Uma Linguagem de Modelagem de Domínio Específico para Linhas de Produto de Software Dinâmicas

Helberth Borelli
Universidade Federal de Mato Grosso – UFMT
hborello@gmail.com

Sérgio Teixeira de Carvalho
Universidade Federal de Goiás – UFG
sergio@inf.ufg.com

ABSTRACT

Software Product Lines can be briefly defined as a family of products that share some commonalities. The feature models approach is used to represent the commonalities and variabilities among products. One possible way for the construction of these models is the use of meta-models. However, the management of these features, traditionally performed at development time, is not sufficient for the development of Adaptive Systems. One solution to this type of development is the approach of Dynamic Software Product Lines which has as one of its aims to promote features input and output at runtime allowing the product to be adapted after changes in contexts. This article proposes a Domain Specific Modelling Language that aims at modeling Dynamic Software Product Lines. This language describes dynamic adaptations promoted at runtime and through the use of the state machine approach. Such adaptations, based on state machines, must promote state features changes or features exchanges at runtime, enabling these software products to manage derivations in its life cycle. Our proposal was implemented with the use of a Healthcare scenario and its development was based on the concept of the component service model.

KEYWORDS

Linguagem de Domínio Específico, Metamodelos, Linha de Produto de Software Dinâmica, Sistemas Adaptativos

1 INTRODUCÃO

Sistemas Sensíveis ao Contexto [22] têm como desafio apresentar soluções para ambientes nos quais se apresentam a Computação Ubíqua [24], por exemplo, tais sistemas devem se adaptar mediante a entrada e saída de recursos em tempo de execução. Esses recursos (dispositivos eletrônicos como: *Smartphones*, TVs, Sensores etc) normalmente atuam de forma distribuída e independente. Em outras palavras, estes recursos podem estar ou não disponíveis, bem como ser ou não utilizados de acordo com o contexto percebido pelo sistema, o qual deve ser capaz de mudar sua configuração em tempo de execução.

Uma abordagem que pode ser utilizada para resolução de problemas relacionados a sistemas sensíveis ao contexto é a utilização do conceito de Sistemas Adaptativos. Esses sistemas têm como princípio a capacidade de se adaptarem mediante as mudanças de contexto [8]. Sistemas adaptativos podem ser aplicados em várias áreas do conhecimento e este estudo aborda primariamente um cenário de Sistemas de *Healthcare*, o que não exclui a aplicação do mesmo trabalho para outras áreas, tais como educação, agricultura, segurança, controles empresariais e demais áreas que por ventura necessitem do conceito de adaptabilidade em tempo de execução.

Contextualizando tais abordagens em um cenário que possui requisitos de adaptação, um Sistema de *Healthcare* deve, por exemplo, auxiliar um paciente a executar tarefas que estejam prescritas em seu tratamento, sendo que tais tarefas são realizadas em um ambiente residencial. Diante disso, um dos requisitos do sistema é que o paciente deve ser informado para que realize alguma tarefa referente a seu tratamento. Para tal, o sistema deve então avisá-lo por meio de dispositivos eletrônicos, tais como *Smartphone*, *TV* ou *Tablet*, os quais poderão exigir do sistema alguns requisitos de adaptabilidade. Assim, caso algum dispositivo esteja indisponível, outro dispositivo deverá ser selecionado, em tempo de execução, pela aplicação para que o paciente receba sua tarefa.

Este trabalho utiliza de abordagens como Linha de Produto de Software Dinâmica (do inglês, *Dynamic Software Product Line* - DSPL) [11], Análise de Domínio Orientada a Características [14, 15], Meta-modelagem MOF (do inglês, *Meta Object Facility*) [21] e Linguagem de Modelagem de Domínio Específico (do inglês, *Domain Specific Modelling Language* - DSML) [5]. Dado o contexto com necessidades de adaptação em tempo de execução, a utilização de técnicas de desenvolvimento tradicional não satisfaz os requisitos exigidos para o desenvolvimento destes Sistemas Adaptativos. Uma maneira de resolver tais problemas é a adaptação de conceitos e abordagens consolidadas à nova realidade de desenvolvimento. Assim, este estudo tem como objetivo produzir metamodelos, representados por uma DSML [25], que possibilitem a modelagem de Sistemas Adaptativos que permitam adequações em tempo de execução.

Este artigo está organizado da seguinte forma: Seção 2 apresenta os conceitos abordados no desenvolvimento da DSML, dividida em subseções que apresentam as abordagens dos Sistemas Adaptativos, DSPL e *Feature Models*, Meta-modelagem e Modelagem. A Seção 3 descreve o desenvolvimento do Metamodelo de Características, de Variabilidades Dinâmicas e Aplicação ou Derivação de Produtos de Software. A Seção 4 aborda o desenvolvimento da DSML por meio da transição do metamodelo para uma gramática de linguagem, o desenvolvimento de restrições de modelagem e o gerador de esqueleto de código que deverão compor a DSML. A seção 5 apresenta o uso passo a passo da DSML desenvolvida no projeto. A Seção 6 apresenta um estudo de caso descrevendo o desenvolvimento de um produto com variabilidade dinâmica que promove a validação da DSML. A Seção 7 relaciona artigos com abordagens semelhantes. Finalmente, a Seção 8 apresenta as considerações finais do artigo.

2 BASE TEÓRICA

Esta seção apresenta as abordagens utilizadas para o desenvolvimento do estudo, o qual tem como objetivo alcançar uma DSML para modelagem de produtos por meio do uso de conceitos de DSPL e Sistemas Adaptativos.

2.1 Sistemas Adaptativos

Para a computação, o conceito de sistemas adaptativos pode ser definido, por exemplo, como o estudo que relaciona os recursos de um sistema e seu ambiente de execução. Durante décadas, esse conceito foi discutido e sua definição é constantemente questionada [19].

Uma das abordagens usadas para desenvolver sistemas adaptativos em tempo de execução é a abordagem da máquina de estado. Em poucas palavras, as máquinas de estados são baseadas em um conjunto de regras que definem a transição de estados. Em geral, essas regras são constituídas por estados, eventos, condições e ações [18]. Por exemplo, um sistema adaptativo pode, através do uso de uma máquina de estado, reagir a um evento que realiza a transição de adaptação de seus recursos. Essas transições ou adaptações podem ter como um de seus objetivos evitar falhas na execução do software.

2.2 Linha de Produto de Software Dinâmica

Uma DSPL é uma abordagem derivada da Linha de Produto de Software (do inglês, *Software Product Line* - SPL) desenvolvida para gerenciar seus recursos como variabilidades estáticas. Em outras palavras, a diferença principal das abordagens, em termos de resultado, está no momento da derivação do produto, sendo que em SPL a derivação ocorre em tempo de desenvolvimento, já em DSPL a derivação pode ocorrer em tempo de execução.

Como visto anteriormente, Sistemas Adaptativos devem ser capazes de se ajustarem às mudanças de contexto, trazendo assim necessidades de adaptação em tempo de execução, ou seja, devem lidar com variabilidades dinâmicas. Esses novos requisitos geram o termo DSPL. Essa abordagem une conceitos como SPL e Sistemas Adaptativos [7]. Sistemas Adaptativos podem invocar a tomada de decisão por meio da observação do ambiente de execução (alterações de requisitos, falhas etc.), esta seleção de mecanismos adaptativos tem o objetivo de manter o sistema funcionando.

A Figura 1 apresenta uma visão dos conceitos de SPL e DSPL mostrando a diferença entre eles. Na ilustração pode ser observado que na DSPL as variabilidades ocorrem mesmo após a implantação dos produtos de derivação.

Pesquisas envolvendo DSPL trazem algumas abordagens ou técnicas para lidar com variabilidades em tempo de execução [3]. Entre as soluções encontradas, destacamos as abordagens Orientadas à Características organizada em uma arquitetura que permita a troca de características em tempo de execução. Nesse conceito, um recurso, representado por uma característica, pode representar pontos comuns e variabilidades de um sistema [14]. Os modelos de características representam em um alto nível de abstração, as partes significativas e suas associações em um sistema [16].

Em relação ao tratamento de variabilidades, uma possível abordagem é o conceito de Contratos (*Design by Contract*) [17]. Nesse contexto, o objetivo é definir regras para descrever as variabilidades da DSPL, abordando um conjunto de diretrizes que determinam restrições de dependência e regras de contexto.

2.3 Metamodelagem

Metamodelos, também conhecidos como modelos de modelos, os metamodelos têm como objetivo analisar, construir e desenvolver

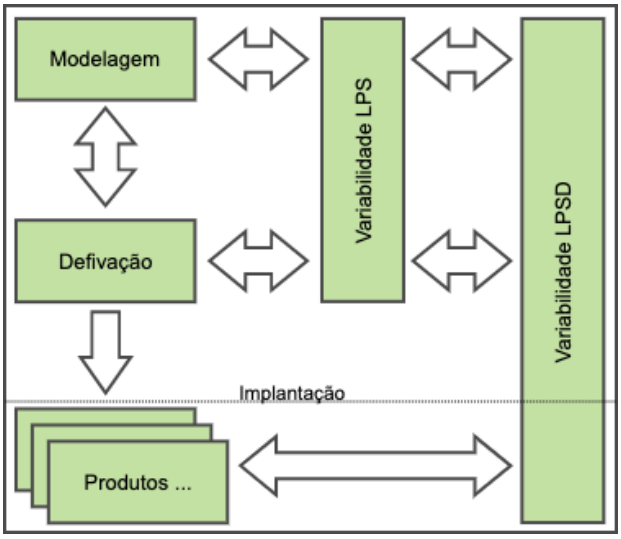

Figure 1: Modelagem e Derivação de Produtos em SPL e DSPL

modelos que definem regras, restrições e aplicação de teorias que determinam a construção de sistemas para um determinado domínio. Os metamodelos não devem ser considerados apenas como parte da documentação do software, mas também como parte da implementação do software [23].

O uso da metamodelagem com a utilização proposta pela arquitetura do MOF possui divisão em níveis. O nível M3 é representado neste trabalho pelo meta-metamodelo *Ecore* juntamente com o *framework* xText [27] para o desenvolvimento da DSML. O nível M2 representa os metamodelos e gramática que serão desenvolvidos para a modelagem dos produtos de software. O nível M1 representa os modelos criados partir da DSML desenvolvida. O nível M0 representa a instanciação dos modelos que deverão resultar em objetos de aplicação [21].

O principal objetivo no desenvolvimento de metamodelos é estabelecer que os esforços sejam mantidos em alto nível, abstraindo as complexidades causadas pela implementação em diferentes plataformas [7].

2.4 Linguagem de Modelagem de Domínio Específico

O termo Linguagem de Domínio Específico (do inglês, *Domain Specific Language* - DSL) é definição de uma linguagem de programação geralmente declarativa, de elementos reduzidos para determinar um problema específico de domínio. As principais motivações para o uso de uma DSL são a proximidade da linguagem com o domínio a ser especificado e o ganho de produtividade ao especificar em uma linguagem mais familiar aos seres humanos.

Podemos considerar o desenvolvimento de um metamodelo como ponto de partida para a construção da DSL, uma vez que o metamodelo pode ser considerado uma linguagem específica para a implementação de modelos [13]. Nesse sentido, a DSL é definida como um conjunto coordenado de modelos, que permite uma redefinição do termo DSL para o termo DSML.

A construção de uma DSML a partir de um metamodelo não determina que uma linguagem seja definida apenas pelas restrições impostas pelo metamodelo. Por exemplo, dadas algumas limitações de representação de regras e restrições em metamodelos representados por linguagens de modelagem (por exemplo, UML), um conceito chamado OCL (*Object Constraint Language*) foi desenvolvido.

A OCL é uma linguagem declarativa e formal proposta para descrever expressões aplicadas a UML. Essas expressões são usadas para especificar condições a serem fornecidas para descrever objetos em uma modelagem. Como os metamodelos, as linguagens de modelagem específicas do domínio também fornecem recursos para programação de restrições por meio das OCLs. Em um caso específico, a estrutura *Xtext*, usada para criar linguagens específicas, também possui métodos de especificação de condições ou restrições com uso similar ao conceito de OCLs.

3 O DESENVOLVIMENTO DO METAMODELO

O desenvolvimento do metamodelo foi dividido em três partes interrelacionadas: o Metamodelo de Características, o Metamodelo de Variabilidades e o Metamodelo de Aplicação (Derivação).

3.1 Metamodelo de Características

O Metamodelo de Características permite a modelagem de uma SPL com a abordagem de modelos de características [14], o que representa um mapeamento dos recursos reutilizáveis da SPL. Usando o metamodelo, é possível estabelecer um relacionamento entre as características, atribuindo regras e restrições ao desenvolvimento de uma família de produtos. A Figura 2 apresenta o Metamodelo de Características.

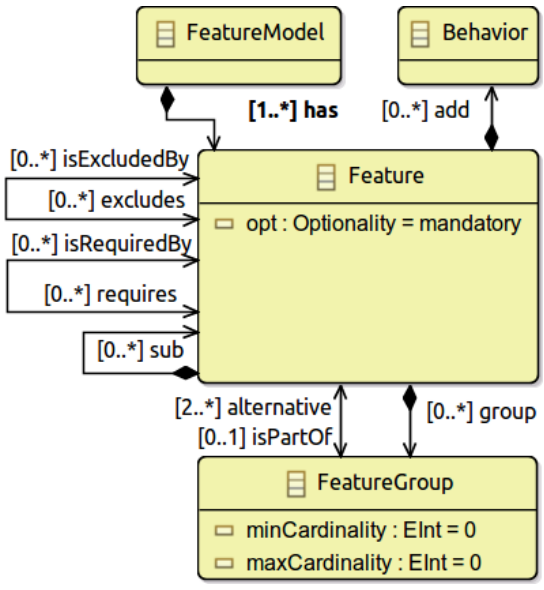

Figure 2: Metamodelo de Características.

Cada modelo de característica é iniciado por uma *FeatureModel*, classe considerada a raiz do modelo desenvolvido para ser uma família de produtos de software. As características são modeladas pela classe *Feature*, sendo que cada característica modelada deve receber suas atribuições durante a modelagem, as quais são determinados por atributos e associações de metamodelos. Uma característica pode ser definida como obrigatória ou opcional por meio do atributo *opt*, o atributo definido como obrigatório determina o uso das características em todas as derivações de produtos. Outro papel importante desempenhado pelos recursos é a autorreferência *sub*, que modela *SubFeatures*, estabelecendo uma hierarquia ou dependência entre os recursos modelados.

As *SubFeatures* modeladas podem desempenhar o papel de características alternativas; para isso, elas precisam ser agrupadas por uma *FeatureGroup*. Inicialmente, uma *FeatureGroup* é composta por uma raiz *Feature* e agrupada por uma *SubFeatures*. Por exemplo, é possível definir em um modelo de característica uma raiz *Feature* chamada Comunicação e *SubFeatures* denominadas *ADSL* e *GSM*, que podem ser agrupadas por uma *FeatureGroup*. Outro atributo de uma *FeatureGroup* é a possibilidade de definir cardinalidades, representadas pelos atributos *minCardinality* e *maxCardinality*. Esses atributos visam determinar a quantidade mínima e máxima de recursos que podem ser selecionados em uma derivação de produto.

O metamodelo de características implementa as autorreferências *isRequiredBy*, *requer*, *excludes* e *isExcludedBy* como restrições de dependência da classe *Feature*. Esses relacionamentos seguem o conceito de que as características podem ser dependentes ou estabelecer dependência uma da outra. Por exemplo, para iniciar a execução de uma determinada características de um produto de software, outra característica pode ser necessária ou excluída para permitir que o produto derivado funcione normalmente.

Complementando o metamodelo de características, a classe denominada *Behavior* visa incluir comportamentos nas características modeladas. Esses comportamentos são baseados na especificação de interfaces de objetos ou artefatos do repositório de características que deve ser gerenciado pela SPL. Por exemplo, ao modelar uma característica ADSL, o método usado para iniciar a conexão pode ser especificado no momento da modelagem.

3.2 Metamodelo de Variabilidade Dinâmica

O Metamodelo de Variabilidades Dinâmicas visa estabelecer regras que determinam a variabilidade dinâmica das características da SPL em tempo de execução, ou melhor DSPL. O desenvolvimento do metamodelo de variabilidades dinâmicas baseia-se no uso do conceito de Contratos desenvolvido por Meyer em *Applying 'design by contract'* [17], que deverá estabelecer regras de negociação para determinar como as variabilidades devem ser modeladas.

A principal forma de variabilidade dinâmica desenvolvida neste estudo é o conceito de Máquinas de Estado, que apresenta duas formas de implementação: mudanças de estado por ação do usuário ou por contexto. As alterações de estado promovidas pela ação do usuário consideram que o usuário pode solicitar a inclusão ou exclusão de algumas características em tempo de execução. As mudanças de estado promovidas pelo contexto estabelecem que, no caso de indisponibilidade de uma característica (*Feature*) definida como padrão, uma característica alternativa começa a funcionar em tempo de execução, possibilitando que o produto de software continue executando. A Figura 3 apresenta o metamodelo de variabilidades.

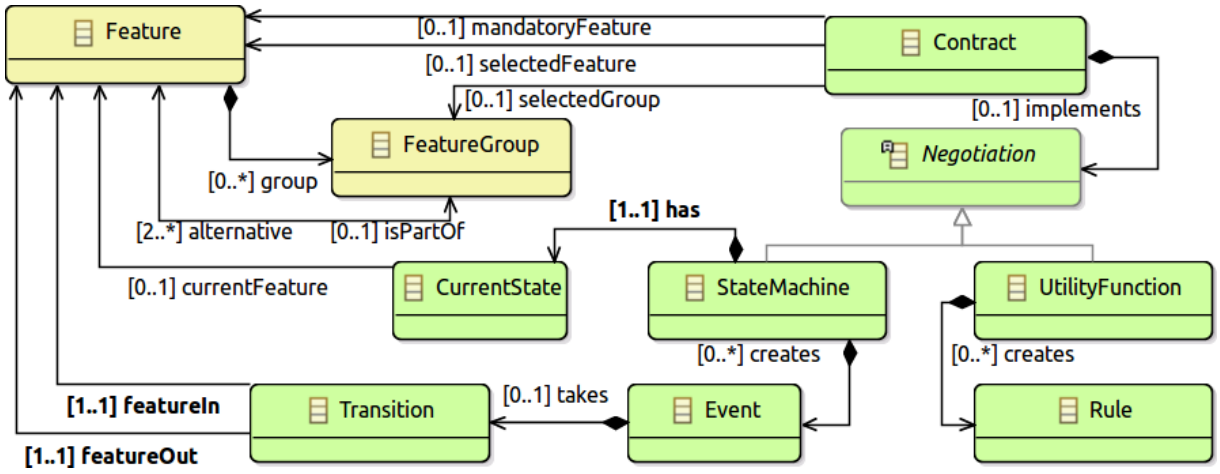

Figure 3: Metamodelo de Variabilidade Dinâmica.

A modelagem de variabilidades tem como foco a modelagem de características, ou seja, cada *Contract* está essencialmente associado a um determinado *Feature* ou a um conjunto de características representadas pela classe *FeatureGroup*. Um Contrato tem como atributos os estados estático ou dinâmico (definidos por um atributo denominado *binding-time*). Geralmente, as características mandatórias são associadas ao contrato estático, uma vez que essas características estão presentes em todos os produtos derivados. O contrato estático também pode ser associado a características opcionais e isso determina que essas características não serão variáveis em tempo de execução. Finalmente, o contrato dinâmico trabalha com características opcionais e grupos de características (*FeatureGroup*) a fim de estabelecer as negociações para as variabilidades dinâmicas.

Definido um Contrato Dinâmico, o próximo passo na modelagem é a definição do tipo de negociação. O metamodelo define dois tipos de negociação: *StateMachine*, que define alterações de estado promovidas por ação de usuário ou mudança de contexto, e o tipo *UtilityFunction*, o qual deve definir alterações baseadas em regras que determinam critérios de qualidade, determinando a característica mais apropriada. No entanto, este artigo não pretende aprofundar nos conceitos e implementações de funções de utilidade, deixando a modelagem desse tipo de variabilidade em um alto nível de abstração.

O metamodelo de variabilidades determina que um Contrato pode estabelecer três tipos de associações com as características: *requiredFeature*, que desempenha o papel exclusivo de associar uma característica mandatória (*mandatoryFeatures*); *selectedFeature*, que associa características opcionais e *selectedGroup*, que desempenha o papel de associar a um conjunto de características por meio de uma *FeatureGroup*.

Com base nas associações e na definição de um Contrato como dinâmico, vem a definição do tipo de negociação. Quando associamos um Contrato a uma característica opcional, espera-se a modelagem de uma máquina de estado por ação do usuário. Portanto, devemos determinar, a seguir, que tal caraterística tem o status "*on*" ou "*off*" por meio da classe *CurrentState*. Dessa forma, essa variabilidade dinâmica, inicialmente opcional, pode ser ativada pelo usuário em tempo de execução.

Quando associamos uma *FeatureGroup* a um Contrato dinâmico, duas negociações possíveis podem ser modeladas. O primeiro caso é a modelagem de uma máquina de estado por contexto, a qual trabalha com a transição de características de acordo com a disponibilidade dessas características. Portanto, uma Características é definida como *CurrentState on* e, em seguida, eventos (*Event's*) são modeladas para determinar as transições de uma característica para outra, observando que os recursos de transição pertencem ao mesmo grupo, *FeatureGroup*. O segundo caso trata da modelagem de uma *UtilityFunction* que permite a descrição, em alto nível, das regras de transição para seu uso.

Um exemplo que pode ser contemplado pelo uso de máquinas de estado, por contexto, é a implementação de uma transição entre os recursos *ADSL* e *GSM*. Supondo que o serviço *ADSL* seja preferencialmente estabelecido como um estado inicial de um Contrato de Comunicação, e por algum motivo esse recurso se torne indisponível, o sistema deve executar a negociação que estabelecerá a transição do *ADSL* para o *GSM*.

3.3 Metamodelo de Derivação

O Metamodelo de Derivação, também chamado aqui de Metamodelo de Aplicação, tem o objetivo derivar produtos de software por intermédio da seleção de Contratos (classe *Contract*). A derivação do produto é definida neste contexto como um conjunto de Contratos estáticos e dinâmicos que visam formar um produto de software.

A Figura 4 apresenta o Metamodelo de Derivação, no qual cada produto será modelado a partir da classe *ProductDerivation* que possui três tipos de associações com a classe *Contract*. As associações são chamadas *requiredContract*, *selectedFeature* e *dynamicContract*, que são baseadas na opcionalidade da classe *Feature*, podendo ser mandatórias, opcionais e dinâmicas.

A derivação do produto é iniciada depois que os modelos de características e variabilidades estiverem prontos. Essencial para cada produto, todos os contratos mandatórios devem ser selecionados em cada derivação, o que acontece por intermédio da associação

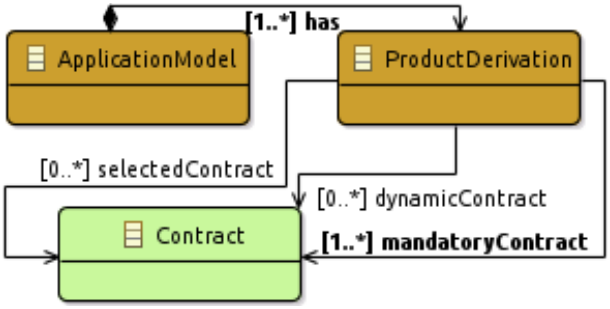

Figure 4: Metamodelo de Derivação ou Aplicação.

requiredContract. A segunda etapa da derivação é a seleção de contratos opcionais, que são divididos em dois tipos, contratos estáticos e dinâmicos. Contrato estático representa a variação que será definida apenas no momento da modelagem, definida pela associação *selectedContract*. Contrato dinâmico representa a variação que será responsável por eventos de adaptação em tempo de execução, que é definida pela associação *dynamicContract*.

Cada modelo definido pelo metamodelo de derivação determina um produto da DSPL. Pode-se definir que o metamodelo de características e o metamodelo de variabilidade dinâmica devem direcionar a modelagem de características e variabilidades e, portanto, apresentar o modelo de variabilidades representado por contratos, que deve modelar a família de produtos por derivação.

4 DESENVOLVIMENTO DA DSML

O primeiro passo para o desenvolvimento de DSML é uma transformação do metamodelo em uma gramática correspondente. O trabalho de transformação é feito pelo *framework* Xtext [28], que foi desenvolvido para trabalhar integrado ao ambiente de desenvolvimento Eclipse. Após produzir uma gramática, o processo continua com o desenvolvimento das restrições de modelagem e do gerador de código de esqueleto.

4.1 Desenvolvimento do Modelo de Restrições

O desenvolvimento da DSML permite aumentar o número de restrições de modelagem ao metamodelo, este agora representado por uma gramática. Nesse sentido, a ferramenta *Xtext* oferece a linguagem acessória *Xtend* [26] que permite implementações e personalizações para o projeto. Semelhante ao Java, a linguagem *Xtend* possibilita o desenvolvimento de métodos para aumentar as restrições, e esse desenvolvimento permite o uso de *APIs Java* para desenvolver métodos.

O desenvolvimento das restrições ajudou a definir regras, tais como: definir que o modelo deve ter ao menos uma Característica mandatória; definir que toda Característica mandatória deve estar vinculada a um Contrato mandatório; determinar que há apenas negociação para contratos dinâmicos etc. Nesse projeto foram implementadas dezoito restrições no total, as quais foram inicialmente consideradas as mais relevantes para manter a corretude dos produtos modelados.

4.2 Desenvolvimento do Gerador de Esqueletos de Códigos

O gerador de códigos, chamado de *Skeleton Code Generator* , tem como resultado a geração de códigos executáveis que são provenientes das modelagens, ou seja, a cada modelagem ou atualização de modelagem promovida o gerador produz classes, métodos e atributos referentes ao modelo desenvolvido. Este conjunto de códigos trata-se de um molde ou esqueleto de códigos que deve direcionar o desenvolvimento manual de acordo com regras de modelagem preestabelecidas.

Os métodos que implementam o gerador de códigos, são executados por eventos (*Event's*) gerados no momento da modelagem DSML. Cada evento é iniciado quando um objeto da gramática (característica, contrato etc.) é descrito pela DSML. Para isso, o *framework* Xtext oferece o desenvolvimento de métodos, os quais podem ser instanciados por métodos denominados *doGenerate* que devem resultar na geração de códigos relacionados aos objetos descritos na gramática.

Na implementação deste projeto, os objetos oriundos das classes de modelagem que resultam em código Java, são: *Feature*, *Contract*, *Enumeration*, *StateMachine*, *CurrentState*, *Event*, *Transition* e *Rule*. As classes *Feature* e *Contract* possuem, ao contrário de outras, uma implementação de códigos específicos (classes) para cada objeto modelado. Outras classes são geradas apenas uma vez para serem instanciadas pelas classes resultantes da modelagem de características e contratos.

Finalmente, a classe *ProductDerivation* tem com resultado a geração do código de construção de um *script* de definição do aplicativo. Após concluir a modelagem para derivação do produto, é gerado um arquivo *XML* que definirá um *POM* (do inglês, *Project Object Model*) para o *Apache Maven Project* [20], o qual visa gerenciar o projeto de construção.

Os arquivos gerados pela derivação definem os parâmetros para configurar o projeto do produto de software. Esses projetos são compilados como componentes dinâmicos de sistema e são denominados *bundles*. Após a compilação do projeto, os pacotes configuráveis podem ser armazenados em um repositório baseado nas especificações *OSGi* [1, 10]. Tais pacotes configuráveis podem ser iniciados e interrompidos, permitindo os testes de validação das variabilidades dinâmicas.

5 MODELANDO COM A DSML

O desenvolvimento da DSML visa diminuir as dificuldades inerentes à implementação dos Sistemas Adaptativos. A linguagem desenvolvida oferece, por exemplo, a possibilidade de modelar variabilidades dinâmicas para obter um esqueleto de código, visando reduzir o esforço do processo de codificação manual.

A Figura 5 mostra uma visão das etapas de modelagem de produtos utilizando a DSML desenvolvida. O uso da DSML foi dividido em seis etapas: a Etapa 1 representa o espaço de problema, o qual foi desenvolvido para modelar características estáticas e dinâmicas de uma DSPL; A Etapa 2 ilustra a geração de classes que representam características e contratos; A Etapa 3 representa o espaço de solução usado pela modelagem de produtos de software (derivações); A Etapa 4 ilustra a geração de códigos para o projeto a partir da derivação; A Etapa 5 mostra o desenvolvimento manual

para ajustes de projetos e componentes de arquitetura; A Etapa 6 representa o ciclo de vida dos projetos na plataforma *OSGi*.

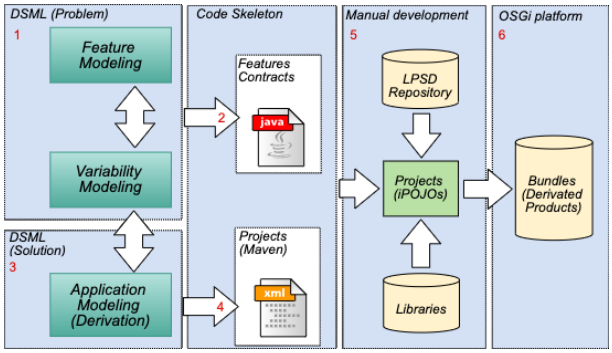

Figure 5: Visão do desenvolvimento de um DSPL modelado pela DSML.

O resultado final da DSML é representado por um esqueleto de códigos provenientes do gerador de código. Os códigos resultantes são compostos por classes representando características e contrato na linguagem Java, bem como, arquivos em *XML* que definem os projetos resultantes da derivação. O código gerado deve ser complementado com o desenvolvimento manual, complemento este necessário para conectar componentes de arquitetura às classes de características e contratos. Outra demanda de desenvolvimento manual diz respeito à adequação dos componentes da arquitetura aos requisitos funcionais da plataforma *OSGi*, em outras palavras, transformar componentes em pacotes configuráveis para serem instalados, iniciados e parados durante a execução do software. O modelo desenvolvido para executar os projetos gerados pela DSML seguem o conceito de implementação do modelo de serviço de componente iPOJO [2].

6 VALIDAÇÃO DA DSML

A proposta desta seção é demonstrar o uso da DSML no desenvolvimento de uma família de produtos a partir de um cenário. Para isso, parte de um modelo de características que compõe um Sistemas de *Healthcare* foi recortado para demonstrar o uso da DSML neste artigo. Apesar de ser um pequeno recorte do modelo, será possível aqui fazer uma validação das principais propriedades da DSML. O objetivo principal aqui é demonstrar o metamodelo de variabilidades dinâmicas (Contratos), desenvolvido e instanciado em um aplicativo de teste que promove a execução desses contratos.

O modelo apresentado visa demonstrar um sistema preparado para exibir notificações com o intuito de avisar o paciente sobre seu planejamento de saúde. Como exemplo prático, o paciente idealizado possui um plano de cuidados planejado com agendamentos para ministrar remédios e aferições de pressão arterial. Nesse contexto, o produto de software deve notificar o paciente, no momento exato, mostrando a tarefa ou prescrição em um recurso (característica) disponível (variável).

O cenário apresentado na Figura 6 foi modelado como um Contrato Dinâmico implementando com a abordagem de máquina de estados. Este Contrato é composto por uma lista de eventos e transições entre as características, *TV, Smartphone* e *Tablet*. Após a

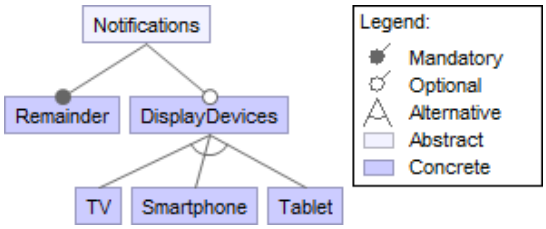

Figure 6: Modelo de Características do Sistema de Notificações.

derivação e implementação do produto de software, o produto foi testado usando o ciclo de vida *OSGi*. A Figura 7 apresenta as etapas para o desenvolvimento do Sistema de Notificações, atividades essa que são comuns ao desenvolvimento de qualquer Produto de Software Adaptativo.

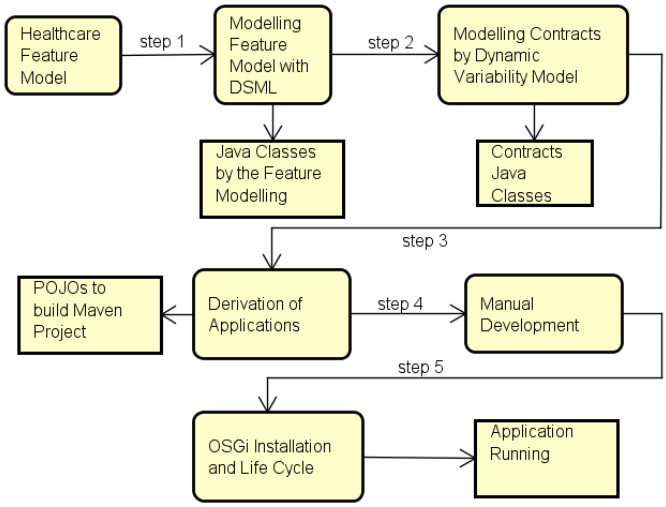

Figure 7: Passos para modelar e executar a aplicação *(Healthcare)*.

A DSML requer uma modelagem sequencial, ou seja, uma sequência que começa em modelos de características, variabilidades e derivação. O Código 6 apresenta parte de uma modelagem DSPL, começando com a modelagem de características (linhas 1 à 9) que descreve o modelo mostrado da Figura 6. No próximo passo, o contrato dinâmico (linhas 12 à 25) é modelado, sendo responsável por orquestrar as transições entre características (linhas 16 à 23) e, finalmente, o código mostra as derivações compostas pelos contratos selecionados (linhas 28 à 32).

```
1  // ***** Modelo de Caracteristicas *****
2  Feature 'DisplayDevice' {opt optional
3   sub {Feature 'TV'{opt optional},
4        Feature 'SmartPhone'{opt optional},
5        Feature 'Tablet'{opt optional}}
6    group {
7           FeatureGroup 'DisplayDeviceGroup' {
8          minCardinality 1 maxCardinality 3
```

```
9           alternative (TV, SmartPhone, Tablet)
10    }}
11  }
12  // *** Modelo de Variabilidades Dinamicas *****
13  Contract DisplayDeviceContract {
14      bindingTime dynamic
15      implements StateMachine {
16        has CurrentState {status on currentFeature
              TV}
17        creates {
18          Event 'noTV' {
19            takes Transition {featureIn SmartPhone
                  featureOut TV}
20          },
21          Event 'noSmartPhone' {
22            takes Transition {featureIn Tablet
                  featureOut SmartPhone}
23          },
24          Event 'noTable' {noService}
25      }}
26  }
27  // ******* Modelo de Derivacao ********
28  ProductDerivation 'Healthcare' {
29    mandatoryContract (ReminderContract,
          CarePlanContract, ...)
30    selectedContract (CommunicationContract, ...)
31    dynamicContract (DisplayDeviceContract, ...)
32  }
```

Código 1: Modelo de Características e Modelo de Variabilidades (Sistema de Notificações)

Após a modelagem de características, variabilidades e derivação, um esqueleto de código em Java é produzido automaticamente pela DSML. Todos os artefatos desenvolvidos nesses testes foram implementados sob a plataforma *OSGi*, usando a estrutura *Apache Felix*, no qual os componentes de software foram instalados, iniciados e parados. Documentação com mais detalhes sobre o desenvolvimento e utilização da DSML podem ser encontrados em Borelli (2016) [6].

7 TRABALHOS RELACIONADOS

Fernandes *et al.* [9] apresentam uma abordagem chamada *UbiFEX*, que oferece suporte à análise de recursos para reconhecimento de contexto em SPLs. Essa abordagem é dividida em duas partes: a primeira parte concentra no desenvolvimento de uma notação para representar informações de contexto em modelos de características; a segunda é sobre o desenvolvimento de um mecanismo de simulação que visa validar a configuração do produto por meio de alterações de contexto. Por fim, este artigo contribui com a descrição de um estudo experimental, que tem como objetivo avaliar a proposta e implementação de um protótipo.

Bocanegra *et al.* [4] propuseram uma DSL chamada DMLAS, em português, Linguagem Específica de Domínio para projetar Sistemas Adaptativos. Essa linguagem faz parte de um projeto chamado Abordagem Orientada a Modelos para Sistemas Adaptativos (MiDAS) que é uma estrutura para desenvolver softwares adaptáveis usando modelos baseados em MDE (*Model Driven Engineering*). O DMLAS visa dar suporte à estrutura MiDAS, armazenando informações sobre os componentes estáticos e dinâmicos do sistema adaptativo.

Hoyos *et al.* [12] apresentam uma DSL para modelar sistemas Sensíveis ao Contexto. Intitulada *MLContext*, essa linguagem permite modelar atributos e requisitos de qualificação de contexto, além de manter a modelagem de detalhes do contexto e a modelagem de negócios separadas. O projeto implementou um ambiente que permite a transformação de códigos escritos pela DSL em códigos Java e ontologia OWL-DL.

Mostarda *et al.* [18] apresentam um trabalho que busca facilitar o desenvolvimento de aplicativos distribuídos, adaptáveis e confiáveis. Um dos objetivos deste trabalho é diminuir os esforços do desenvolvedor relacionados à descentralização e distribuição. Este projeto desenvolveu uma estrutura que inclui uma linguagem baseada no conceito de máquina de estados, que tem o objetivo de especificar algoritmos adaptativos que são distribuídos dentro de uma plataforma. Esta plataforma visa garantir a tolerância a falhas e uma execução adequada.

8 CONSIDERAÇÕES FINAIS

Este artigo apresentou o desenvolvimento de uma DSML para modelar DSPLs que visa a implementação de sistemas capazes de promover adaptações em tempo de execução. As adaptações são baseadas no conceito de máquina de estados que, neste estudo, desempenham o papel de promover entrada e saída de características. A DSML trabalha, basicamente, com dois tipos de máquinas de estados: por contexto, que lida com a transição de características, e por ação do usuário, que pode incluir ou excluir uma característica opcional implementada como variabilidade para o produto de software.

O desenvolvimento da DSML permitiu a aplicação de conceitos como DSPL, Modelagem de Características e Metamodelagem. Esta pesquisa começou com o desenvolvimento de um metamodelo dividido em Metamodelos de Características, Variabilidades e Derivações. O desenvolvimento do metamodelo também usou o conceito de Arquitetura Orientada a Modelo, resultando na modelagem de DSPLs com base em características. O desenvolvimento da DSML, como resultado final, começa com a transformação dos metamodelos em uma gramática que determina como a linguagem deve ser usada. Com base nesta gramática, regras e restrições são implementadas para determinar um nível mais alto de correção no uso da linguagem. Finalmente, são implementados métodos que interpretam a modelagem realizada em códigos que definem projetos, pacotes e um esqueleto de códigos para a implementação de produtos de software.

Como possíveis limitações do projeto, destacamos a necessidade de um desenvolvimento mais detalhado dos contratos com base nas funções de utilidade. Assim, os trabalhos futuros devem se concentrar em: desenvolver soluções para contratos que implementem funções de utilidade; evoluir a gramática da linguagem, tornando sua escrita mais amigável; desenvolver mais restrições de modelagem para permitir uma modelagem mais precisa dos conceitos de DSPL, máquinas de estado e funções de utilidade; desenvolver uma linguagem gráfica para a DSML, tornando a modelagem mais objetiva; e apresentar uma proposta em ambientes com aplicações complexas para avaliar a escalabilidade do uso da linguagem.

AGRADECIMENTOS

Os autores agradecem à CAPES pelo apoio financeiro durante a pesquisa, à UFG pela oportunidade de realizar este estudo e à UFMT por permitir e apoiar a continuidade do trabalho de pesquisa.

REFERENCES

[1] Osgi Alliance. 2016. OSGi Service Platform. https://www.osgi.org/developer/architecture/.
[2] Apache Software Foundation. 2015. Apace Felix iPOJO. http://felix.apache.org/documentation/subprojects/apache-felix-ipojo.html. acessado em 2015-11-10.
[3] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636.
[4] Jose Bocanegra, Jaime Pavlich-Mariscal, and Angela Carrillo-Ramos. 2015. DMLAS: A Domain-Specific Language for designing adaptive systems. In *Computing Colombian Conference (10CCC), 2015 10th*. IEEE, 47–54.
[5] Booch G., Brown A., Iyengar S., Rumbaugh J., Selic B., An MDA Manifesto, MDA Journal,. 2015. DSM - Domain-Specific Modeling. http://www.dsmforum.org/why.html. acessado em 2015.
[6] H. Borelli. 2016. *Uma linguagem de modelagem de domínio específico para linhas de produto de software dinâmicas*. Master's thesis. http://repositorio.bc.ufg.br/tede/handle/tede/5893 Instituto de Informática - INF (RG).
[7] Sergio Teixeira Carvalho. 2013. *Modelagem de Linha de Produto de Software Dinâmica para Aplicações Ubíquas*. Tese (Doutorado em Computação). Instituto de Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro.
[8] Betty HC Cheng, RogÚrio de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al. 2009. Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*. Springer, 1–26.
[9] Paula Fernandes, Cláudia Werner, and Eldânae Teixeira. 2011. An Approach for Feature Modeling of Context-Aware Software Product Line. *J. UCS* 17, 5 (2011), 807–829.
[10] Walid Joseph Gdon. 2010. OSGi and Apache Felix 3.0 Beginner's Guide. (2010).
[11] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. 2008. Dynamic software product lines. *Computer* 41, 4 (2008), 93–95.
[12] José R Hoyos, Davy Preuveneers, Jesús J García-Molina, and Yolande Berbers. 2011. A DSL for context quality modeling in context-aware applications. In *Ambient Intelligence-Software and Applications*. Springer, 41–49.
[13] Frédéric Jouault, Jean Bézivin, and Ivan Kurtev. 2006. TCS:: a DSL for the specification of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering*. ACM, 249–254.
[14] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. DTIC Document.
[15] Christian Kastner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. 2009. FeatureIDE: A tool framework for feature-oriented software development. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 611–614.
[16] Timo Laakko and Martti Mäntylä. 1993. Feature modelling by incremental feature recognition. *Computer-Aided Design* 25, 8 (1993), 479–492.
[17] B. Meyer. 1992. Applying 'design by contract'. *Computer* 25, 10 (Oct 1992), 40–51. https://doi.org/10.1109/2.161279
[18] Leonardo Mostarda, Daniel Sykes, and Naranker Dulay. 2010. A state machine-based approach for reliable adaptive distributed systems. In *Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on*. IEEE, 91–100.
[19] Kumpati S Narendra and Anuradha M Annaswamy. 2012. *Stable adaptive systems*. Courier Corporation. 9 pages.
[20] O'Brien T., McCulloch S., Demers B. 2015. The Maven Cookbook. http://books.sonatype.com/mcookbook/reference/. acessado em 2015.
[21] OMG. 2016. OMG Meta Object Facility (MOF) Core Specification, Version 2.5.1. https://www.omg.org/spec/MOF.
[22] Bill Schilit, Norman Adams, and Roy Want. 1994. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. IEEE, 85–90.
[23] Douglas C Schmidt. 2006. Guest editor's introduction: Model-driven engineering. *Computer* 39, 2 (2006), 0025–31.
[24] Mark Weiser. 1999. The Computer for the 21st Century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3, 3 (July 1999), 3–11. https://doi.org/10.1145/329124.329126
[25] Jules White, James H Hill, Jeff Gray, Sumant Tambe, Aniruddha S Gokhale, and Douglas C Schmidt. 2009. Improving domain-specific language reuse with software product line techniques. *Software, IEEE* 26, 4 (2009), 47–53.
[26] Xtend Eclipse Framawork. 2015. Xtend is a flexible and expressive dialect of Java. http://eclipse.org/xtend/documentation.html. Acessado em 15-02-2015.
[27] Xtext Eclipse Framawork. 2015. Xtext and Integration with EMF. https://eclipse.org/Xtext/documentation/308_emf_integration.html. acessado em 2014-11-10.
[28] Xtext Eclipse Framawork. 2015. Xtext grammar language extension for Eclipse. http://www.eclipse.org/Xtext/. acessado em 2014-11-10.