Doc.
3.3

Abordagem SyMPLES v2.0

I. Visão Geral da Abordagem SyMPLES v2.0

A abordagem *SysML-based Product Line Approach for Embedded Systems (SyMPLES)* que foi proposta para o desenvolvimento de Linhas de Produto de Software (LP) baseadas na linguagem SysML para o domínio de Sistemas Embarcados (SE).

A abordagem *SyMPLES* contém um perfil, que é uma extensão da linguagem *SysML* criado por um mecanismo de profiling: o *SyMPLES Profile for Functional Blocks (SyMPLES-ProfileFB)*. O objetivo do *SyMPLES-ProfileFB* é fornecer uma semântica adicional aos blocos SysML.

II. A linguagem CVL

A abordagem *SyMPLES v2.0* utiliza a linguagem *CVL* para a especificação e resolução de variabilidades. A linguagem *CVL* pode ser aplicada a qualquer linguagem que seja baseada em MOF, como a *UML* e *SysML*.

A linguagem *CVL* expressa os conceitos de variabilidade através de um modelo chamado de Modelo de Variabilidades, que é geralmente representado em uma estrutura de árvore (*CVL Tree*), que por sua vez contém nós que são as especificações da variabilidade (*VSpec*) do Modelo Base.

III. Diretrizes para identificação e delimitação de variabilidade

Nesta seção é apresentado o processo *SyMPLES-ProcessVarCVL* que tem o objetivo de identificar e delimitar as variabilidades de uma LP baseada em *SysML* para a criação de modelos de variabilidade da *CVL*.

As atividades e diretrizes do *SyMPLES-ProcessVarCVL* são descritas nas subseções seguintes:

III.1 Diretrizes para identificação de variabilidades

Na definição da *CVL*, pontos de variação são especificações de variabilidade no Modelo Base e fazem parte do Modelo de Variabilidade. Eles definem modificações específicas que serão aplicadas ao Modelo Base durante a materialização.

Identificação dos Pontos de Variação CVL ObjectExistence

ObjectExistence é um ponto de variação *CVL* que indica que um objeto do Modelo Base pode ou não existir no modelo materializado .

D.1.1. 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 estendidos representam pontos de variação, enquanto os casos de uso que se estendem, representam variantes;

D.1.2. elementos de modelos de casos de uso relacionados com a associação de inclusão (*«include»*) ou associados a atores indicam variantes obrigatórias ou opcionais;

D.1.3. em modelos de requisitos, pontos de variação e suas variantes podem ser identificadas nos seguintes relacionamentos:

- **a.** contenção (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;
- **b.** derivação (relação deriveReqt), os requisitos derivados indicam variantes obrigatórias ou opcionais;
- **c.** dependência (relação dependency), os requisitos dependentes indicam variantes obrigatórias ou opcionais;

D.1.4. em modelos de blocos (diagrama de definição de blocos), pontos de variação e suas variantes 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.1.5. os elementos de modelos de blocos (diagramas de definição de blocos), relacionados às associações nas quais os seus atributos aggregationKind possuem valor none, ou seja, não representam nem agregação nem composição, indicam variantes obrigatórias ou opcionais;

D.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.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;

D.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;

Identificação dos Pontos de Variação CVL *SlotValueAssignment*

Um slotValueAssignment é um ponto de variação que especifica que um valor pode ser atribuído a um slot de um objeto no Modelo Base .

D.2.1. blocos marcados com o estereótipo *«constant»* possuem um parâmetro, no qual sugerem um slot para atribuição de valores;

D.2.2. atributos de tipos primitivos, como int, string ou boolean, e que possuem valores iniciais sugerem um slot para atribuição de valores;

Identificação de Restrições entre Pontos de Variação CVL

D.3.1. 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.3.2. variantes mutuamente exclusivas (X e Y) para um determinado produto devem ter seus relacionamentos de dependência marcados com a restrição "*X implies not Y*" na *CVL Tree*;

III.2 Árvore VSpec

Um *Vspec* é uma indicação da variabilidade no Modelo Base. Na definição da CVL existem 4 tipos de *VSpec*, nos quais 2 são tratados neste documento:

1. escolha (*choice*) – é um *VSpec* cuja resolução requer uma decisão Sim/Não (Verdadeiro/Falso);

2. variável (*variable*) – é um *VSpec* cuja resolução envolve o fornecimento de um valor de um tipo específico;

Definição dos VSpecs

D.4.1. cada ponto de variação *objectExistence* deve ser representado por um *VSpec* do tipo escolha;

D.4.2. cada ponto de variação *slotValueAssignment* deve ser representado por um *VSpec* do tipo variável;

Criação da árvore VSpec

Os *VSpecs* podem ser organizadas como árvores, onde a relação pai-filho organiza o espaço de resolução por imposição da estrutura e lógica nas resoluções permitidas. Uma árvore é uma estrutura de dados não-linear constituída por um nó raiz e nós adicionais que formam uma hierarquia de nós. Uma *VSpec Tree* 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 LP.

D.5.1. os *VSpecs* do tipo escolha que são obrigatórios deverão ser ligados ao seu elemento pai por meio de uma linha sólida;

D.5.2. os *VSpecs* do tipo escolha que são opcionais deverão ser ligados ao seu elemento pai por meio de uma linha tracejada;

D.5.3. os *VSpecs* do tipo variável deverão estar relacionadas a *VSpecs* do tipo *choice*, e deverão ser ligados ao seu elemento pai por meio de uma linha sólida;

III.3 Exemplo

A Figura 1 apresenta um exemplo de representação de variabilidade na LP "*Printer*" em um Diagrama de Definição de Blocos da *SysML*. A criação do Modelo de Variabilidades se da em dois passos consecutivos:

1. Identificação dos pontos de variação: A LP apresenta 3 interfaces de conexão (*Usb2Connector*, *Usb3Connector* e *WifiConnector*), 2 alternativas de escolha de *Tonner* e uma porta de Fornecimento de Energia Emergencial que é opcional. De acordo com a diretriz D.1.4 foram identificados os seguintes pontos de variação CVL *objectExistence*: os blocos *Usb2Connector*, *Usb3Connector*, *WifiConnector*, *BWTonner* e *ColorTonner*. Pela definição da LP, o bloco *EmgPower* é opcional, e por este motivo ele foi identificado como um ponto de variação *objectExistence*.
2. Criação da Árvore *VSpec*: Cada ponto de variação que foi identificado na tarefa anterior se tornará um nó (*VSpec*) na árvore. Nós adicionais foram criados para que fosse possível criar a estrutura de árvore e dar semântica para os nós originais. O nó *Connector* recebeu uma multiplicidade de grupo *1*..* indicando que no mínimo um descendente deve ser selecionado durante a materialização do produto. O nó *Tonner* recebeu uma multiplicidade de grupo *1..1*, que indica que apenas 1 de seus descendentes deve ser selecionado durante o processo de materialização. Uma linha tracejada ligando dois *VSpecs* indica que a resolução do nó filho é opcional, como é o caso do *EmergencyPower*. Ainda neste modelo existem 2 restrições: (a) "*ColorTonner implies Usb3Connector*" indicando que se o nó *ColorTonner* for resolvido positivamente, então o nó *Usb3Connector* deve obrigatoriamente ser resolvido positivamente, e (b) "*EmergencyPower implies not WifiConnector*" indicando que se *EmergencyPower* for resolvido positivamente, o inverso deve acontecer para o *WifiConnector*.

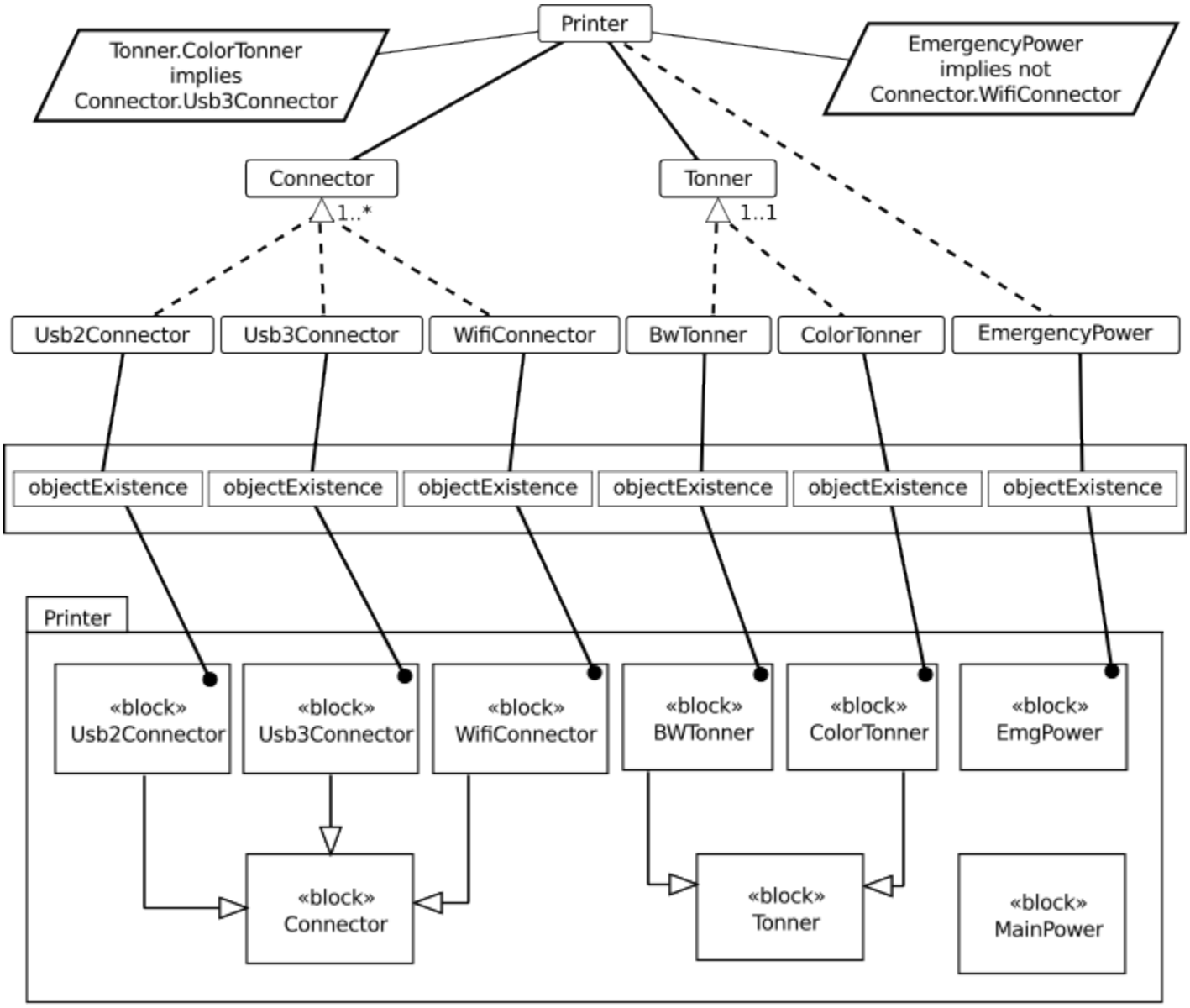

Figura 1 – LP “Printer” representada com o uso da abordagem SyMPLES v2.0. Na parte superior existe um Modelo de Variabilidades da linguagem CVL, e na parte inferior existe um Diagrama de Definição de Blocos do SysML.