abstratos de Variabilidade. Por exemplo, ao invés pensar "GPS" como uma característica que uma camera pode ou não ter, podemos imaginar isto como uma **escolha**, que pode ser decidida como positivamente ou negativamente (Haugen, 2012).

Um *Vspec* é uma indicação da variabilidade no Modelo Base. Os detalhes da variabilidade, ou, que elementos no Modelo Base estão envolvidos e como eles serão afetados, não é especificado, isto que torna os *VSpecs* abstratos. Na CVL, o efeito sobre o Modelo Base é especificado pela ligação dos Pontos de Variação CVL com os *VSpecs*, que referem-se ao Modelo Base.

Na definição da CVL existem 4 tipos de *VSpecs* (Haugen, 2012), dos quais 3 serão tratados neste trabalho. São eles:

1. escolha (*choice*) - é um *Vspec* cuja resolução requer uma decisão sim/não. Nada é conhecido sobre a natureza de uma escolha no nível de uma Árvore *VSpec*, exceto o que é sugerido pelo seu nome;

2. variável (*variable*) - é um *VSpec* cuja resolução envolve o fornecimento de um valor de um tipo especificado. Este valor é destinado para ser utilizado no Modelo Base.

3. classificador de variabilidade (*variability classifier* ou *VClassifier*) - é uma espécie de *Vspec* cuja resolução significa a criação de instâncias e então, o fornecimento de resoluções para os *Vspecs* da sua sub-árvore;

Esta atividade pode ser executada de acordo com as seguintes diretrizes:

D.2.6.1 identificar Pontos de Variação CVL Existência de Objeto. Cada um deve ser representado por um *VSpec* do tipo Escolha;

D.2.6.2 identificar grupos de Ponto de Variação CVL Existência de Objetos que estão ligados aos elementos do Modelo Base que foram identificados como variantes de um grupo de variantes inclusivas/exclusivas. Cada grupo identificado deve ser representado por um *VSpec* Escolha. Este *VSpec* é considerado um *VSpec* pai para os *VSpecs* do grupo identificado;

D.2.6.3 identificar Pontos de Variação CVL Substituição de Objeto. Cada um deve ser representado por um *VSpec* do tipo Escolha;

D.2.6.4 identificar Pontos de Variação CVL Atribuição de Valor Paramétrico. Cada um deve ser representado por um *VSpec* do tipo Variável;

D.2.6.5 identificar Pontos de Variação CVL Repetível. Cada um deve ser representado por um *VSpec* do tipo *VClassifier*;

Por exemplo, todos os Pontos de Variação CVL da Figura - 3.2 são do tipo Existência de Objeto e devem estar representados por um *VSpec* do tipo Escolha na Árvore *VSpec*. Assim, durante o processo de Materialização os elementos do Modelo Base que estão ligados com estes Pontos de Variação CVL poderão ser removidos.

3.2.3 Criação da Árvore VSpec

Os *VSpecs* podem ser organizados como árvores onde a relação pai-filho organiza o espaço de resolução através da imposição de uma estrutura e de uma lógica para resoluções permitidas. Uma árvore de *VSpecs* possui um nó raiz que representa o ponto de partida da materialização de um produto, e os outros nós serão responsáveis para representar a variabilidade da LPS.

Esta atividade tem como objetivo a organização dos *VSpecs* definidos na Seção 3.2.2 em uma estrutura de árvore. As diretrizes para apoiar a criação da árvore *VSpec* são:

D.2.7.1 criar o *VSpec* raiz. Este *VSpec* deve ser do tipo Escolha e deverá possuir o valor *true* no atributo *isImpliedByParent*;

D.2.7.2 organizar os *VSpecs* de acordo com a estrutura de árvore, identificando *VSpecs* pai e subordinados. Ainda não fazer nenhuma ligação entre os *VSpecs*;

D.2.7.3 identificar os *VSpecs* do tipo Escolha que são obrigatórios. Estes deverão possuir o valor *true* no atributo *isImpliedByParent*. Assim deverão ser ligados ao seu elemento pai por meio de uma linha sólida;

D.2.7.4 identificar os *VSpecs* do tipo Escolha que são opcionais. Estes deverão possuir o valor *false* no atributo *isImpliedByParent*. Assim deverão ser ligados ao seu elemento pai por meio de uma linha tracejada;

D.2.7.5 identificar os *VSpecs* do tipo Variável. Estes deverão estar ligados à *VSpecs* do tipo Escolha, e o valor do atributo *isImpliedByParent* deverá ser *true*. Assim deverão ser ligados ao seu elemento pai por meio de uma linha sólida;

D.2.7.6 identificar os *VSpecs* do tipo *VClassifier*. Estes deverão ter sua multiplicidade de instâncias definido de acordo com a quantidade mínima e máxima de instâncias que deverão ser criadas durante a Materialização. Este tipo de *VSpec* deve estar ligado ao seu elemento pai por meio de uma linha sólida.

Esta atividade será exemplifica com base na Figura - 3.2. Primeiramente um *VSpec* raiz deve ser definido para a árvore de acordo com a diretriz **D.2.7.1**. O nome fornecido ao *VSpec* não tem influência na Materialização, mas deve ser representativo. Os *VSpecs* foram organizados em uma estrutura de árvore, e então foram identificados que `Acelerometro` e `GPS` seriam *VSpecs* pai (**D.2.7.2**). Foram identificados os *VSpecs* `Acelerometro` e `GPS` como obrigatórios, logo foram ligados ao *VSpec* pai por meio de uma linha sólida (**D.2.7.3**). Por fim os *VSpecs* restantes foram identificados como sendo do tipo Escolha e opcionais. Então foram ligados aos seus pais por meio de uma linha tracejada (**D.2.7.4**).

3.2.4 Criação do Modelo de Resolução

Um Modelo de Resolução da CVL é uma coleção de *VSpecs* de Resolução que resolvem os *VSpecs* de um Modelo de Variabilidade. Cada Modelo de Variabilidade pode possuir mais de um Modelo de Resolução, cada qual resultando em um produto materializado.

Um *VSpec* de Resolução resolve apenas um *VSpec* e cada tipo de *VSpec* possui seu próprio tipo de resolução. Escolhas são resolvidas por decisões positivas (verdadeiras) ou negativas (falsas). Variáveis são resolvidas pelo fornecimento de um valor. *VClassifiers* são resolvidos pelas instâncias criadas. Espelhando na organização de estruturas em árvores, os *VSpecs* de resolução também são organizados em estruturas semelhantes visando facilitar a materialização dos produtos.

A atividade Criação do Modelo de Resolução implica na criação dos *VSpecs* de resolução a partir dos *VSpecs* do Modelo de Variabilidades, e em seguida organizar as

resoluções em formato de árvore, com o objetivo de gerar Modelos de Resolução CVL que serão utilizados no Processo CVL. As diretrizes para criação dos *VSpecs* de Resolução são:

D.2.8.1 respeitar a estrutura imposta pela Árvore *VSpec* para a resolução dos *VSpecs*, seguindo uma abordagem de resolução *top-down*, ou seja, do nó raiz em direção aos nós da ponta da árvore;

D.2.8.2 criar para o *VSpec* raiz um *VSpec* de resolução do tipo Escolha. O *VSpec* raiz sempre será resolvido positivamente (*true*);

D.2.8.3 criar *VSpecs* de Resolução resolvendo de forma positiva os *VSpecs* Escolha que possuem o valor *true* no atributo *isImpliedByParent* e que sejam filhos de (a) um *VSpec* Escolha que foi resolvido positivamente (*true*) ou (b) um *VSpec VClassifier* que originou instâncias;

D.2.8.4 criar *VSpecs* de Resolução resolvendo de forma positiva ou negativa os *VSpecs* Escolha que possuem o valor *false* no atributo *isImpliedByParent* e que sejam filhos de (a) um *VSpec* Escolha que foi resolvido positivamente (*true*) ou (b) um *VSpec VClassifier* que originou instâncias, deverão ser criados *VSpecs* de resolução resolvendo estes de forma positiva ou negativa, respeitando se houver, a multiplicidade de grupo do *VSpec* pai;

D.2.8.5 criar *VSpecs* de Resolução resolvendo de forma negativa os *VSpecs* Escolha que seja filho de um *VSpec* Escolha que foi resolvido de forma negativa (*false*);

D.2.8.6 criar *VSpecs* de Resolução para os *VSpecs* Variável que seja filho de (a) um *VSpec* Escolha que foi resolvido de forma positiva (*true*) ou de (b) um *VSpec VClassifier* que originou instâncias. Aos *VSpecs* de Resolução criadas deverão ser atribuídos valores do tipo correspondente ao indicado no *VSpec*;

D.2.8.7 criar instâncias correspondentes para cada *VSpec VClassifier* que seja filho de (a) um *VSpec* Escolha que foi resolvido de forma positiva (*true*) ou (b) um *VSpec VClassifier* que originou instâncias. As instâncias criadas deverão respeitar a multiplicidade de instâncias definido pelo *VClassifier*, e para cada instância resolver todos os *VSpecs* de sua sub-árvore (*VSpecs* descendentes);

As diretrizes para organização dos *VSpecs* de Resolução em formato de árvore são:

D.2.9.1 *VSpecs* de Resolução que resolvem *VSpecs* do tipo Escolha ou Variável deverão ser organizados conforme a estrutura do Modelo de Variabilidades;

D.2.9.2 instâncias criadas a partir da resolução de *VSpecs* do tipo *VClassifiers*, deverão ter suas sub-árvores organizadas conforme a estrutura de árvore do modelo de variabilidades;

Através da Figura - 3.6, a atividade de criação do Modelo de Resolução é ilustrado. Respeitando a estrutura imposta pelo Modelo de Variabilidades, o Modelo de Resolução foi concebido. Primeiramente foi criado o *VSpec* de Resolução para o *VSpec* raiz `mini-Vant` resolvido positivamente (**D.2.8.2**). Em seguida foram definidos *VSpecs* de Resolução com valor *true* para os *VSpecs* identificados como obrigatórios, são os casos de `Acelerometro` e `GPS` (**D.2.8.3**). Então finalmente foram criados as resoluções para os *VSpecs* restantes, identificados como opcionais, respeitando a multiplicidade de grupo quando houve (**D.2.8.4**). Assim foram criados os *VSpecs* de Resolução com valores *true* ou *false* para `Acelerometro_A`, `Acelerometro_B`, `Acelerometro_C`, `GPS_A`, `GPS_B` e `Piloto_Automatico`, cada um com seu respectivo valor de resolução.

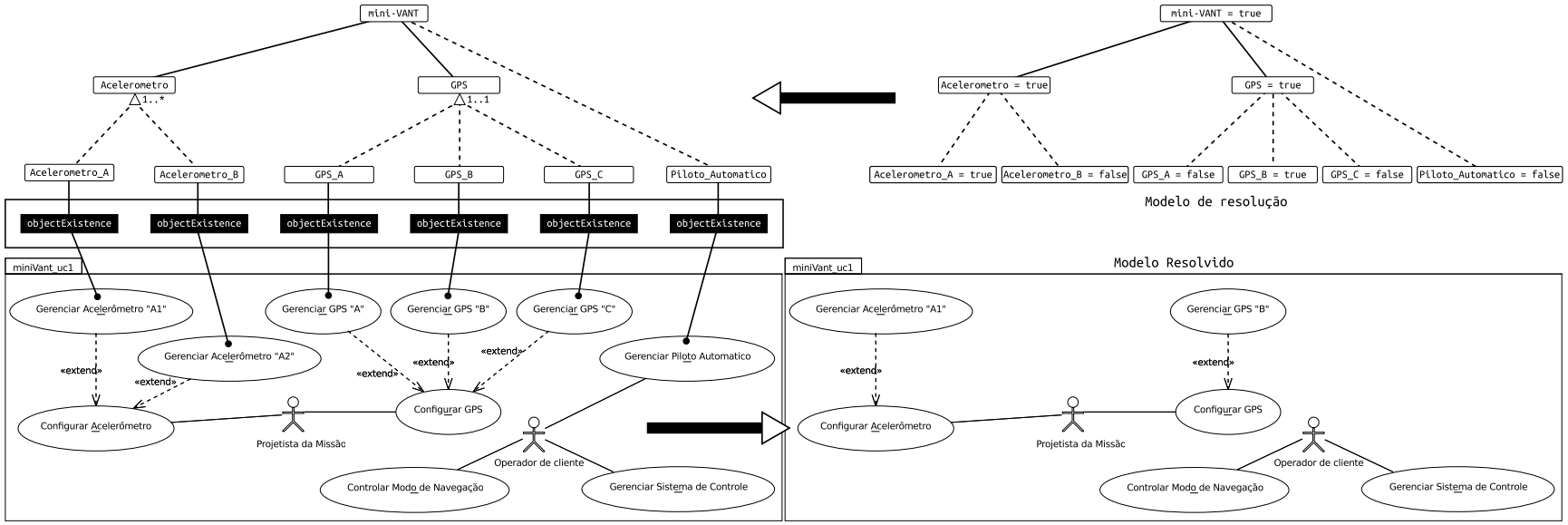

Figura 3.6: Representação da Materialização CVL - Diagrama de Caso de Uso, Modelo de Variabilidades, Modelo de Resolução e Modelo Resolvido

3.2.5 Configuração de Produtos

A solicitação de proposta enviada para a OMG para tornar a CVL uma linguagem padrão para a modelagem de variabilidades deixa claro que a CVL tem um processo chamado Processo CVL que é utilizado para a derivação de produtos. Este processo recebe como entrada o Modelo de Variabilidades, os Modelos de Resolução e o Modelo Base, e como saída, para cada Modelo de Resolução utilizado como entrada ele gera um Modelo Resolvido.

Os Modelos Resolvidos são descritos na mesma linguagem do Modelo Base e podem ser manipulados pelas mesmas ferramentas que foram utilizadas na criação dos Modelos Base.

CVL possui uma limitação, pois atualmente a única ferramenta disponível para os pesquisadores e profissionais capaz de executar o Processo CVL (Materialização) é a *CVL Tool from SINTEF* (OMG, 2014a). Esta ferramenta possui muitas limitações, como por exemplo, ela é um *plugin* do Eclipse (Eclipse Foundation, 2015), trabalha apenas com Modelos Base gerados por *plugins* específicos do Eclipse e não oferece um suporte satisfatório para modelos SysML.

Esta atividade tem como objetivo ser uma alternativa ao Processo CVL de configuração de produtos, e como vantagem, a independência de ferramentas de modelagem, deixando a decisão de escolha para o usuário. Tal atividade é específica para o diagrama de definição de blocos e os diagramas internos de blocos.

Esta atividade pode ser executada de acordo com as seguintes diretrizes:

D.2.10.1 criar uma cópia do Modelo Base, chamando-o de Modelo Resolvido;

D.2.10.2 os valores nos *VSpecs* de Resolução que resolvem *VSpecs* do tipo Variável devem ser atribuídos ao respectivo *slot* no Modelo Resolvido através do Ponto de Variação Atribuição de Valor Paramétrico;

D.2.10.3 os *VSpecs* de Resolução que resolvem *VSpecs* de Escolha, e podem afetar o Modelo Base de duas formas:

a) através dos Pontos de Variação CVL Existência de Objetos. *VSpecs* ligados a este tipo de ponto de variação, se resolvidos negativamente irão eliminar elementos do Modelo Base;

b) através dos Pontos de Variação CVL Substituição de Objetos. *VSpecs* ligados a este tipo de ponto de variação, se resolvidos de forma positiva irão realizar a substituição de um Objeto de Fixação por um Objeto de Substituição;

D.2.10.4 instâncias criadas a partir de *VClassifiers*, através dos Pontos de Variação CVL Repetíveis, devem ser adicionadas ao Modelo Base;

Vamos ilustrar a aplicação desta atividade com o auxílio dos modelos contidos na Figura - 3.6. A partir do Modelo Base foram definidos o Modelo de Variabilidades e o Modelo de Resolução. E é a partir destes três modelos que a materialização de um produto é realizada. Iniciamos o processo pela duplicação do Modelo Base e chamando-o de Modelo Resolvido (**D.2.10.1**). Isto é feito para que o Modelo Base não sofra nenhum tipo de alteração. No Modelo de Variabilidade da figura existem apenas Pontos de Variação CVL do tipo Existência de Objetos, então aplicaremos somente a diretriz **D.2.10.3** na Materialização proposta. Os *VSpecs* Escolha que são resolvidos com o valor *false*, informam para os seus respectivos Pontos de Variação CVL que os seus elementos no Modelo Resolvido deverão ser removidos. Desta forma os elementos do Modelo Base `Acelerometro_B`, `GPS_A`, `GPS_C` e `Piloto_Automatico` não irão estar contidos no produto gerado pela LPS.

Ao final deste processo teremos o Modelo Base livre de anotações ou conceitos de variabilidades. E o Modelo Resolvido poderá ser manipulado pelas mesmas ferramentas que forma utilizadas na criação e manipulação do Modelo Base.

3.3 Considerações finais

A utilização da linguagem SysML possibilitou a especificação de modelos de alto nível para sistemas embarcados, desde a atividade de análise de requisitos até a fase de projeto. Tal linguagem foi escolhida, pois ela se tornou a linguagem de modelagem padrão da OMG para os processos de desenvolvimento de sistemas embarcados.

Neste contexto, este capítulo apresentou a abordagem SyMPLES-CVL como uma alternativa para SyMPLES-SMarty. O SyMPLES-CVL utiliza a linguagem CVL, em vez da abordagem SMarty, para a representação e gerenciamento de variabilidades em LPS. As principais alterações em relação a proposta original foram: (a) não reutilização do perfil SyMPLES-ProfileVar, oriundo de SMarty; (b) não reutilização do SyMPLES-ProcessVar que adiciona conceitos de variabilidades por meio do SyMPLES-ProfileVar; (c) criação do SyMPLES-ProcessVarCvl (em substituição ao SyMPLES-ProcessVar), no qual gera modelos que representam a variabilidade nos modelos SysML gerados pelo SyMPLES-ProcessPL.

A extensão SysML contida no SyMPLES-CVL visa facilitar a especificação de alto nível de sistemas embarcados. O SyMPLES-ProfileFB define um conjunto de estereótipos

para relacionar um diagrama de blocos ou diagrama interno do bloco em SysML com as principais classes da abordagem de blocos funcionais.

Os processos definidos na abordagem definem um conjunto de atividades e diretrizes que guiam o usuário na elaboração dos artefatos da LPS e na especificação das variabilidades. O processo SyMPLES-ProcessPL auxilia na construção da infraestrutura de LPS baseada em SysML. Simultaneamente executado a ele, o processo SyMPLES-ProcessVarCvl auxilia a identificação e delimitação das variabilidades, além da resolução da variabilidade e configuração de novos produtos.

4

Avaliação experimental da abordagem SyMPLES-CVL

Um método de avaliação empírico é um conjunto de princípios organizados em torno do qual os dados empíricos são coletados e analisados (Easterbrook et al., 2008). No entanto, em virtude dos métodos de pesquisa serem adaptados a partir de várias áreas de pesquisa, não existe uma terminologia uniforme para descrição dos métodos experimentais e não há um consenso sobre a forma de distinção entre eles.

Dessa forma, selecionar um método de estudo para pesquisa empírica não é algo trivial porque os benefícios e desafios na utilização de cada um, ainda, não estão bem descritos e aceitos na literatura. Também, não há método melhor ou pior do que outro, eles simplesmente são melhores em alguns aspectos e piores em outros (Dennis e Valacich, 2001).

Segundo Wohlin et al. (2000) existem basicamente quatro métodos relevantes para a condução de estudos empíricos na área de engenharia de software (ES): científico, de engenharia, empírico e analítico. No entanto, Travassos et al. (2002) sugerem que a abordagem mais apropriada para a experimentação na área de ES seja o método experimental.

O método experimental sugere um modelo e, em seguida desenvolve um método que pode ser qualitativo e/ou quantitativo, aplica um procedimento empírico (experimento, quase experimento, etc.), mede e analisa, avalia o modelo e repete novamente o processo, ou seja, segue uma abordagem orientada à melhoria evolucionária (Travassos et al., 2002).

Neste contexto, foi realizado um estudo experimental comparando a efetividade da abordagem SyMPLES-SMarty (Silva, 2012) com a abordagem proposta por este trabalho, o SyMPLES-CVL, como segue nas próximas seções.

4.1 Metodologia e Planejamento Experimental

No estudo experimental foi comparada a efetividade entre a abordagem SyMPLES-SMarty e a proposta nesta dissertação, a abordagem SyMPLES-CVL.

A efetividade, neste trabalho é definida como o quão bem uma abordagem permite a identificação e a representação de variabilidades e seus elementos em modelos SysML de LPS. Ela foi proposta com base nos estudos de Basili e Selby (1987) e Wohlin et al. (2000). Para o cálculo da efetividade, os participantes interpretaram a LPS *WeatherStation* (Beuche e Dalgarno, 2006) que foi modelada com as abordagens SyMPLES-SMarty ou SyMPLES-CVL, e em seguida responderam 10 questões relacionadas a LPS que interpretaram. O número de respostas corretas de cada participante foi utilizado para encontrar a efetividade da abordagem avaliada. Esses números foram então aplicados na equação de efetividade, apresentada na Equação 4.1:

$$efetividade(z) = nVarC \tag{4.1}$$

onde:

- `z` é a abordagem de gerenciamento de variabilidade;
- `nVarC` é o número de respostas corretas no formulário de coleta de dados de acordo com a abordagem `z`;

Os itens que foram estabelecidos para o planejamento e a execução experimental são elencados a seguir. Esses itens foram definidos com base nos estudos propostos por Juristo e Moreno (2001) e Wohlin et al. (2000).

1. Planejamento
    - Contexto local
    - Treinamento
    - Projeto Piloto
    - Participantes
    - Instrumentação
    - Formulação de Hipóteses
    - Variáveis Dependentes
    - Variáveis Independentes

- Capacidade Aleatória
- Classificação em Bloco
- Balanceamento
- Mecanismo de Análise
- Validade Interna
- Validade Externa
- Validade de *Constructo*
- Validade de Conclusão

2. Execução

- Seleção de participantes
- Procedimento de participação
- Instrumentação

A Figura - 4.1 apresenta o resumo da metodologia utilizada nas etapas realizadas no estudo experimental, com foco principal na análise dos resultados, apresentando uma síntese dos testes estatísticos das amostras, que, conduzidos pelo teste de normalidade, seguem para um teste paramétrico, já que a amostra deste estudo foi considerada normal.

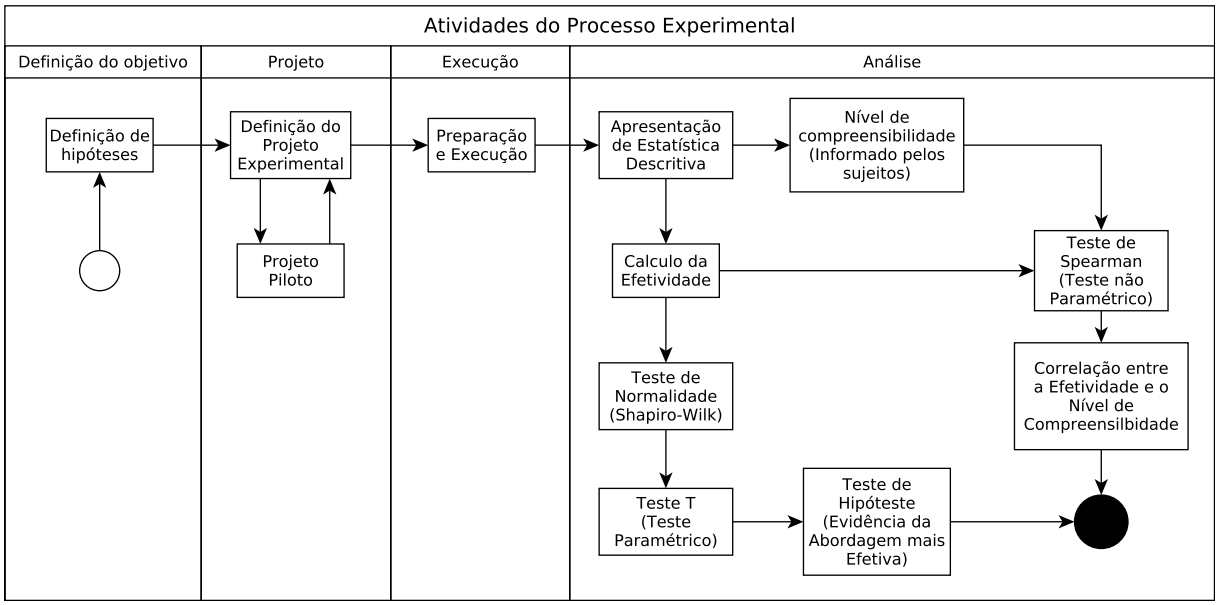

Figura 4.1: Atividades do processo experimental

Primeiramente, os dados coletados na execução experimental referentes à efetividade, são resumidos em tabelas juntamente com sua estatística descritiva.

Em segundo lugar, são submetidos ao teste de normalidade *Shapiro-Wilk*. De acordo com o resultados do teste (amostra normal ou não normal), um teste estatístico para verificar se a diferença entre os valores de efetividade para as duas abordagens que são considerados no estudo são significativos, podendo inferir se a hipótese nula pode ser rejeitada com nível de significância.

O teste para verificar a diferença estatística entre a efetividade total da abordagem SyMPLES-SMarty e a abordagem SyMPLES-CVL foi o Teste-T (Wohlin et al., 2000), um teste paramétrico.

Testada e evidenciada a abordagem mais efetiva no experimento, uma correlação entre a efetividade obtida para cada participante e o nível de compreensibilidade da abordagem de cada um deles foi realizada, por meio de respostas coletadas do formulário de coleta de dados.

Os participantes do estudo foram divididos em 2 grupos, onde cada grupo se especializou em uma das duas abordagens comparadas. Então, para o questionário do nível de compreensibilidade, apenas uma das seguintes perguntas estava disponível para os participantes:

Grupo A: Na sua opinião, qual o nível de compreensibilidade da *LPS WeatherStation* representada com a abordagem SyMPLES-SMarty?

Grupo B: Na sua opinião, qual o nível de compreensibilidade da *LPS WeatherStation* representada com a abordagem SyMPLES-CVL?

Para ambos os grupos, as respostas da questão Nível de compreensibilidade foram as mesmas:

(4) Muito compreensível

(3) Compreensível

(2) Incompreensível

(1) Muito Incompreensível

Assim, as respostas para a pergunta receberam um peso 1-4, caracterizando uma escala ordinal (Appolinario, 2012). Por meio desses pesos, foi calculada a correlação permitindo indicar o nível de correlação obtido e assim, uma análise das evidências é realizada para averiguar se o nível de compreensibilidade indicada pelos participantes teve alguma influência no resultado da aplicação da respectiva abordagem.

As próximas seções explicam os procedimentos realizados para a obtenção e a análise dos resultados do estudo experimental realizado.

4.2 Definição do estudo experimental

A definição do estudo experimental utiliza uma notação baseada em *Goal/Question/Metric* (GQM), pois a mesma fornece uma abordagem sistemática para definir as metas do estudo experimental (Basili et al., 1986). A notação consiste na elaboração dos objetivos (`G`), formulação das questões (`Q`) e definição das métricas (`M`). Dessa forma, o objetivo do estudo experimental é apresentado a seguir:

O objetivo do experimento foi
Comparar as abordagens SyMPLES-SMarty e SyMPLES-CVL
Com o propósito de identificar a mais efetiva
Em respeito da capacidade de identificação e representação de variabilidades em Linhas de Produto de Software em Diagramas de Definição de Blocos da linguagem SysML
Do ponto de vista de arquitetos de Linha de Produto de Software
No contexto de estudantes, profissionais e professores da área de Engenharia de Software da Universidade Estadual de Maringá, Universidade Paranaense (UNIPAR) e indústrias da região da cidade de Maringá/PR.

O modelo GQM é definido com duas questões de pesquisa (Q.P.1 e Q.P.2):

Q.P.1 Qual abordagem é mais efetiva na identificação e representação de variabilidades no contexto de Diagramas de Definição de Blocos da linguagem SysML em LPS?

Q.P.2 Existe uma correlação entre o nível de compreensibilidade indicado pelos participantes e a quantidade de acertos no Formulário de Coleta de Dados?

4.3 Planejamento do estudo experimental

Os itens correspondentes ao planejamento experimental são apresentados nesta seção.

Contexto local Uma *LPS Printer*, *LPS Mindstorms* e uma *LPS WeatherStation*, que foram levadas em consideração para a aplicação das abordagens SyMPLES-SMarty e SyMPLES-CVL visando a representação de variabilidades em Diagramas de Definição de Blocos da notação SysML.

Treinamento Os participantes foram treinados com os conceitos essenciais de LPS e gerenciamento de variabilidades. Durante os treinamento foram utilizados a *LPS*

Printer e a *LPS Mindstorms*. Os participantes foram divididos em dois grupos. O grupo A recebeu um treinamento específico com os conceitos da abordagem SyMPLES-SMarty, enquanto que, o grupo B recebeu um treinamento específico com os conceitos da abordagem SyMPLES-CVL. Após o treinamento foi solicitado aos participantes que respondessem os formulários de coleta de dados.

Projeto Piloto Antes da real execução do estudo experimental, um estudo piloto foi realizado com o intuito de avaliar a instrumentação a ser utilizada no estudo. Para isso, um participante com conhecimentos básicos na abordagem SMarty e linguagem CVL foi utilizado, sendo atribuídos a ele os mesmos instrumentos do estudo experimental. Em seguida, foram feitas correções na instrumentação com base nas anotações e resultados do projeto piloto. Os dados obtidos pelo estudo piloto não foram utilizados para compor o estudo.

Seleção de participantes Foram selecionados para estudo um total de 2 professores doutores, 4 alunos mestrandos e 14 profissionais graduados, totalizando 20 participantes da região de Maringá. Todos os participantes pertencem a área de Engenharia de Software e possuem um conhecimento mínimo na modelagem de sistemas. Após as sessões de treinamento, cada participante sentiu-se familiarizado com os conceitos essenciais de gerenciamento de variabilidades.

Instrumentação Cada participante recebeu os seguintes documentos:

- Termo de Consentimento Livre e Esclarecido (TCLE) do estudo experimental;
- questionário de caracterização, em que os participantes deveriam indicar seu conhecimento acadêmico, área de conhecimento e experiência, nível de experiência com a notação SysML, a abordagem *SMarty* e a linguagem CVL;
- conceitos essenciais para gerenciamento de variabilidades em LPS;
- descrição da *LPS Printer* e seu Diagrama de Definição de Blocos em SysML com as variabilidades representadas;
- descrição da *LPS Mindstorms* e seu Diagrama de Definição de Blocos em SysML sem as variabilidades representadas; e
- descrição da *LPS WeatherStation* e seu Diagrama de Definição de Blocos em SysML com as variabilidades representadas.

Os participantes do grupo A receberam também os seguintes documentos:

- descrição da abordagem SyMPLES-SMarty; e
- formulário de coleta de dados da *LPS WheaterStation* para a abordagem SyMPLES-SMarty.

Para o grupo B, os participantes também receberam:

- descrição da abordagem SyMPLES-CVL; e
- formulário de coleta de dados da *WheaterStation LPS* para a abordagem SyMPLES-CVL.

O projeto piloto levou à identificação do elevado tempo utilizado pelos participantes ao receberem as duas abordagens para serem aplicadas. Assim, foi definida a divisão de dois grupos de participantes, para a execução experimental e consequente divisão dos instrumentos para tal. Um grupo de participantes recebeu as especificações da abordagem SyMPLES-SMarty e outro da abordagem SyMPLES-CVL, o mesmo ocorreu para a divisão dos Formulários Experimentais, divididos especificamente para cada abordagem.

No estudo foram utilizados a (i) *LPS Printer*, (ii) *LPS Mindstorms* e a (iii) *LPS WeatherStation*. Elas foram entregues nesta ordem para os participantes, mas cada uma foi utilizada para um propósito distinto durante a execução do experimento. A primeira foi utilizada para demonstrar a abordagem em questão; a segunda foi utilizada para o treinamento na respectiva abordagem, e finalmente a terceira foi utilizada para a coleta de dados do experimento.

Os participantes foram convidados para participar do estudo, e em seguida os que aceitaram foram divididos convenientemente em dois grupos. Assim foram agendadas as seções, de acordo com a disponibilidade dos participantes.

Cada uma das sessões seguiu os seguintes passos:

Passo 1: Apresentação da pesquisa, bem como a proposta e os objetivos, seguido pelo preenchimento do TCLE, no qual deixava claro os direitos e deveres do participante no decorrer da pesquisa;

Passo 2: Apresentação dos conceitos básicos de LPS e Gerenciamento de Variabilidades, utilizando Linhas de Produto utilizadas pela indústria;

Passo 3: Apresentação da abordagem SyMPLES-SMarty/SyMPLES-CVL, utilizando a *LPS Printer*.

Passo 4: Aplicação pelos participantes da abordagem SyMPLES-SMarty/SyMPLES-CVL à *LPS Mindstorms*. O objetivo deste passo era esclarecer possíveis dúvidas dos participantes em relação à LPS, Gerenciamento de Variabilidades e à abordagem utilizada para o grupo;

Passo 5: Aplicação do Formulário de Coleta de Dados. Neste último passo os participantes interpretaram a *LPS WeatherStation* e responderam 10 questões acerca da LPS e finalmente informaram o nível de compreensibilidade da abordagem.

A seguir são apresentados os últimos itens correspondentes ao planejamento experimental:

Formulação de Hipóteses As seguintes hipóteses foram testadas neste estudo:

- **Hipótese nula (H_0)**: ambas, SyMPLES-SMarty e SyMPLES-CVL são igualmente efetivas em termos de representação de variabilidades em Diagramas de Definição de Blocos da linguagem SysML;
  $H_0 : \mu(\text{efetividade(SyMPLES-SMarty)}) = \mu(\text{efetividade(SyMPLES-CVL)})$

- **Hipótese alternativa (H_1)**: a abordagem SyMPLES-SMarty é menos efetiva que a abordagem SyMPLES-CVL;
  $H_1 : \mu(\text{efetividade(SyMPLES-SMarty)}) < \mu(\text{efetividade(SyMPLES-CVL)})$

- **Hipótese alternativa (H_2)**: a abordagem SyMPLES-SMarty é mais efetiva que a abordagem SyMPLES-CVL;
  $H_2 : \mu(\text{efetividade(SyMPLES-SMarty)}) > \mu(\text{efetividade(SyMPLES-CVL)})$

Variáveis Independentes As versões da abordagem *SyMPLES*, é um fator com dois tratamentos (SyMPLES-SMarty e SyMPLES-CVL) e a LPS, é um fator com um único tratamento (*LPS WeatherStation*).

Variáveis Dependentes A efetividade calculada para cada abordagem de gerenciamento de variabilidades (SyMPLES-SMarty e SyMPLES-CVL), representa a variável dependente no estudo experimental.

A Equação 4.1 apresenta a maneira que é calculado a variável dependente efetividade.

Capacidade Aleatória A seleção dos participantes não foi aleatória dentro do universo de voluntários, pois é muito restrita. A capacidade aleatória teve lugar na atribuição da abordagem de gerenciamento de variabilidade (SyMPLES-SMarty ou SyMPLES-CVL) para cada grupo de participantes.

Classificação em Bloco Como a aplicação do experimento se dá pela aplicação de duas abordagens, em que a população foi dividida em dois grupos, um que recebeu treinamento sobre a abordagem SyMPLES-SMarty e o outro a abordagem SyMPLES-CVL, balanceados pelo nível de formação, obtido pelo Questionário de Caracterização, onde também foi obtido entre outras informações, o nível de conhecimento em LPS e Gerenciamento de Variabilidades.

Balanceamento As tarefas foram realizadas em igual número para um número igual de participantes.

Mecanismo de Análise O estudo experimental realizado analisa, com base na opinião de estudantes, professores e profissionais da área da Engenharia de Software e que possuem conhecimento mínimo na modelagem de sistemas, os mecanismos de análise foram: o cálculo da efetividade e a aplicação de teste estatístico para verificar se as diferenças entre a efetividade da abordagem SyMPLES-SMarty e SyMPLES-CVL são significativas, permitindo evidenciar qual a mais efetiva.

4.4 Execução do estudo experimental

Nesta seção, são apresentados os passos executados para a obtenção dos resultados do estudo experimental.

Para este estudo foram selecionados um total de 20 participantes, entre eles alunos mestrandos e graduados, professores de universidades e profissionais da indústria, todos com no mínimo conhecimento básico em modelagem de sistemas.

Procedimento de Participação: Os procedimentos executados no estudo experimental se resumem como segue:

(a) os participantes foram recebidos na sala para a execução experimental, bem como dispostos em carteiras para a realização do experimento;

(b) o experimentador distribui aos participantes um conjunto de documentos (instrumentação), de acordo com os blocos balanceados pelo nível de formação dos participantes:

- o TCLE do estudo experimental;
- os conceitos essenciais sobre LPS e gerenciamento de variabilidade;
- os conceitos essenciais da abordagem SyMPLES-SMarty/SyMPLES-CVL;
- a descrição e diagramas SysML com variabilidades representadas em SyMPLES-SMarty/SyMPLES-CVL da *LPS Printer*;
- a descrição e diagramas SysML sem variabilidades representadas da *LPS Mindstorms*;
- a descrição e diagramas SysML com variabilidades representadas em SyMPLES-SMarty/SyMPLES-CVL da *LPS WeatherStation*; e
- formulário de coleta de dados com 10 questões acerca da identificação de variabilidades na *LPS WeatherStation*, mais a questão envolvendo o nível de compreensibilidade da abordagem aplicada para o grupo.

(c) os participantes leem cada um dos documentos entregues;

(d) o experimentador ministrou um treinamento sobre LPS, Gerenciamento de Variabilidades e abordagem em questão utilizando a *LPS Printer*;

(e) os participantes, após o treinamento sanam dúvidas quanto os documentos que foram entregues a eles;

(f) quando todos os participantes do grupo se sentem familiarizados com a abordagem, eles identificam e representam as variabilidades especificadas na descrição da *LPS Mindstorms* utilizando os conceitos da abordagem que receberam (SyMPLES-SMarty ou SyMPLES-CVL);

(g) participantes respondem as questões do formulário de coleta de dados, e também informam o nível de compreensibilidade da abordagem; e finalmente

(h) o experimentador recolhe a instrumentação distribuída para posterior correção e análise dos resultados.

É importante ressaltar que o procedimento de participação dos grupos A e B foi idêntico, e em nenhum momento os participantes de um grupo tiveram acesso à documentação utilizada no outro grupo, evitando assim algum tipo de comparação das abordagens por parte dos participantes.

Após a execução e a resolução dos formulários experimentais, os dados coletados são apresentados em tabelas respectivas para suas abordagens, bem como analisados

utilizando métodos estatísticos apropriados, como os definidos no início da seção atual. A análise dos dados e os testes estatísticos são apresentados na seção a seguir.

4.5 Análise e Interpretação dos resultados

Os resultados obtidos na aplicação das abordagens SyMPLES-SMarty e SyMPLES-CVL para a *LPS WeatherStation* foram submetidos aos passos que seguem:

- análise e interpretação dos dados coletados para as abordagens SyMPLES-SMarty e SyMPLES-CVL, apresentados na Tabela - 4.1, por meio do teste de normalidade Shapiro-Wilk e o teste de hipóteses Teste-T; e
- análise e interpretação da correlação entre a efetividade das abordagens e as respostas fornecidas pelos participantes no questionário de caracterização, com a aplicação da técnica de correlação de Spearman.

Tabela 4.1: Resultados coletados e estatística descritiva (SyMPLES-SMarty e SyMPLES-CVL)

Abordagem SyMPLES-SMarty		
#	Efetividade	Nível de Compreensibilidade
1	6	3
2	10	3
3	9	4
4	8	3
5	6	3
6	7	3
7	9	4
8	9	4
9	9	4
10	7	3
Média	8,00	
Desvio Padrão	1,34	
Mediana	8,50	

Abordagem SyMPLES-CVL		
#	Efetividade	Nível de Compreensibilidade
1	7	4
2	6	3
3	7	3
4	9	4
5	7	3
6	10	4
7	10	4
8	8	4
9	10	4
10	9	3
Média	8,30	
Desvio Padrão	1,42	
Mediana	8,50	

A Figura - 4.2 apresenta os Box Plots com os valores da Efetividade (Tabela - 4.1) para as abordagens SyMPLES-SMarty e SyMPLES-CVL.

4.5.1 Efetividade das abordagens (Q.P.1)

Teste de Normalidade dos Dados: o teste de normalidade de Shapiro-Wilk (Shapiro e Wilk, 1965) foi aplicado à amostra (Tabela - 4.1) para cada uma das abordagens, conforme apresentado nos histogramas das Figura - 4.3 e Figura - 4.4, indicando os resultados que seguem:

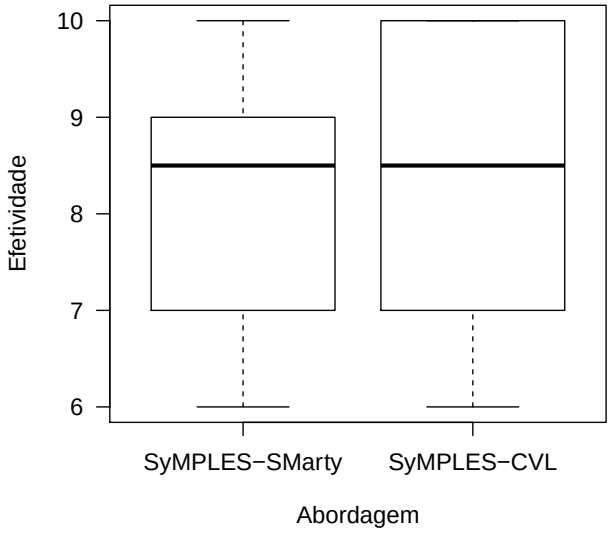

Figura 4.2: Box Plot Efetividade para as abordagens SyMPLES-SMarty e SyMPLES-CVL

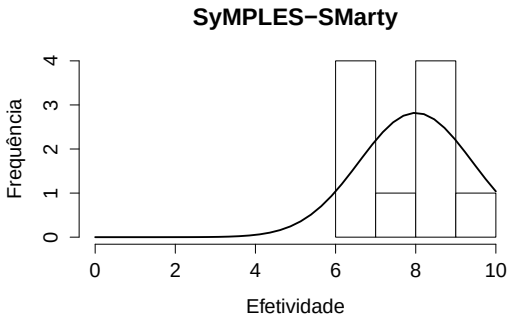

Figura 4.3: Histograma da Amostra da Efetividade para a abordagem SyMPLES-SMarty

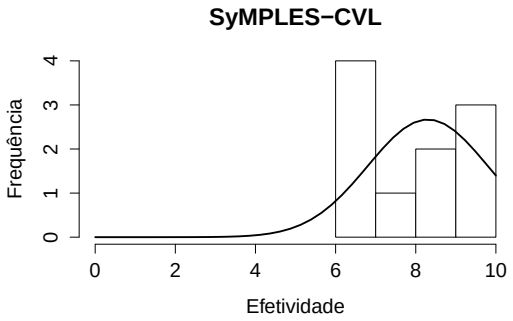

Figura 4.4: Histograma da Amostra da Efetividade para a abordagem SyMPLES-CVL

A abordagem SyMPLES-SMarty ($N = 10$): Com valor médio (μ) $8,0$, valor de desvio padrão de (σ) $1,34$, a efetividade para a abordagem SyMPLES-SMarty sobre a *LPS WeatherStation* foi $p = 0,1576$ para o teste de normalidade de *Shapiro-Wilk*.

No teste *Shapiro-Wilk* com uma amostra de tamanho *(N)* 10 com 95% de nível de significância ($\alpha = 0,05$), $p = 0,1576$ ($0,1576 > 0,05$) e valor calculado de $W = 0,8872 > W = 0,8420$, a amostra é considerada normal.

A abordagem SyMPLES-CVL ($N = 10$): Com valor médio (μ) $8,3$, valor de desvio padrão de (σ) $1,42$, a efetividade para a abordagem SyMPLES-CVL sobre a *LPS WeatherStation* foi $p = 0,1376$ para o teste de normalidade de *Shapiro-Wilk*.

No teste *Shapiro-Wilk* com uma amostra de tamanho *(N)* 10 com 95% de nível de significância ($\alpha = 0,05$), $p = 0,1376$ ($0,1376 > 0,05$) e valor calculado de $W = 0,8820 > W = 0,8420$, a amostra é considerada normal.

Após a análise da normalidade das amostras, foi constatado que a distribuição da efetividade para as abordagens SyMPLES-SMarty e SyMPLES-CVL possuem uma distribuição normal, ambos resultados baseados no teste de normalidade *Shapiro-Wilk*. Isto implica a determinação de um teste de hipóteses paramétrico para permitir a análise e a coleta de evidências, baseada nos resultados da amostra para indicar se é possível rejeitar a hipótese nula deste estudo.

Teste-T para as amostras SyMPLES-SMarty e SyMPLES-CVL

Este teste estatístico de hipóteses pode ser aplicado para amostras independentes e emparelhadas, com tamanho inferior a 30 (Appolinario, 2012; Wohlin et al., 2000). Neste estudo, as amostras de SyMPLES-SMarty e SyMPLES-CVL são independentes. Como cada uma das amostras tem tamanho menor que 30, e foram identificadas como normais, as hipóteses a seguir foram definidas no Teste-T, para verificação:

- **Hipótese nula (H_0)**: os valores da efetividade das abordagens SyMPLES-SMarty e SyMPLES-CVL não possuem uma diferença significante.

  H_0: $\mu(efetividade(SyMPLES\text{-}SMarty)) = \mu(efetividade(SyMPLES\text{-}CVL))$;

- **Hipótese alternativa (H_1)**: os valores da efetividade das abordagens SyMPLES-SMarty e SyMPLES-CVL possuem um diferença significante.

  H_0: $\mu(efetividade(SyMPLES\text{-}SMarty)) \neq \mu(efetividade(SyMPLES\text{-}CVL))$;

Primeiro obtivemos o valor de T, permitindo a identificação do intervalo inserido na tabela estatística T (*student*). Este valor é calculado usando-se a média das amostras SyMPLES-SMarty ($\mu_1 = 8,0$) e SyMPLES-CVL ($\mu_2 = 8,3$), o valor do desvio padrão de ambas amostras ($\sigma_1 = 1,34$ e $\sigma_2 = 1,42$), e o tamanho da amostra, para cada uma delas, que corresponde a 10 para ambas ($N = 10$). Obtendo assim o valor $t_{calculado} = -0,4611$.

Levando em consideração o tamanho da amostra ($N = 10$), obtivemos o grau de liberdade ($df = 18$), que, combinado com o valor t indica qual o valor de p na tabela T deve ser selecionado. O valor p é utilizado para aceitar ou rejeitar a hipótese nula do teste *T-test*.

Ao pesquisar o índice $df = 18$ e definir o valor t na tabela T (*student*), encontramos um valor para o t crítico de $2,101$ ($t_{critico} = 2,101$), com um nível de significância (α) de $0,05$. Assim, comparando o $t_{critico}$ com o $t_{calculado}$, a hipótese nula H_0 (não há diferenças entre os valores) não pode ser rejeitada. Isso significa que com 95% de significância a abordagem SyMPLES-SMarty e a abordagem SyMPLES-CVL não possuem diferença para representar variabilidades em LPS no nível de Diagramas de Definição de Blocos para este estudo experimental.

4.5.2 Correlação entre as Efetividades das Abordagens e o nível de compreensibilidade dos participantes (Q.P.2)

A correlação, entre o nível de compreensibilidade informado pelos participantes e a efetividade calculada para as abordagens, é realizado para identificar a possível influência do nível de compreensibilidade, nos resultados de efetividade obtidos.

Nível de compreensibilidade para participantes da abordagem SyMPLES-SMarty

Para uma amostra de tamanho (N) 10, com média (μ) $3,4$, desvio padrão de (σ) $0,4898$, o nível de compreensibilidade dos participantes foi $p = 0,0001687$ para o teste *Shapiro-Wilk*.

No teste *Shapiro-Wilk*, para uma amostra de tamanho 10 com 95% de significância ($\alpha = 0,05$), $p = 0,0001687$ ($0,0001687 < 0,05$) e valor calculado $W = 0,6405 < W = 0,8420$ a amostra é considerada não normal.

Nível de compreensibilidade para participantes da abordagem SyMPLES-CVL

Para uma amostra de tamanho (N) 10, com média (μ) $3,6$, desvio padrão de (σ) $0,4898$, o nível de compreensibilidade dos participantes foi $p = 0,0001687$ para o teste *Shapiro-Wilk*.

No teste *Shapiro-Wilk*, para uma amostra de tamanho 10 com 95% de significância ($\alpha = 0,05$), $p = 0,0001687$ ($0,0001687 < 0,05$) e valor calculado $W = 0,6405 < W = 0,8420$ a amostra é considerada não normal.

Correlação de Spearman

Esta técnica é aplicada para identificar a correlação entre a efetividade de cada abordagem (SyMPLES-SMarty e SyMPLES-CVL) e o nível de compreensibilidade informado pelos participantes no formulário de coleta de dados durante as sessões. A Equação 4.2 apresenta a fórmula para calcular o coeficiente ρ de Spearman, onde `n` representa o tamanho da amostra:

$$\rho = 1 - \frac{6}{n(n^2-1)} \sum_{i=1}^{n} d_i^2 \tag{4.2}$$

A Tabela - 4.2 apresenta os dados aplicados no cálculo da correlação de Spearman para as abordagens SyMPLES-SMarty e SyMPLES-CVL e o nível de compreensibilidade das abordagens estudadas. Os valores das colunas `ra` e `rb` foram obtidos após a ordenação decrescente dos valores da efetividade e do seu respectivo nível de compreensibilidade. Nas situações em que ocorreram empates de posições, a média entre essas foi calculada.

Tabela 4.2: Correlação de Spearman entre a Efetividade das Abordagens SyMPLES-SMarty e SyMPLES-CVL e o Nível de Compreensibilidade

Abordagem SyMPLES-SMarty						
#	Efetividade	r_a	Compreensibilidade	r_b	$d(r_a-r_b)$	d^2
1	6	1,50	3	3,50	-2,00	4,00
2	10	10,00	3	3,50	6,50	42,25
3	9	7,50	4	8,50	-1,00	1,00
4	8	5,00	3	3,50	1,50	2,25
5	6	1,50	3	3,50	-2,00	4,00
6	7	3,50	3	3,50	0,00	0,00
7	9	7,50	4	8,50	-1,00	1,00
8	9	7,50	4	8,50	-1,00	1,00
9	9	7,50	4	8,50	-1,00	1,00
10	7	3,50	3	3,50	0,00	0,00
					Σ d^2:	56,50
					ρ:	0,6576

Abordagem SyMPLES-CVL						
#	Efetividade	r_a	Compreensibilidade	r_b	$d(r_a-r_b)$	d^2
1	7	3,00	4	7,50	-4,50	20,25
2	6	1,00	3	2,50	-1,50	2,25
3	7	3,00	3	2,50	0,50	0,25
4	9	6,50	4	7,50	-1,00	1,00
5	7	3,00	3	2,50	0,50	0,25
6	10	9,00	4	7,50	1,50	2,25
7	10	9,00	4	7,50	1,50	2,25
8	8	5,00	4	7,50	-2,50	6,25
9	10	9,00	4	7,50	1,50	2,25
10	9	6,50	3	2,50	4,00	16,00
					Σ d^2:	53,00
					ρ:	0,6788

O cálculo para cada correlação de acordo com a abordagem é mostrado nas Equações 4.3 e 4.4.

$$\left.\begin{aligned} \rho(Corr.SyMPLES-SMarty) &= 1 - \tfrac{6}{10(10^2-1)} * 56,5 \\ \rho(Corr.SyMPLES-SMarty) &= 1 - 0,3424 = 0,6576 \end{aligned}\right\} \tag{4.3}$$

$$\left.\begin{array}{l} \rho(Corr.SyMPLES-CVL) = 1 - \frac{6}{10(10^2-1)} * 53 \\ \rho(Corr.SyMPLES-CVL) = 1 - 0,3212 = 0,6788 \end{array}\right\} \quad (4.4)$$

Assim, foram obtidos os valores que seguem para ρ e analisados por meio da escala de classificação de *Spearman*, apresentada na Figura - 4.5:

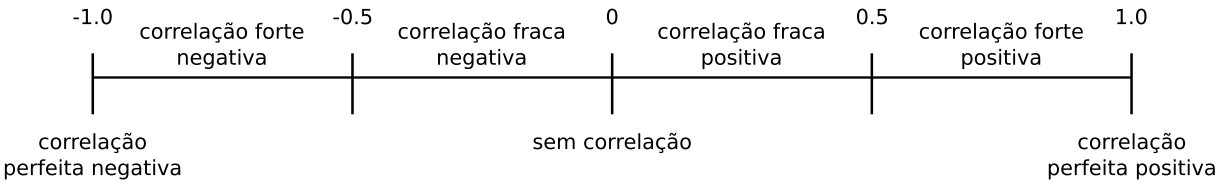

Figura 4.5: Escala de Correlação de Spearman. Adaptado de (Spearman, 1987)

- *Abordagem SyMPLES-SMarty:* $\rho = 0,66$ - Correlação positiva forte;
- *Abordagem SyMPLES-CVL:* $\rho = 0,68$ - Correlação positiva forte;

Por meio da análise dos resultados obtidos pelas correlações de Spearman, podemos observar que todos eles se encontram dentro da mesma faixa de escala de classificação mostrado na Figura - 4.5. Nota-se a diferença não significativa entre o valor obtido para a abordagem SyMPLES-SMarty com relação à abordagem SyMPLES-CVL.

Estes resultados indicam que, para ambas abordagens citadas neste estudo, existe uma forte correlação positiva entre o nível de compreensibilidade indicada pelos participantes e a sua quantidade de acertos. E, apesar de ser uma abordagem anotativa e outra composicional, a diferença não é significativa para os valores obtidos em cada uma das abordagens.

4.6 Avaliação de Validade do Estudo Experimental

Wohlin et al. (2000) propõe uma classificação dos tipos de ameaças a validade, que são as ameaças à validade de conclusão, validade interna, validade de construção e a validade externa. As ameaças consideradas e os procedimentos adotados para suavizar e sanar alguns dos problemas encontrados durante a execução do estudo experimental são apresentados nesta seção.

4.6.1 Ameaças à Validade de Conclusão

A validade de conclusão refere-se à capacidade do estudo experimental em relacionar os tratamentos e os resultados, permitindo a generalização dos resultados (Wohlin et al., 2000).

A ameaça com maior impacto sobre a pesquisa deve-se ao tamanho da amostra utilizada, um total de 20 participantes para o estudo experimental. Assim, para garantir a generalização dos resultados, e consequente aumentar o poder estatístico desses, devem ser feitas replicações com uma quantidade maior de participantes em estudos futuros.

Esta ameaça ocorre pela grande dificuldade de se obter voluntários bem qualificados para participar dos estudos, visto que estes não receberam quaisquer benefícios. Além disso, há a necessidade da obtenção de participantes com nível de conhecimento mínimo em SysML, LPS, Gerenciamento de Variabilidade e de preferência estarem inseridos na área de engenharia de software.

A participação de voluntários com pouco conhecimento nas áreas necessárias para a realização de estudos que usam um conjunto de técnicas ou abordagens, de métodos e procedimentos, são motivos de vieses que levam a resultados inconsistentes, em que não expressam com fidedignidade o que a proposta avaliada se propõe. Logo, a obtenção de pessoas com o mínimo de conhecimento necessário torna-se fator limitador, que consequentemente reflete em uma amostragem menor, e culmina na necessidade de replicações para coletar evidências que corroborem com os resultados das execuções originais (Kitchenham et al., 2010; Marcolino, 2014).

4.6.2 Ameaças à Validade de Constructo

A validade de *constructo* refere-se à capacidade do estudo experimental em relacionar a instrumentação com os participantes do estudo e a teoria que está sendo provada (Wohlin et al., 2000).

Para o experimento conduzido, a variável dependente efetividade foi definida e testada com base no projeto piloto e estudos de Basili e Selby (1987) e Wohlin et al. (2000). Sendo coletada na execução experimental, por meio de um formulário de coleta de dados, que contém os modelos da *LPS WeatherStation* e 10 questões onde os participantes deveriam responder, além do nível de compreensibilidade das abordagens.

Em relação aos participantes, receberam o treinamento para diminuir a diferença de conhecimento em LPS e variabilidades e possuem nível de conhecimento necessário em SysML para a realização das tarefas, garantindo que as respostas coletas fossem consideradas significativas.

Em relação aos participantes, estes receberam um treinamento para diminuir a diferença de conhecimento em LPS e variabilidades. Todos responderam no questionário de caracterização que possuem um conhecimento mínimo em SysML, garantindo que as respostas coletadas no formulário de coleta de dados fossem significantes.

4.6.3 Ameaças à Validade Interna

No estudo apresentado, as seguintes dificuldades foram encontradas:

Diferenças entre os participantes: como as amostras foram consideradas pequenas, variações entre as habilidades dos participantes foram reduzidos por meio da realização de uma sessão de treinamento e tarefas na mesma ordem. Os participantes tinham praticamente os mesmos níveis de conhecimento em modelagem SysML e conceitos de gerenciamento de variabilidade;

Efeitos de Fadiga: o projeto piloto teve uma duração de aproximadamente 120 minutos sem intervalo, e o participante foi treinado com as duas abordagens comparados neste estudo, o que causou fadiga no participante. Esta ameaça foi minimizada pela divisão dos participantes em dois grupos, cada grupo participando de sessões de apenas uma abordagem. Assim o tempo máximo das sessões foi de aproximadamente 70 minutos.

Influência entre participantes a comunicação entre participantes pode influencia-los. Acreditou-se que esta ameaça foi minimizada porque os participantes ficaram sob a supervisão de um observador.

4.6.4 Ameaças à Validade Externa

A validade externa do estudo visa medir a sua capacidade de refletir o mesmo comportamento, em outros grupos de participantes e profissionais da indústria, além dos que participaram do estudo (Marcolino, 2014).

Duas ameaças à validade externa foram identificadas:

Instrumentação: a utilização de LPS modeladas para o experimento colocam em risco a validade externa. Isto foi minimizado pelo uso de LPS utilizadas em outros estudos, por exemplo a *LPS WeatherStation*. Além disso replicações devem ser conduzidas utilizando LPS utilizadas pela indústria.

Participantes: a seleção por conveniência e o número restrito de participantes dificulta a generalização dos resultados. Acreditou-se que esta ameaça foi reduzida pela seleção de participantes, descrita na Seção 4.3. Contudo, mais experimentos devem ser conduzidos levando em consideração participantes da indústria, permitindo a generalização dos resultados.

4.7 Apresentação e Empacotamento do Estudo Experimental

A instrumentação, bem como planilhas de análise de dados e o diário experimental, reportando dificuldades e visando transferir o conhecimento dos experimentadores (Mendonça et al., 2008; Shull et al., 2004), estão disponíveis em: `https://github.com/chiquitto/symples-smarty-cvl`.

O estudo experimental aqui apresentado contribui com a difusão de procedimentos experimentais e demais conhecimentos utilizados (Brooks et al., 1996; Shull et al., 2004), além de permitir sua utilização para a análise de efetividade de outras abordagens de gerenciamento de variabilidade em LPS do domínio de sistemas embarcados.

4.8 Considerações finais

Variabilidades são elementos cruciais para LPS, logo abordagens para gerenciamento destas são fundamentais para garantir que produtos derivados consistentes de uma LPS sejam construídos seguindo rigorosamente as definições necessárias (Marcolino, 2014).

A avaliação experimental conduzida apresentou evidências da efetividade das abordagens SyMPLES-SMarty e SyMPLES-CVL, para os Diagramas de Definição de Blocos. Os resultados indicaram, que apesar das abordagens SyMPLES-SMarty e SyMPLES-CVL usarem notações anotativas (abordagens SMarty) e composicionais (linguagem CVL), respectivamente, as duas oferecem a princípio o mesmo nível de efetividade para a identificação e representação de variabilidades em LPS. Além disso, a correlação entre o nível de efetividade calculado e o nível de compreensibilidade informado pelos participantes possui uma forte correlação positiva para as duas abordagens.

Assim sendo, novos experimentos devem ser conduzidos para coletar evidências para identificar a abordagem mais efetiva entre SyMPLES-SMarty e SyMPLES-CVL. A fim de reduzir as ameaças à validade externa, novos estudos com participantes da indústria

devem ser realizados com LPS reais. Reduzindo as ameaças à validade externa estaremos facilitando a adoção das abordagens SyMPLES pela indústria.

5

Avaliação qualitativa da abordagem SyMPLES-CVL

Este capítulo apresenta uma avaliação qualitativa discutindo os benefícios e as limitações de SMarty e CVL aplicado ao SyMPLES. De acordo com a avaliação experimental (Capítulo 4), não foi possível evidenciar que SyMPLES-CVL é mais efetiva que SyMPLES-SMarty. Sendo assim, o estudo experimental não forneceu evidências de qual a abordagem mais efetiva no processo de identificação e representação de variabilidades em modelos SysML.

Esta avaliação qualitativa foi considerada necessária para complementar os resultados fornecidos pela avaliação experimental. Os resultados desse capítulo são baseados em critérios que evidenciam as diferenças das abordagens comparadas, facilitando a identificação das vantagens e limitações.

Inicialmente é feita a avaliação por meio de uma comparação das vantagens e limitações dentre SMarty e CVL. Em seguida, os resultados obtidos são discutidos e exibidos em uma tabela.

5.1 Comparação entre SyMPLES-SMarty e SyMPLES-CVL

Nesta seção é feita uma comparação das abordagens SyMPLES-SMarty e SyMPLES-CVL com base nos critérios citados por Kästner e Apel (2008). Os parâmetros selecionados enfatizam as diferenças entre as abordagens comparadas. Os critérios são exibidos na Tabela - 5.1. Essa avaliação baseia-se tanto em pesquisas na área de LPS e gerenciamento de variabilidades quanto na própria experiência.

5.1.1 Rastreabilidade de características

Rastreabilidade de características é a capacidade de rastrear uma característica do modelo de características (espaço de domínio) para a implementação (espaço de solução) (Kästner e Apel, 2008). A rastreabilidade pode ser utilizada para verificar a consistência entre o modelo de características e os modelos de arquitetura (Satyananda et al., 2007), por exemplo.

De acordo com Haugen (2012), os Modelos de Variabilidade CVL são baseados nos Modelos de Características (Kang et al., 1990). Pela análise dos Modelos de Variabilidades CVL gerados pelo SyMPLES-ProcessVarCvl, pode-se notar a semelhança entre eles. Esta semelhança entre os Modelos de Variabilidades para Modelos Base diferentes pode auxiliar no rastreamento de elementos entre modelos. Por exemplo, por meio do Modelo de Variabilidades da Figura - 3.3 podemos identificar que o requisito `Gerenciar Piloto Automatico` atende ao caso de uso `Gerenciar Piloto Automatico` da Figura - 3.2. Além disso, existem estudos como Satyananda et al. (2007) e Riebisch (2004) propondo soluções que apoiam a rastreabilidade a partir dos Modelos de Características.

Na abordagem SMarty a representação das variabilidades é feita por meio de estereótipos, o que não fornece um meio para rastrear elementos entre modelos, principalmente a partir do modelo de características. Há evidências que o meta-atributo *realizes*, aplicado à metaclasse `UML Comment`, oferece suporte para tal característica, mas o SMarty não discute de forma clara este assunto.

5.1.2 Independência de linguagem

A independência de linguagem das abordagens pode ser um fator decisivo para a sua adoção, pois, dependendo do ponto de vista, pode ser visto como uma vantagem ou uma limitação da abordagem.

A abordagem SMarty contém um conjunto de estereótipos e meta-atributos para representar variabilidades em modelos UML. SMarty usa a notação UML e seu mecanismo de perfil para fornecer uma extensão da UML e permitir a representação de conceitos de variabilidade (OliveiraJr et al., 2013a). SMarty também pode ser aplicada a modelos de linguagens que estendem a UML, como é o caso da SysML.

Em comparação, a CVL pode ser utilizada para representar variabilidades sobre qualquer tipo de linguagem baseado no metamodelo MOF (inclui-se UML e SysML neste grupo). Temos como exemplo de aplicação da CVL o trabalho de Svendsen et al. (2010) aplicando CVL sobre modelos TCL (*Train Control Language*) para gerenciar as variabilidades.

5.1.3 Adoção de LPS

A indústria é muito cautelosa na adoção de abordagens composicionais, porque este tipo de abordagem tem muita influência sobre a base existente de códigos e processos. Somente depois de um cuidadoso planejamento, que este tipo de abordagem é adotada (Bass et al., 2003).

Em contra partida, as abordagens anotativas tendem a ser adotadas com maior facilidade, porque introduzem ferramentas que causam pouco impacto sobre os processos existentes nas organizações (Kästner e Apel, 2008). Assim, as abordagens anotativas, entre elas o SMarty, se favorecem do fato de causarem menos impacto na adoção pelas organizações se comparado com as abordagens composicionais.

5.1.4 Suporte de ferramentas

As organizações de desenvolvimento de software usam ferramentas para apoiar suas atividades diárias para transformar um conjunto de requisitos em um produto em perfeitas condições para o cliente. Uma grande quantidade de ferramentas CASE estão disponíveis para ajudar a automatizar as atividades de análise, projeto, implementação e manutenção de produtos. O desafio é escolher e usar ferramentas com sabedoria para apoiar os objetivos da organização e as necessidades técnicas dos desenvolvedores (SEI, 2012).

O suporte por parte de ferramentas provavelmente seja o maior influenciador na adoção de uma abordagem de LPS.

Pelo fato de SMarty ser uma abordagem anotativa que identifica e representa as variabilidades por meio de estereótipos em modelos UML/SysML, as ferramentas que apoiam esta abordagem são todas aquelas no qual oferecem suporte para a criação de modelos baseados no metamodelo UML 2 definido pela OMG em OMG (2015b).

A CVL por sua vez necessita de ferramentas específicas para o desenvolvimento de modelos. Dessa forma, existem poucas ferramentas que apoiam o desenvolvimento em CVL, sendo que apenas uma apoia completamente o processo de materialização de produtos da CVL.

5.2 Discussão da Avaliação Qualitativa

Na seção anterior foram feitas avaliações entre o SyMPLES-SMarty e a SyMPLES-CVL. Uma síntese dos resultados é exibida a seguir:

Rastreabilidade de características A avaliação indicou que a CVL pode permitir a rastreabilidade de elementos entre o Modelo de Características e os Modelos Base por meio dos Modelos de Variabilidade CVL;

Independência de linguagem SMarty expressa os conceitos de variabilidades em modelos UML e extensões como por exemplo a linguagem SysML. Em CVL os conceitos podem ser expressos em qualquer modelo baseado no metamodelo MOF, incluindo UML e SysML. Isto permite que a CVL possa ser aplicada à uma maior variedade de modelos;

Adoção LPS Segundo Kästner e Apel (2008) as abordagens anotativas tendem a ser adotadas com uma maior facilidade. Desta forma, SMarty tende a ser adotada com uma maior facilidade e com mais frequência que a linguagem CVL.

Suporte de ferramentas Das ferramentas apresentas, apenas uma delas realiza a derivação de produtos, a *CVL Tool from SINTEF*. Em contraste o SMarty se beneficia do fato se existir inúmeras (além daquelas exibidas neste trabalho) ferramentas que dão suporte para a modelagem na linguagem UML.

Os resultados obtidos indicam que as duas abordagens possuem vantagens e limitações, e que uma possível integração entre elas poderia combinar as suas vantagens. Isto é discutido em um estudo de Kästner e Apel (2008) que propõe a integração de abordagens anotativas e composicionais.

A Tabela - 5.1 exibe os resultados em um formato tabular, organizando os resultados obtidos. Para cada característica discutida, existe uma indicação (`X`) de qual abordagem possui mais vantagens em relação à sua concorrente. No final são exibidas a quantidade de vezes que uma abordagem ofereceu mais benefícios se comparado com a outra abordagem.

Tabela 5.1: Resultados da Avaliação Qualitativa

Critério	SMarty	CVL
Rastreabilidade de características		X
Independência de linguagem		X
Adoção LPS	X	
Suporte de ferramentas	X	
Total	2	2

CVL levou vantagens sobre SMarty em relação ao critério Rastreabilidade de características. Independência de linguagens é uma característica muito importante, mas no contexto das abordagens SyMPLES somente a linguagem SysML é utilizada, diminuindo

a importância deste critério pois SMarty e CVL possuem suporte à linguagem SysML. O critério Adoção LPS indicou que a abordagem SMarty poderia ser mais facilmente adotada se comparada com a CVL. O critério Suporte de Ferramentas está relacionado com o apoio que a abordagem irá receber caso ela seja adotada. Neste sentido, SMarty leva vantagem pois foi melhor avaliada em dois critérios enquanto que a CVL apenas em um critério, deixando o SyMPLES-SMarty com uma vantagem sobre o SyMPLES-CVL.

5.3 Considerações finais

Este capítulo apresentou uma avaliação qualitativa comparando as abordagens SyMPLES-SMarty e SyMPLES-CVL. No início foi realizada uma breve introdução das abordagens anotativas e composicionais, necessários para o entendimento da avaliação proposta. Em seguida foram apresentadas as características que seriam utilizadas na avaliação, e com base nestas características foram discutidas as vantagens e limitações de SMarty e CVL. Após isto, as avaliações foram sintetizadas e exibidas na Tabela - 5.1. Por fim, os resultados encontrados foram discutidos de um forma geral, indicando a abordagem que foi melhor avaliada durante as comparações. Os resultados desta avaliação complementam os resultados da avaliação experimental, de forma a indicar pontos positivos e negativos das duas abordagens SyMPLES.

6

Conclusão

Nos trabalhos que apresentam abordagens de especificação de sistemas embarcados, é recorrente o argumento de que é necessário encontrar técnicas para lidar com a complexidade da atual geração de sistemas embarcados (Silva, 2012).

A abordagem de LPS tem se mostrado efetiva, oferecendo muitos benefícios à geração de produtos, como redução de custos e tempo de desenvolvimento, diminuição de perdas e riscos, acelerando o retorno do investimento (Pohl et al., 2005). Segundo Marcolino (2014), esses benefícios citados podem ser obtidos por meio de um gerenciamento das variabilidades, presente no núcleo de artefatos, realizado de forma objetiva e sistematizada.

As várias abordagens de LPS existentes podem ser divididas em dois grupos: anotativas e composicionais. O gerenciamento de variabilidades de SyMPLES-SMarty é realizado com SMarty. Esta dissertação apresentou uma abordagem alternativa para SyMPLES-SMarty, a SyMPLES-CVL, visando a identificação e representação de variabilidades em CVL. SMarty e CVL pertencem, respectivamente, aos grupos das abordagens anotativas e composicionais.

SyMPLES-CVL foi baseada em SyMPLES-SMarty, mas utiliza a linguagem CVL para gerenciar e representar variabilidades sobre modelos SysML. A abordagem proposta inclui o SyMPLES-ProfileFB e o SyMPLES-ProcessPL originais do SyMPLES-SMarty. Para que fosse possível a substituição do SMarty por CVL, foi criado um novo processo, denominado de SyMPLES-ProcessVarCvl.

O SyMPLES-ProcessVarCvl é um processo executado simultaneamente ao processo SyMPLES-ProcessPL e define um conjunto de atividades e diretrizes que auxiliam o usuário a identificar, delimitar e representar as variabilidades em modelos SysML com o apoio da linguagem CVL. A CVL possui uma limitação em relação às ferramentas

que a apoiam. Então, para tornar o processo de derivação de produtos independente de ferramentas foi definida a atividade Configuração de Produtos. Esta atividade tem por objetivo sistematizar a geração de produtos a partir dos Modelos SysML, Modelos de Características e Modelos da CVL.

As abordagens SyMPLES-SMarty e SyMPLES-CVL foram então comparadas por meio de uma estudo experimental. Os resultados indicaram que as duas oferecem a princípio o mesmo nível de efetividade para a identificação e representação de variabilidades em LPS.

Assim, viu-se a necessidade de um novo estudo para complementar os resultados do estudo experimental. Uma avaliação qualitativa foi conduzida comparando critérios importantes das abordagens SyMPLES-SMarty e SyMPLES-CVL. A avaliação foi realizada por meio de pesquisas na área de LPS e na própria experiência. Podemos afirmar que os resultados dessa avaliação foram relevantes para indicar vantagens e limitações das duas abordagens comparadas.

6.1 Contribuições

Neste trabalho, procurou-se oferecer soluções para alguns dos problemas encontrados no domínio de sistemas embarcados tais como *time-to-market*, qualidade de sistemas e reuso de artefatos por meio da utilização de técnicas de reúso da Engenharia de Software como LPS. Estudos e pesquisas foram realizados em áreas relacionadas, o que resultou na proposta de uma abordagem alternativa para a identificação e representação de variabilidades com a linguagem CVL, que foi avaliada e comparada com a abordagem que a originou. Essa nova abordagem possui um processo que guia o usuário na identificação e delimitação de variabilidades no desenvolvimento de uma LPS. Nesse sentido é possível ressaltar as seguintes contribuições deste trabalho:

- permitir que as variabilidades da abordagem SyMPLES-SMarty sejam representadas pela linguagem CVL, uma linguagem proposta para a OMG como um padrão para gerenciamento de variabilidades;
- oferecer um processo que guia o usuário na identificação e delimitação de variabilidades em modelos SysML por meio da linguagem CVL;
- apresentar uma comparação entre uma abordagem anotativa e uma abordagem composicional;

- oferecer uma visão geral das ferramentas disponíveis para a modelagem UML/SysML, identificação e representação de variabilidades a partir das abordagens SyMPLES-SMarty e SyMPLES-CVL.

6.2 Limitações

Este estudo apresenta algumas limitações e são exibidas a seguir:

- este trabalho leva em consideração apenas os diagramas de casos de uso, requisitos, definição de bloco e os diagramas internos de blocos como artefatos para especificar uma LPS para sistemas embarcados; outros recursos da linguagem, entre eles o diagrama paramétrico, não foram levados em consideração;
- este trabalho leva em consideração apenas uma parte da linguagem CVL. Isto é devido ao fato de ser utilizado apenas os construtores e operados básicos, necessários para a realização deste trabalho;
- uma quantidade pequena de LPS foi desenvolvida utilizando a abordagem proposta. É necessário que se desenvolvam outras LPS utilizando a abordagem apresentada para que as atividades possam ser refinadas e melhoradas e se tornem em uma abordagem mais confiável;
- a única ferramenta CVL que executa o processo de Materialização não recebe mais atualizações e também não materializa produtos SysML. Assim, o SyMPLES-ProcessVarCvl não possui um método automatizado para a derivação de produtos.

6.3 Trabalhos futuros

Os trabalhos futuros estão relacionados com as limitações descritas na seção anterior. Com o objetivo de dar continuidade ao trabalho aqui desenvolvido, de forma a complementá-lo e ampliá-lo, algumas pesquisas que podem ser seguidas são as seguintes:

- integrar as abordagens SyMPLES-SMarty e SyMPLES-CVL propondo uma terceira abordagem na tentativa de combinar as vantagens e reduzir as limitações;
- considerar diagramas da linguagem SysML que não foram abordados neste trabalho, assim como outros elementos;

- considerar a linguagem CVL por completo, aplicando todos os conceitos da linguagem na elaboração de LPS para Sistemas Embarcados;
- desenvolver um processo de transformação que envolva o SysML, CVL e Modelos Simulink;
- desenvolvimento de uma ferramenta para automatizar a atividade de configuração de produtos, descrita na Seção 3.2.5;
- realizar novos estudos permitindo encontrar novas evidências sobre a efetividade das abordagens SyMPLES e refinamento dos processos SyMPLES-ProcessPL e SyMPLES-ProcessVarCvl;
- desenvolvimento de uma metodologia para a criação de repositórios para armazenamento de versões das LPS desenvolvidas nos estudos das abordagens SyMPLES.

REFERÊNCIAS

3TU.Federatie 3TU MSc in Embedded Systems. 2015.
Disponível em http://www.3tu.nl/es/en/programme/overview/ (Último Acesso em 18/06/2015)

Appolinario, F. *Metodologia Da Ciência: Filosofia E Prática Da Pesquisa.* 2ª edição ed. São Paulo: Cengage, 240 p., 2012.

Bak, K.; Novakovic, M.; Passos, L. *Exemplar of Automotive Architecture with Variability.* Relatório Técnico, University of Waterloo, Waterloo, 2010.
Disponível em http://gsd.uwaterloo.ca/node/320 (Último Acesso em 18/12/2013)

Basili, V. R.; Selby, R. W. Comparing the Effectiveness of Software Testing Strategies. *IEEE Trans. Softw. Eng.*, v. 13, n. 12, p. 1278–1296, 1987.

Basili, V. R.; Selby, R. W.; Hutchens, D. H. Experimentation in Software Engineering. *IEEE Trans. Softw. Eng.*, v. 12, n. 7, p. 733–743, 1986.

Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice.* 2 ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

Bassi, L.; Secchi, C.; Bonfe, M.; Fantuzzi, C. A SysML-Based Methodology for Manufacturing Machinery Modeling and Design. *Mechatronics, IEEE/ASME Transactions on*, v. 16, n. 6, p. 1049–1062, 2011.

Berger, A. S. *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques.* CMP Books. Lawrence, Kansas: CMP Book, 237 p., 2002.

Beuche, D.; Dalgarno, M. *Software product line engineering with feature models.* Software Acumen, 2006.

Bosch, J. Software Product Families and Populations. In: *2nd Groningen Workshop on Software Variability Management*, Groningen, 2004.

Brooks, A.; Daly, J.; Miller, J.; Roper, M.; Wood, M. *Replication of Experimental Results in Software Engineering.* Relatório Técnico, IEEE Transactions on Software Engineering, 1996.

Buschmann, M.; Bange, J.; Vörsmann, P. MMAV-a miniature unmanned aerial vehicle (Mini-UAV) for meteorological purposes. In: *Proceedings of the 16th Symposium on Boundary Layers and Turbulence*, Portland, 2004, p. 21–27.

Chen, L.; Babar, M. A.; Ali, N. Variability Management in Software Product Lines : A Systematic Review. *Proceedings of the 13th International Software Product Line Conference*, p. 81–90, 2009.

Combemale, B.; Barais, O.; Alam, O.; Kienzle, J. Using CVL to operationalize product line development with reusable aspect models. *Proceedings of the VARiability for You Workshop on Variability Modeling Made Useful for Everyone - VARY '12*, p. 9–14, 2012.

Contieri Junior, A. C.; Correia, G. G.; Colanzi, T. F.; Gimenes, I. M. S.; OliveiraJr, E.; Ferrari, S.; Masiero, P. C.; Garcia, A. F. Extending UML Components to Develop Software Product-Line Architectures : Lessons Learned. In: *Software Architecture*, v. 6903, Springer Berlin Heidelberg, p. 130–138, 2011.

Czarnecki, K.; Antkiewicz, M.; Kim, C. Model-driven software product lines. In: *OOPSLA '05 Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2005, p. 126–127.

Dennis, A. R.; Valacich, J. S. Conducting Research in Information Systems. *Communications of the Association for Information Systems*, v. 7, n. 5, p. 1–41, 2001.

Easterbrook, S.; Singer, J.; Storey, M.-A.; Damian, D. Selecting Empirical Methods for Software Engineering Research. *Guide to Advanced Empirical Software Engineering*, p. 285–311, 2008.

Eclipse Foundation Graphical Modeling Framework (GMF). 2005. Disponível em https://wiki.eclipse.org/Graphical_Modeling_Framework (Último Acesso em 03/04/2015)

Eclipse Foundation Eclipse Modeling Framework Project. 2008. Disponível em http://www.eclipse.org/modeling/emf/ (Último Acesso em 25/04/2015)

Eclipse Foundation Eclipse Sirius. 2013.
Disponível em https://www.eclipse.org/sirius/ (Último Acesso em 01/05/2015)

Eclipse Foundation Eclipse Luna: Papyrus 1.0 Release. 2014.
Disponível em http://www.eclipse.org/community/eclipse_newsletter/2014/april/article2.php (Último Acesso em 20/07/2014)

Eclipse Foundation Eclipse Luna. 2015.
Disponível em https://eclipse.org/ (Último Acesso em 22/02/2015)

Falvo Junior, V.; Duarte Filho, N. F.; OliveiraJr, E.; Barbosa, E. F. A contribution to the adoption of software product lines in the development of mobile learning applications. In: *Frontiers in Education Conference (FIE), 2014 IEEE*, Madrid: Proceedings of the 2014 Frontiers in Education Conference, 2014, p. 1–8.

Fragal, V. H. *Engenharia de aplicação para sistemas embarcados : transformando especificações SysML em Simulink.* Dissertação de Mestrado, Universidade Estadual de Maringá - UEM, Maringá, 2013.

Galster, M.; Weyns, D.; Tofan, D.; Michalik, B.; Avgeriou, P. Variability in Software Systems - A Systematic Literature Review. *IEEE Transactions on Software Engineering*, v. 40, n. 3, p. 282–306, 2014.

Group, M. R. CVL 2 Tool. 2014.
Disponível em http://modelbased.net/tools/cvl-2-tool/ (Último Acesso em 24/03/2014)

Haugen, Ø. Common Variability Language (CVL) OMG Revised Submission. 2012. Disponível em http://www.omgwiki.org/variability/doku.php (Último Acesso em 20/03/2014)

Heverhagen, T. *Integration of languages for programmable controllers into the Unified Modeling Language through Function Block Adapters.* Tese de Doutoramento, University of Duisburg-Essen, Duisburg-Essen, 2003.

Jacobson, I.; Griss, M.; Jonsson, P. *Software reuse: architecture process and organization for business success.* ACM Press books. ACM Press, 497 p., 1997.

Juristo, N.; Moreno, A. M. *Basics of Software Engineering Experimentation.* Springer, 2001.

Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S. *Feature-Oriented Domain Analysis (FODA): Feasibility Study.* Relatório Técnico November, Carnegie-Mellon University Software Engineering Institute, 1990.

Kästner, C.; Apel, S. Integrating compositional and annotative approaches for product line engineering. In: *Proc. GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering*, Passau, Alemanha, 2008, p. 35–40.

Kästner, C.; Apel, S.; Kuhlemann, M. Granularity in Software Product Lines. In: *Proceedings of the 30th International Conference on Software Engineering*, New York, NY, USA: ACM, 2008, p. 311–320.

Kitchenham, B.; Sjø berg, D. I. K.; Brereton, O. P.; Budgen, D.; Dybå, T.; Höst, M.; Pfahl, D.; Runeson, P. Can We Evaluate the Quality of Software Engineering Experiments? In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA: ACM, 2010, p. 2:1—-2:8 (*ESEM '10*, v.1).

Lee, E. A.; Seshia, S. A. *Introduction to Embedded Systems: A Cyber-physical Systems Approach.* Electrical Engineering & Computer Sciences. Lulu.com, 519 p., 2011.

Lee, K.; Kang, K. C.; Lee, J. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: *7th International Conference on Software Reuse*, Austin, 2002, p. 62–77.

van der Linden, F.; Schmid, K.; Rommes, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer-Verlag New York, Inc., 334 p., 2007.

Lykins, H.; Friedenthal, S.; Meilich, A. Adapting UML for an Object Oriented Systems Engineering Method (OOSEM). In: *Proceedings of the INCOSE International Symposium*, Minneapolis, MN, 2000, p. 15–20.

Marcolino, A. S. *Avaliação experimental da abordagem SMarty para gerenciamento de variabilidade em linhas de produto de software baseadas em UML.* Dissertação de Mestrado, Universidade Estadual de Maringá, 2014.

MATHWORKS Real - Time Workshop Data Sheet. 2007.
Disponível em http://www.mathworks.com/products/rtw (Último Acesso em 07/14/2014)

MATHWORKS Simulink: Simulation and Model-Based Design. 2011.
Disponível em http://www.mathworks.com/products/simulink (Último Acesso em 04/03/2014)

MENDONÇA, M. G.; MALDONADO, J. C.; OLIVEIRA, M. C. F. D.; CARVER, J.; FABBRI, C. P. F.; SHULL, F.; TRAVASSOS, G. H.; HÖHN, E. N.; BASILI, V. R. A Framework for Software Engineering Experimental Replications. In: *13th IEEE International Conference on Engineering of Complex Computer Systems*, Belfast, Irlanda do Norte, 2008, p. 203–212.

NASCIMENTO, A. S.; RUBIRA, C. M. F.; BURROWS, R.; CASTOR, F. A Model-Driven Infrastructure for Developing Product Line Architectures Using CVL. In: *VII Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, Brasília, Brasil, 2013, p. 119–128.

OBEO UML Designer. 2013.
Disponível em http://www.umldesigner.org/ (Último Acesso em 01/05/2015)

OBEO Atlas Transformation Language. 2015.
Disponível em http://www.obeo.fr/en/products/eclipse-projects (Último Acesso em 25/04/2015)

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

OLIVEIRAJR, E.; GIMENES, I. M. S.; MALDONADO, J. C. Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science*, v. 16, n. 17, p. 2374–2393, 2010.
Disponível em http://dblp.uni-trier.de/db/journals/jucs/jucs16

OLIVEIRAJR, E.; GIMENES, I. M. S.; MALDONADO, J. C.; MASIERO, P. C.; BARROCA, L. Systematic Evaluation of Software Product Line Architectures. *Journal of Universal Computer Science*, v. 19, n. 1, p. 25–52, 2013a.

OliveiraJr, E.; Pazin, M. G.; Gimenes, I. M. S.; Kulesza, U.; Aleixo, F. A. SMartySPEM: A SPEM-based approach for variability management in software process lines. In: *Lecture Notes in Computer Science*, v. 7983 LNCS de *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 169–183, 2013b.

OMG MetaObject Facility (MOF). 2013.
Disponível em http://www.omg.org/mof/ (Último Acesso em 27/03/2014)

OMG CVL Tool from SINTEF. 2014a.
Disponível em http://www.omgwiki.org/variability/doku.php/doku.php?id=cvl_tool_from_sintef (Último Acesso em 01/06/2014)

OMG Object Constraint Language (OCL). 2014b.
Disponível em http://www.omg.org/spec/OCL/ (Último Acesso em 11/08/2014)

OMG Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). 2015a.
Disponível em http://www.omg.org/spec/QVT/ (Último Acesso em 22/03/2015)

OMG Unified Modeling Language (UML). 2015b.
Disponível em http://www.omg.org/spec/UML/Current (Último Acesso em 30/04/2015)

Pohl, K.; Böckle, G.; van der Linden, F. J. *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 473 p., 2005.

PolarSys Group PolarSys. 2005.
Disponível em https://www.polarsys.org/ (Último Acesso em 01/05/2015)

Riebisch, M. Supporting evolutionary development by feature models and traceability links. In: *Proceedings 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2004)*, Brno, República Checa: IEEE Computer Society Press, 2004, p. 370–377.

Rodrigues, E.; Passos, L.; Teixeira, L. On the Requirements and Design Decisions of an In-House Component-Based SPL Automated Environment. In: *The 26th International Conference on Software Engineering and Knowledge Engineering*, Vancouver: SEKE 2014, 2014.

Satyananda, T. K.; Lee, D.; Kang, S.; Hashmi, S. I. Identifying Traceability between Feature Model and Software Architecture in Software Product Line using Formal

Concept Analysis. In: *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on*, San Francisco, California, USA, 2007, p. 380–388.

Scheidt, R. F.; Vilain, P.; Dantas, M. A. R. Modeling a distributed environment for a petroleum reservoir engineering application with software product line. *Journal of Physics: Conference Series*, v. 540, n. 1, p. 12008, 2014.

SEI Software Engineering Institute. 2012.
Disponível em http://www.sei.cmu.edu/ (Último Acesso em 15/04/2014)

SEI A Framework for Software Product Line Practice, Version 5.0. 2014.
Disponível em http://www.sei.cmu.edu/productlines/frame_report/index.html (Último Acesso em 06/04/2014)

Shapiro, S. S.; Wilk, M. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika*, v. 52, n. 3/4, p. 591–611, 1965.

Shull, F.; Mendoncça, M. G.; Basili, V.; Carver, J.; Maldonado, J. C.; Fabbri, S.; Travassos, G. H.; Ferreira, M. C. Knowledge-Sharing Issues in Experimental Software Engineering. *Empirical Softw. Engg.*, v. 9, n. 1-2, p. 111–137, 2004.

Silva, R. F. *SyMPLES : Uma Abordagem de Desenvolvimento de Linha de Produto para Sistemas Embarcados baseada em SysML.* Dissertação de Mestrado, Universidade Estadual de Maringá, 2012.

Silva, R. F.; Fragal, V. H.; OliveiraJr, E.; Gimenes, I. M. S.; Oquendo, F. SyMPLES - A SysML-based Approach for Developing Embedded Systems Software Product Lines. In: *ICEIS (2)'13*, 2013, p. 257–264.

Sjostedt, C.-J.; Torngren, M.; Shi, J.; Chen, D.-J.; Ahlsten, V. Mapping Simulink to UML in the design of embedded systems:Investigating scenarios and transformations. In: *4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems*, Paderborn, 2008, p. 137–160.

Spearman, C. The proof and measurement of association between two things. *The American journal of psychology*, v. 100, n. 3-4, p. 441–471, 1987.

Svendsen, A.; Zhang, X.; Lind-tviberg, R.; Fleurey, F. Developing a Software Product Line for Train Control : A Case Study of CVL. *SPLC*, p. 106–120, 2010.

TopCased TopCased Project. 2013.
Disponível em http://www.topcased.org/ (Último Acesso em 01/05/2015)

Travassos, G. H.; Gurov, D.; do Amaral, E. A. G. *Introdução à Engenharia de Software Experimental*. Relatório Técnico, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2002.

Vasilevskiy, A. *Conquering overlapping fragments in CVL*. Dissertação de Mestrado, University of Oslo, 2013.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

Wong, W. E.; Debroy, V.; Surampudi, A.; Kim, H.; Siok, M. F. Recent Catastrophic Accidents: Investigating How Software was Responsible. In: *Proceedings of the 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, Singapura: IEEE, 2010, p. 14–22.

Apêndice A

Este apêndice contém a descrição de todos os estereótipos definidos para o perfil SyMPLES-ProfileFB. Tais estereótipos estabelecem um comportamento para os blocos SysML, por meio da relação com a classe equivalente de blocos funcionais.

A.1 Descrição dos estereótipos do SyMPLES-ProfileFB

«BusCreator»

Cria um barramento de sinais. Um bloco do tipo *bus creator* combina um conjunto de sinais de entrada em um barramento com um sinal de saída.

«BusSelector»

Seleciona os sinais de um determinado barramento. Um bloco do tipo *bus selector* replica um subconjunto de sinais provenientes de um barramento de entrada.

«Constant»

Gera o valor de uma constante, definida como parâmetro. Um bloco do tipo constante gera um valor constante do tipo real ou complexo. O bloco pode gerar valores escalares, vetores ou matrizes.

«DataTypeConversion»

Converte um sinal de entrada para um outro determinado tipo de dado especificado como parâmetro.

«Display»

Mostra o valor da entrada sobre o ícone do bloco.

«Enabler»

Indica uma porta de entrada com a propriedade de habilitar ou desabilitar a funcionalidade de um determinado bloco.

«Function»

Executa o código de uma função armazenada no bloco.

«Gain»

Multiplica o valor da entrada por uma constante definida como parâmetro no bloco. A entrada pode ser um valor do tipo escalar, vetor ou matriz.

«Integrator»

Um bloco do tipo *integrator* calcula o valor da integral de suas entradas em relação ao tempo corrente. A equação A.1 representa a saída do bloco `y` como uma função das suas entradas `u` e tempo inicial $\mathtt{y}_0$, onde `y` e `u` são funções de vetores do tempo corrente `t`.

$$\mathtt{y(t)} = \int_{\mathtt{t0}}^{\mathtt{t}} \mathtt{u(t)dt} + \mathtt{y}_0 \tag{A.1}$$

«ManualSwitch»

Um bloco do tipo *manual switch* alterna, por meio da alteração de um parâmetro, qual das suas entradas será selecionada como saída.

«Mean»

Encontra o valor médio de uma sequência de valores de entrada.

«Memory»

Um bloco do tipo *memory* fornece o valor de um sinal armazenado em um intervalo de tempo anterior.

«MinMaxRunningResettable»

Determina o valor mínimo ou máximo de um sinal em um determinado intervalo de tempo.

«Mux»

Um bloco do tipo *mux* combina os sinais de suas entradas em um único vetor de sinais como saída.

«Product»

Produz como saída o resultado da multiplicação ou divisão dos valores de entrada do bloco.

«RateTransition»

Um bloco do tipo *rate transition* gerencia a transferência de dados entre blocos que transmitem sinais em taxas diferentes.

«Scope»

Exibe os sinais gerados durante o processo de simulação.

«SignalSpecification»

Possibilita especificar os atributos do sinal conectado ao bloco, como tipo do dado e tipo numérico, entre outros.

«StopSimulation»

Um bloco do tipo *stop simulation* termina o processo de simulação do sistema, quando o valor do sinal de entrada é diferente de zero.

«SubSystem»

Representa um bloco de subsistema cuja execução é iniciada a partir de uma entrada externa.

«Sum»

Um bloco do tipo *sum* permite somar elementos escalares ou vetores de elementos.

«Switch»

Este é um bloco de roteamento que permite selecionar por meio de uma porta qual será o sinal de saída, a partir de um conjunto de sinais de entrada.

Apêndice B

Nesta seção apresentamos uma visão geral das principais ferramentas que apoiam as abordagens SMarty e CVL.

B.1 Ferramentas de apoio à SyMPLES-SMarty

A seguir são exibidas ferramentas que apoiam a abordagem SMarty em conjunto com a linguagem SysML. São basicamente ferramentas que suportam a modelagem em linguagens UML e SysML.

UML Designer (Obeo, 2013) é uma ferramenta gráfica para editar e visualizar modelos UML. Ela é baseada no projeto Eclipse Sirius (Eclipse Foundation, 2013) no qual fornece um ambiente de trabalho para engenharia da arquitetura baseada em modelos. Projetos baseados no Eclipse Sirius se aproveitam das tecnologias do *Eclipse Modeling* (Eclipse Foundation, 2015), incluindo as tecnologias EMF (Eclipse Foundation, 2008) e GMF (Eclipse Foundation, 2005). Esta ferramenta também possui uma versão específica para a geração de modelos SysML, chamada de *SysML Designer*. A Figura - 2.1 exibe uma visão do *UML Designer* onde é possível visualizar a tela de aplicação de estereótipos ao elemento selecionado.

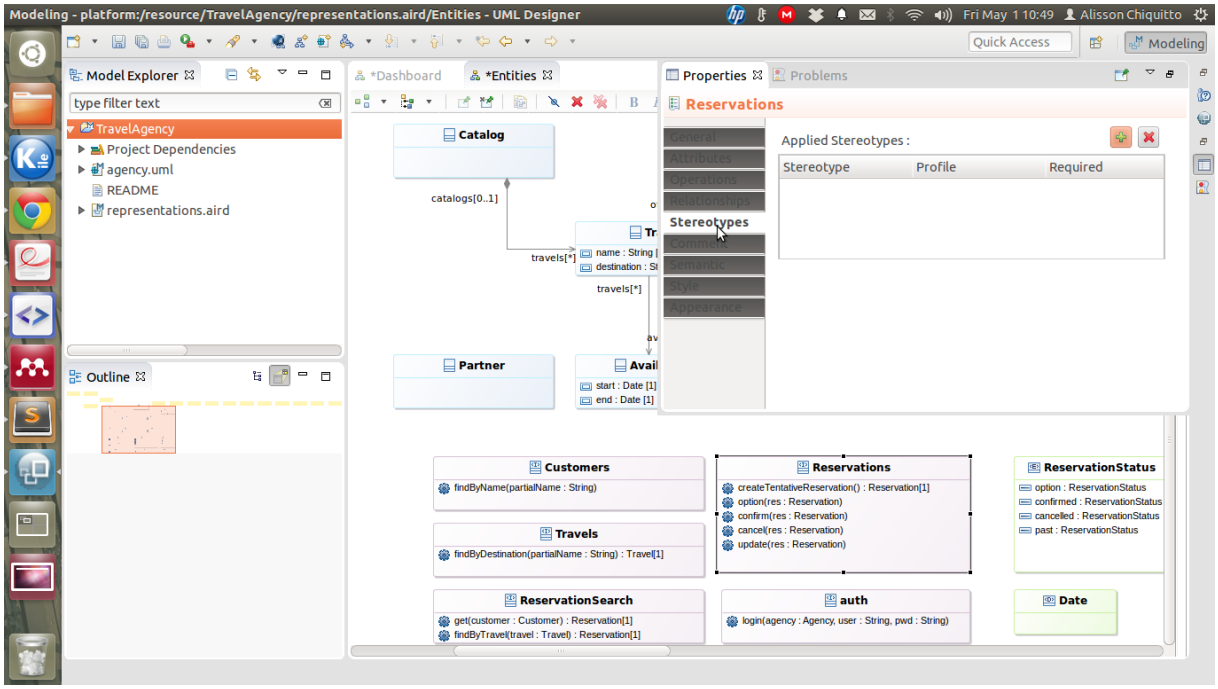

Figura 2.1: Tela da ferramenta UML Designer

A ferramenta *TopCased* (TopCased, 2013) é um software baseado na plataforma Eclipse e tem como objetivo fornecer um conjunto de ferramentas de desenvolvimento dedicado a softwares e hardwares para sistemas críticos. Basicamente é uma distribuição da plataforma Eclipse integrada com os editores UML/SysML do *plugin* Papyrus (Eclipse Foundation, 2014). *TopCased* foi descontinuado em 2012, e suas atividades foram migradas para o projeto *PolarSys*.

PolarSys é um grupo de trabalho (consórcio) da Fundação Eclipse que reúne grandes organizações para a colaboração na criação e suporte de ferramentas *Open Source* com foco no desenvolvimento de sistemas embarcados (PolarSys Group, 2005). Sua principal ferramenta é a *Polarsys IDE*, baseada na plataforma Eclipse e agregando componentes selecionados por membros do grupo *PolarSys* para a engenharia de sistemas baseada em modelos e desenvolvimento de sistemas embarcados. Na Figura - 2.2 é exibido uma tela do *PolarSys IDE*.

Por fim temos o Eclipse IDE que em conjunto com o *plugin* Papyrus torna a IDE em um editor de UML e linguagens DSL. Papyrus é um *plugin* para Eclipse baseado no Eclipse EMF, que permite a criação e edição de modelos UML e SysML. O Papyrus oferece um suporte ao perfil UML e permite que usuários definam DSLs baseadas em no perfil UML.

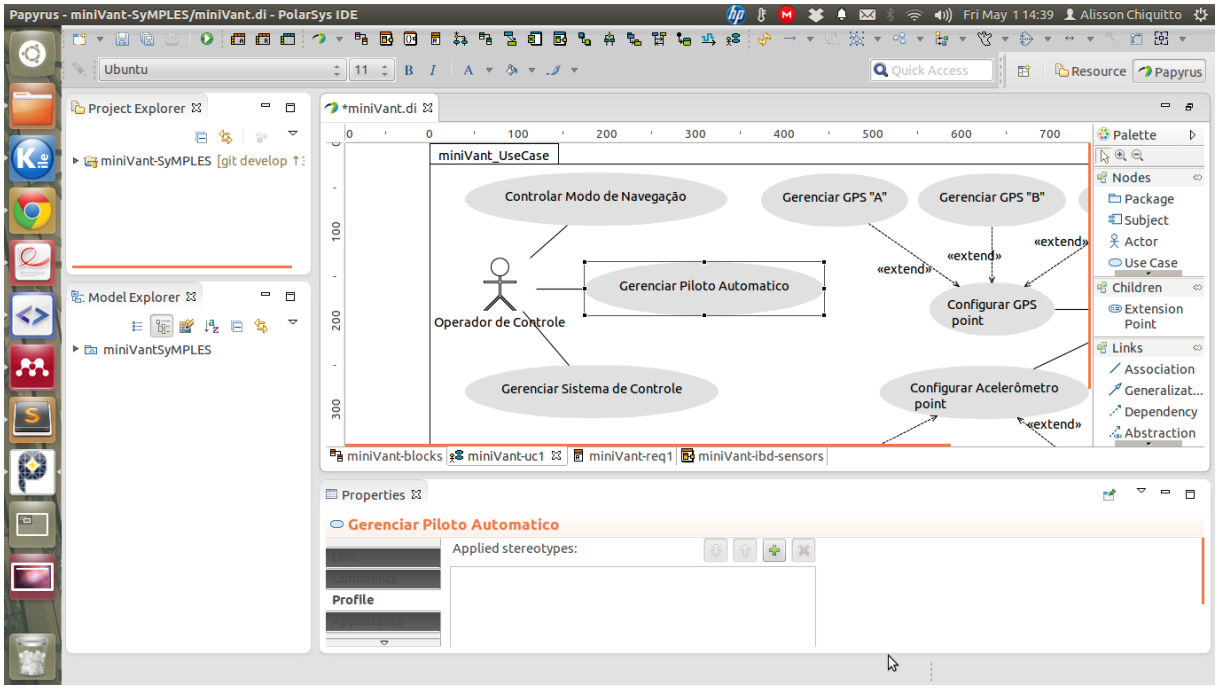

Figura 2.2: Tela do PolarSys IDE

Todas as ferramentas citadas são *Open Source* baseadas na plataforma Eclipse IDE e em suas tecnologias relacionadas. Este é o motivo pela qual todas são muito semelhantes visualmente.

Para a realização deste trabalho, todos os modelos SysML das LPS, incluindo os modelos gerados durante o processo da avaliação experimental, foram criados com a ajuda do Eclipse IDE versão Luna e o *plugin* Papyrus na versão *1.0*. A Figura - 2.3 exibe a utilização da plataforma Eclipse Luna integrado com o *plugin* Papyrus.

B.2 Ferramentas de apoio à CVL

A CVL por ter um caráter composicional, necessita de ferramentas que abordam tanto os Modelos Base CVL quanto os modelos da CVL. Isto está ilustrado na Figura - 2.4.

No contexto das abordagens SyMPLES, as ferramentas para a definição dos Modelos Base CVL são aquelas no qual é possível modelar diagramas UML/SysML. Algumas ferramentas dessa categoria foram citadas na Seção B.1.

A seguir são exibidas duas ferramentas que apoiam a criação de modelos CVL. Todas as ferramentas são específicas para a aplicação da linguagem CVL.

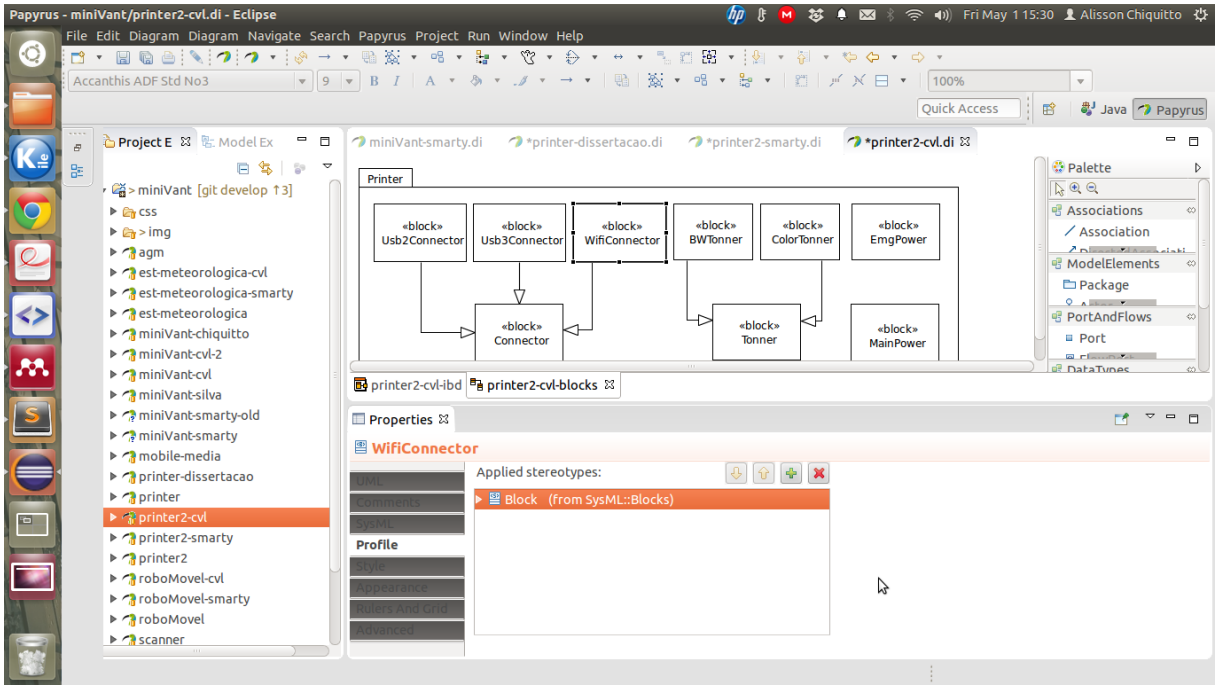

Figura 2.3: Tela da ferramenta Eclipse Luna com plugin Papyrus 1.0

A primeira ferramenta é o *plugin* para Eclipse (versão Helios SR2) chamado *CVL Tool from SINTEF* (OMG, 2014a)[1]. Por meio do seu uso é possível gerar Modelos de Variabilidade CVL, Modelos de Resolução CVL, mapear os elementos do Modelo de Variabilidade CVL com os do Modelo Base CVL e finalmente, executar a Materialização para a derivação de produtos. Essa ferramenta já foi utilizada em alguns estudos ((Svendsen et al., 2010), (Vasilevskiy, 2013) e (Nascimento et al., 2013)), mas atualmente não recebe mais atualizações, por este motivo o seu uso só é possível em versões muito específicas do Eclipse e do Papyrus. Um outro problema desta ferramenta é que em testes com a linguagem SysML, esta ferramenta não foi capaz de Materializar produtos.

A segunda ferramenta é *CVL 2 TOOL* (Group, 2014). Apesar de ainda estar em versão *alpha* ela oferece suporte para a geração da Árvore *VSpec* CVL e Modelos de Resolução CVL. Ela não oferece apoio ao mapeamento de elementos com Modelos Base CVL e consequentemente não executa a materialização para a geração de produtos.

Na Figura - 2.5 é exibido um Modelo de Variabilidade CVL gerado pela *CVL Tool from SINTEF*. E na Figura - 2.4 é exibido uma Árvore VSpec criada pela ferramenta *CVL 2 TOOL*.

[1]Um guia de instalação desta ferramenta encontra-se disponível em `https://gist.github.com/chiquitto/9195116`

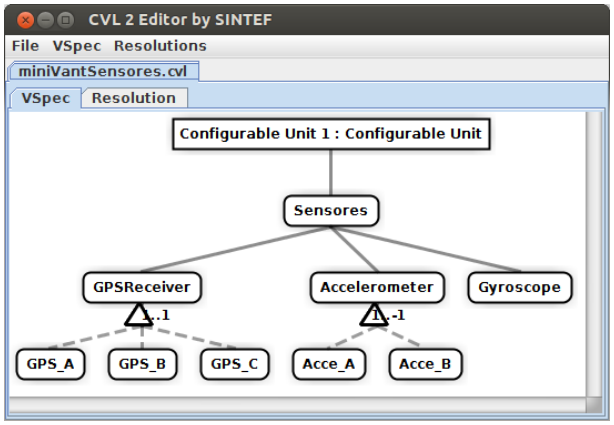

Figura 2.4: Tela da ferramenta CVL 2 TOOL

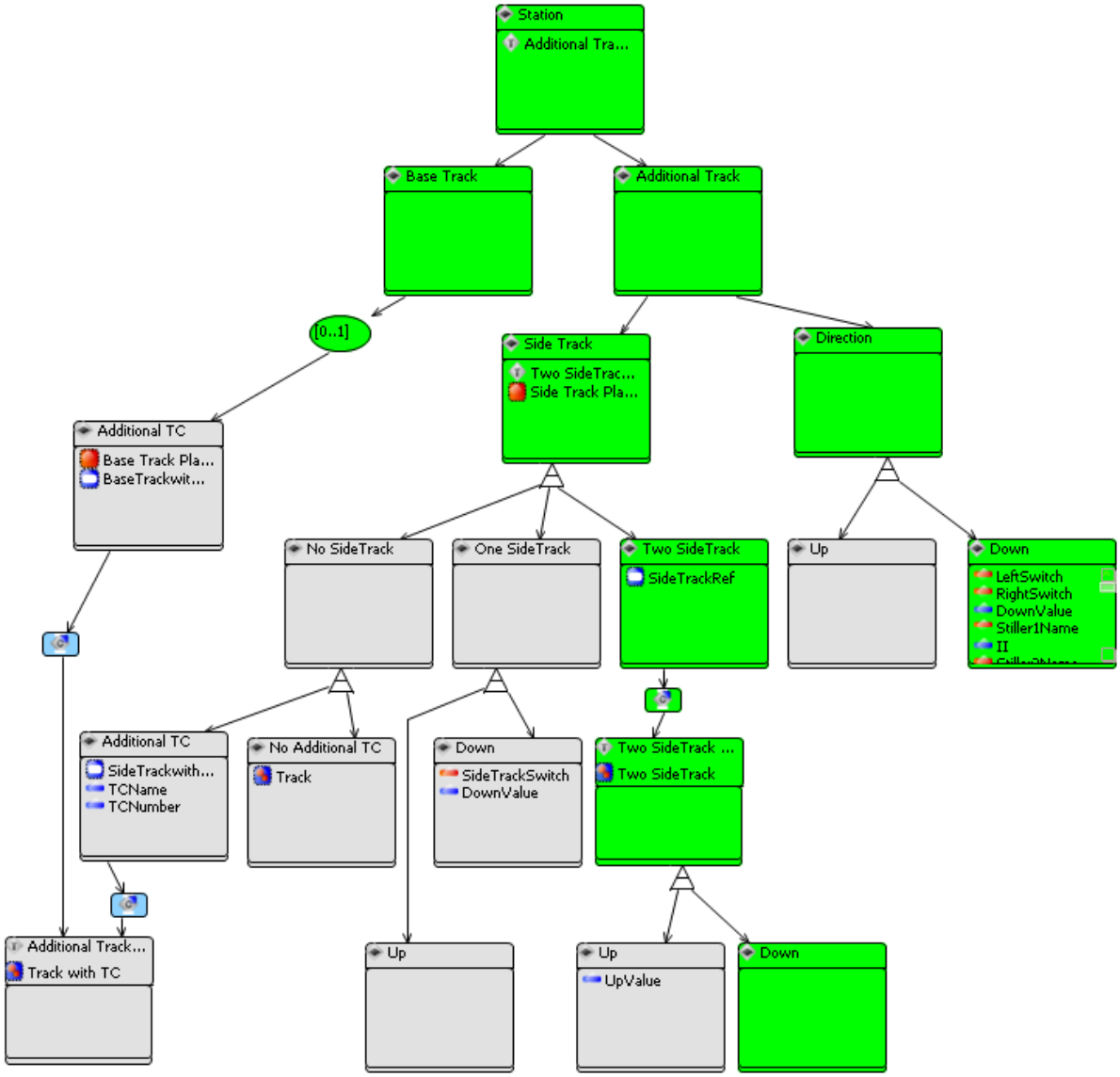

Figura 2.5: Modelo de Variabilidades CVL gerado pela ferramenta CVL Tool from SINTEF