Modularization of Variabilities From Software Product Lines of Web Information Systems (Modularizando Variabilidades em Linhas de Produto de Sistemas de Informação Web)

DEMÓSTENES SENA (demostenes.sena@ifrn.edu.br), Federal University of Rio Grande do Norte
FELIPE PINTO (felipe.pinto@ifrn.edu.br), Federal University of Rio Grande do Norte
GLEYDSON LIMA (gleydson@info.ufrn.br), Federal University of Rio Grande do Norte
JADSON SANTOS (jadson@info.ufrn.br), Federal University of Rio Grande do Norte
JALERSON LIMA (jalerson.lima@ifrn.edu.br), Federal University of Rio Grande do Norte
UIRÁ KULESZA (uira@dimap.ufrn.br), Federal University of Rio Grande do Norte
DAVID PEREIRA (david@info.ufrn.br), Federal University of Rio Grande do Norte
VICTOR FERNANDES (victorhcf@gmail.com), Federal University of Rio Grande do Norte
ALEXANDRE VIANNA (strapacao@gmail.com), Federal University of Rio Grande do Norte

---

This paper presents an industrial experience of design pattern application for the modularization of variabilities from software product lines of web information systems. These web systems were developed at the *Superintendência de Informática (SINFO)* from Federal University of *Rio Grande do Norte (UFRN)*. The work describes details about the adoption and composition of traditional design patterns to the modularization of variabilities (optional, alternative and or-feature) that are usually used in web information systems. Furthermore, it also reports some usage scenarios of the conditional execution annotative technique in the implementation of fine-grained variabilities, which are not adequately implemented using traditional design patterns.

Categories and Subjects Descriptors D.2.11 [**Software Architectures**] Patterns

General Terms Design Patterns

Keywords Information Systems, Software Product Lines

ACM Reference Format:
Sena, D. et al. Modularization of Variabilities From Software Product Lines of Web Information Systems. Proceedings of the 9th Latin-American Conference on Pattern Languages of Programming (SugarLoafPloP 2012). September 2012, 15 pages.

---

1. INTRODUÇÃO

Linhas de produto de software [Clements and Northrop 2001, Weiss and Lai 1999] têm se consolidado como uma abordagem de engenharia de software que prioriza a reutilização estratégica de funcionalidades dentro do contexto de famílias de sistemas de software que focalizam um determinado segmento de mercado. Uma linha de produto de software (LPS) pode ser vista como uma família de sistemas que mantém: (i) *features* que são comuns a todos os membros da família, denominadas similaridades; e (ii) *features* que variam de acordo com o membro da família (produto) sendo considerado, chamadas de variabilidades. Uma *feature* representa uma funcionalidade ou propriedade

A preliminary version of this paper was presented in a pattern application at the *9th Latin American Conference on Pattern Languages of Programs (SugarLoafPLoP'12)*, September 20–22, 2012, Natal, Rio Grande do Norte, Brazil.

da LPS que é usada para capturar suas similaridades e distinguir suas variabilidades.

Ao longo dos últimos anos, diversas abordagens de engenharia de linhas de produto de software têm sido propostas [Clements and Northrop 2001, Czarnecki and Eisenecker 2000, Weiss and Lai 1999]. Tais abordagens preconizam a especificação, modelagem e implementação de linhas de produto de software em termos de suas *features* comuns e variáveis. O desenvolvimento de linhas de produto seguindo as diretrizes de tais abordagens é tipicamente organizado em: (i) engenharia de domínio – que objetiva o desenvolvimento de artefatos reusáveis para todos os produtos da LPS; e (ii) engenharia de aplicação – que busca o desenvolvimento de produtos da LPS a partir da reutilização dos artefatos reusáveis definidos na engenharia de domínio. A engenharia de domínio envolve tipicamente atividades de: (i) análise de domínio – responsáveis pela definição do escopo da LPS e da identificação e classificação de suas *features* comuns e variáveis; (ii) projeto de domínio – busca definir uma arquitetura e componentes para a LPS que contemplem as variabilidades elicitadas na análise de domínio; e (iii) implementação de domínio – busca a implementação dos componentes reusáveis que definem a arquitetura da LPS, através do uso de técnicas de implementação de variabilidades.

As abordagens propostas recentemente para engenharia de linhas de produto de software trazem um conjunto de diretrizes gerais para o processo de desenvolvimento de LPS, ajudando a organizá-lo de forma a oferecer um melhor gerenciamento de suas variabilidades. Entretanto, elas são bastante carentes no que se refere ao provimento de técnicas para especificação, modelagem e implementação de variabilidades para domínios específicos. Como consequência, tais abordagens quando adotadas no contexto de desenvolvimento de LPS de domínios específicos precisam ser estendidas e adaptadas para contemplar suas características e peculiaridades, de forma a trazer o impacto esperado para o reuso estratégico de similaridades e variabilidades entre os membros (produtos) de uma LPS.

Neste contexto, este trabalho apresenta uma experiência prática de adoção de técnicas de projeto na modularização de variabilidades de linhas de produto de software (LPS) de sistemas de informação (SI) web que foram desenvolvidos e são mantidos pela Superintendência de Informática (SINFO) da Universidade Federal do Rio Grande do Norte (UFRN), com destaque aos padrões de projeto usados para implementação das variabilidades dessas LPSs. Além do uso de padrões de projeto tradicionais, o artigo também reporta a experiência do uso de técnicas para o tratamento de variabilidades de granularidade fina, no contexto de tais linhas de produto.

Existem trabalhos na literatura que discutem técnicas para implementação de variabilidades em LPSs. Porém, são abordagens mais genéricas, na prática, quando é preciso adotá-las é necessário realizar adaptações. Um exemplo é a proposta apresentada em [Svahnberg and Bosch 2000] que discute algumas técnicas para implementação de variabilidades (herança, pontos de extensão, parametrização e configuração), contudo sem apresentar cenários práticos que ilustrem como essas técnicas seriam usadas. De forma mais abrangente, o trabalho proposto em [Svahnberg et al 2005] apresenta uma taxonomia para um conjunto de diversas técnicas que podem ser usadas para implementar variabilidades, detalhando para cada técnica os seguintes pontos: (i) intenção, (ii) motivação, (iii) solução, (iv) ciclo de vida, (v) consequências, (vi) exemplos. Já o presente trabalho possui um foco mais específico mostrando que a aplicação de padrões de projeto e execução condicional são úteis para implementação de variabilidades, tomando como base a experiência prática vivenciada na SINFO/UFRN.

As seções seguintes do artigo estão organizadas da seguinte forma. A Seção 2 apresenta o contexto de desenvolvimento de linhas de produto de sistemas de informação web pela SINFO/UFRN e a demanda para customização de tais LPSs para diferentes instituições nacionais. A Seção 3 detalha as técnicas para modularização de variabilidades utilizadas na SINFO/UFRN. Finalmente, a Seção 4 apresenta as conclusões do trabalho, assim como trabalhos em andamento e futuros.

2. DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO WEB NA UFRN

A UFRN, através da Superintendência de Informática (SINFO) [SINFO/UFRN 2012], vem desenvolvendo um conjunto de sistemas integrados de gestão de informação [SIG/SINFO 2012] que objetivam a completa informatização da administração universitária da UFRN. Nos últimos anos, esses sistemas vêm se destacando em âmbito nacional, sendo adotados por diversos órgãos da administração pública federal. Eles podem ser agrupados em: (i) sistemas da área fim da universidade, tais como, sistemas acadêmicos; ou (ii) sistemas da área meio, que objetivam a gestão eficiente das áreas de administração, planejamento e gestão de pessoas para atingir com qualidade os objetivos da área fim. Os sistemas de informação desenvolvidos pela UFRN são integrados entre si, garantindo uma melhor comunicação entre os processos administrativos da instituição, e possuem também integração com sistemas do governo federal, para consultas, importação de dados, etc. Dentre os sistemas desenvolvidos pela SINFO/UFRN, destacam-se os seguintes: (i) o SIPAC – Sistema Integrado de Patrimônio, Gestão e Contratos; (ii) o SIGRH – Sistema Integrado de Gestão e Recursos Humanos; e (iii) o SIGAA – Sistema Integrado de Gestão de Atividades Acadêmicas.

O SIPAC (Sistema Integrado de Patrimônio, Gestão e Contratos) oferece operações fundamentais para a gestão das unidades responsáveis pelas finanças, patrimônio e contratos da UFRN, sendo, portanto, atuante nas atividades meio dessa instituição. O sistema SIPAC integra totalmente a área administrativa desde a requisição (material, prestação de serviço, suprimento de fundos, diárias, passagens, hospedagem, material informacional, manutenção de infraestrutura) até o controle do orçamento distribuído internamente.

O SIGRH (Sistema Integrado de Gestão e Recursos Humanos) informatiza os procedimentos de recursos humanos, tais como: marcação/alteração de férias, cálculos de aposentadoria, avaliação funcional, dimensionamento de força de trabalho, controle de frequência, concursos, capacitações, atendimentos on-line, serviços e requerimentos, registros funcionais, relatórios de RH, dentre outros.

O SIGAA (Sistema de Gestão de Atividades Acadêmicas) informatiza os procedimentos da área acadêmica através dos módulos de: graduação, pós-graduação (*stricto sensu* e *lato sensu*), ensino infantil, médio e técnico, submissão e controle de projetos e bolsistas de pesquisa, submissão e controle de ações de extensão, submissão e controle dos projetos de ensino (monitoria e inovações), registro e relatórios da produção acadêmica dos docentes, atividades de ensino a distância, gestão de bibliotecas e um ambiente virtual de aprendizado denominado Turma Virtual.

Os sistemas de informação da UFRN foram implantados durante o ano de 2007 e atualmente são responsáveis pela gerência dos diferentes processos de gestão de informação dentro da instituição. Ao longo dos últimos anos, dado a qualidade e aceitação dos sistemas, a UFRN tem estabelecido contratos de colaboração [SINFO/UFRN 2012] com 17 instituições federais de ensino superior e outros sete órgãos de administração direta federal (entre eles, Ministério da Justiça, Polícia Federal, Agência Brasileira de Inteligência e Controladoria Geral da União) para adaptação e customização dos sistemas de informação produzidos na UFRN para sua realidade e contexto. A demanda para adaptação e customização para cada uma de tais instituições tem requerido a adoção de técnicas de engenharia de linhas de produto de forma complementar as técnicas já adotadas. O objetivo da adoção de tais técnicas é promover o desenvolvimento de cada um dos sistemas existentes como uma LPS, através de um adequado gerenciamento de suas variabilidades.

Devido a restrições de espaço, várias informações relacionadas aos sistemas não podem ser apresentadas em detalhes no artigo, porém detalhes adicionais podem ser obtidos em [SIG/SINFO 2012] e [SINFO/UFRN 2012]. A Tabela 1 mostra alguns dados dos sistemas para o contexto da UFRN.

3. MODULARIZANDO VARIABILIDADES EM LINHAS DE PRODUTO

Esta seção apresenta a experiência de adoção de técnicas de modularização de variabilidades no contexto de linhas de produto de software de sistemas de informação web na SINFO/UFRN. O objetivo de adotar tais técnicas é atender as demandas de adaptação e customização dos sistemas para a realidade de diferentes instituições. Neste contexto, cada sistema desenvolvido pela SINFO/UFRN representa uma linha de produto de software que requer o gerenciamento apropriado das suas variabilidades de forma a permitir a customização para diferentes clientes derivando uma versão da LPS (produto) que atenda suas necessidades. Embora, todas as linhas de produto da SINFO/UFRN (Seção 2) utilizem as técnicas apresentadas, por questões de restrições de espaço, esta seção utiliza apenas o SIGAA para ilustrar o uso de tais técnicas.

Tabela 1. Algumas métricas sobre os sistemas da SINFO/UFRN (valores médios aproximados).

Sistema	Usuários Ativos	Usuários Online Simultaneamente	Linhas de Código	Número de Classes	Número de Métodos
SIPAC	7.527	150	803.000	4.758	60.219
SIGRH	5.099	50	432.000	2.588	41.790
SIGAA	52.000	1000	685.000	4.778	66.975

A Subseção 3.1 apresenta uma visão parcial das variabilidades existentes no SIGAA. A Subseção 3.2 descreve a arquitetura geral adotada pelas LPSs de SIs Web da SINFO, exemplificando com a arquitetura do SIGAA. Finalmente, a Subseção 3.3 apresenta o conjunto de técnicas composicionais e anotativas usadas para implementação dos diferentes tipos de variabilidades identificadas nas linhas de produto da SINFO/UFRN.

3.1. Variabilidades de Linhas de Produto de Sistemas de Informação Web

A especificação de requisitos dos sistemas da SINFO é realizada através de casos de uso, os quais são atualmente documentados em um sistema *wiki*. Cada caso de uso expressa uma funcionalidade relacionada a um determinado processo de negócio, cuja especificação é refinada com um conjunto de regras de negócio e customizações específicas que podem variar de acordo com a instituição sendo considerada. Cada uma dessas instituições requer, portanto, diferentes customizações nos sistemas de informação produzidos, tais como: (i) definir suas próprias regras de negócio; (ii) modificar passos na execução de *workflows* dos processos de negócio (institucionais); (iii) assim como customizar sua *interface* gráfica dependendo dos tipos de usuários e variabilidades habilitadas para o produto desejado.

Com a finalidade de promover o gerenciamento das variabilidades existentes nos casos de uso da SINFO/UFRN e caracterizar cada uma das suas linhas de produto, uma das atividades constituintes do processo é a modelagem e especificação do modelo de *features* das LPSs da UFRN, o qual é essencial para prover o gerenciamento efetivo de suas variabilidades. A Figura 1 apresenta uma visão parcial do modelo de *features* do SIGAA (modelado usando o *Feature Modeling Plugin* – FMP [Antkiewicz and Czarnecki 2004]).

Esse modelo foi gerado usando informações tanto da especificação de requisitos do sistema quanto do código que implementa cada uma de suas *features*. Devido restrições de espaço, não é possível apresentar o modelo de *features* completo de cada LPS e, embora usemos apenas o SIGAA ao longo do artigo como exemplo, as demais linhas de produto (SIPAC e SIGRH) seguem a mesma estratégia.

A Figura 1 apresenta diferentes tipos de *features* do SIGAA, tais como: (i) *obrigatório* – autenticação, por exemplo, sempre existe; (ii) *opcional* – possibilita que componentes curriculares sejam cadastrados com flexibilidade de horário; (iii) *alternativo* – diferentes opções de autenticação (banco de dados ou diretórios LDAP); (iv) *or-feature* – diferentes tipos de cálculos de índices para

alunos de graduação (onde mais de um tipo de *feature* pode ser escolhido dentre várias alternativas); e (v) *parametrizável* – cálculo no prazo entre empréstimos (este tipo de *feature* deve ser parametrizado com algum tipo de valor específico, tal como um número inteiro, por exemplo). A organização das diferentes *features* no modelo foi realizada mapeando cada uma delas para ser parte dos casos de uso que estão relacionadas. Isso significa que a maioria das *features* dessas LPSs representam variações que ocorrem durante a execução de um caso de uso.

Ainda na Figura 1, por exemplo, podemos verificar que a *feature* Permite Flexibilidade está localizada dentro do caso de uso Grade de Horários. Da mesma forma, diferentes tipos de cálculos de índices (Eficiência Acadêmica, Rendimento Acadêmico, etc) podem ser escolhidos dentro do caso de uso Índices Acadêmicos. Essa mesma regra está sendo usada para modelar as demais linhas de produto de sistemas de informação da SINFO/UFRN, demonstrando ser uma diretriz bastante útil para modelar variações em sistemas de informação. É importante ressaltar que a relação entre *features* e casos de uso é extremamente dependente do domínio da LPS sendo desenvolvida [Chastek et. 2001]. No caso específico das linhas de produto da SINFO/UFRN, cada caso de uso mantém um conjunto de variabilidades dentro da sua estrutura. É interessante notar, entretanto, que existem *features* que não estão necessariamente relacionados a apenas um caso de uso, como por exemplo, Autenticação e Estratégias de Consolidação de Turmas.

3.2. Arquitetura e Projeto de Domínio

A arquitetura adotada pelas linhas de produto da SINFO segue o padrão arquitetural em camadas [Buschmann et al. 1996]. Os artefatos de implementação são organizados de acordo com sua finalidade e estruturados em quatro camadas principais, sendo elas:

- **(i)** **Camada de apresentação** (*Graphical User Interface*) – que trata dos aspectos de interação com o usuário (*controller*), através de um conjunto de classes *MBeans* do *framework Java Server Faces* (JSF), e a captação e exibição das informações (*view*) usando páginas *Java Server Pages* (JSP) codificadas com bibliotecas de *tags* JSF. O padrão *Model-View-Controller* (MVC) [Buschmann et al. 1996] é também utilizado para organizar a camada de apresentação em visualizações e controladores e reger sua comunicação com a camada de serviços;
- **(ii)** **Camada de serviços** (*Service*) – define os serviços oferecidos pela aplicação e delimita as transações a serem executadas pelo sistema através da sua implementação com *Enterprise Java Beans* (EJB). Nesta camada, o padrão de projeto EJB *Command* foi aplicado para criar uma fachada de ativação das classes de lógica de negócio através de objetos comando e classes processadoras;
- **(iii)** **Camada de negócio** (*Business*) – contém as classes processadoras e de entidades do sistema que representam o domínio do negócio e são responsáveis pela implementação de várias regras de negócio;
- **(iv)** **Camada de dados** (*Data*) – mantém as classes de acesso ao banco de dados (DAO – *Data Access Object*) que utilizam um *framework* de Mapeamento Objeto Relacional (MOR) para atualizar e recuperar informações do banco de dados. O *framework Hibernate* é usado para realizar o mapeamento dos objetos para o banco de dados e vice-versa.

3.3. Técnicas de Modularização de Variabilidades

A arquitetura atual adotada promove a modularização das principais funcionalidades e oferece flexibilidade para a implementação de novos casos de uso de cada uma das LPSs de SIs da SINFO. Apesar destes benefícios, ela não é suficiente para atender o projeto e implementação modular das variabilidades presentes nas LPSs.

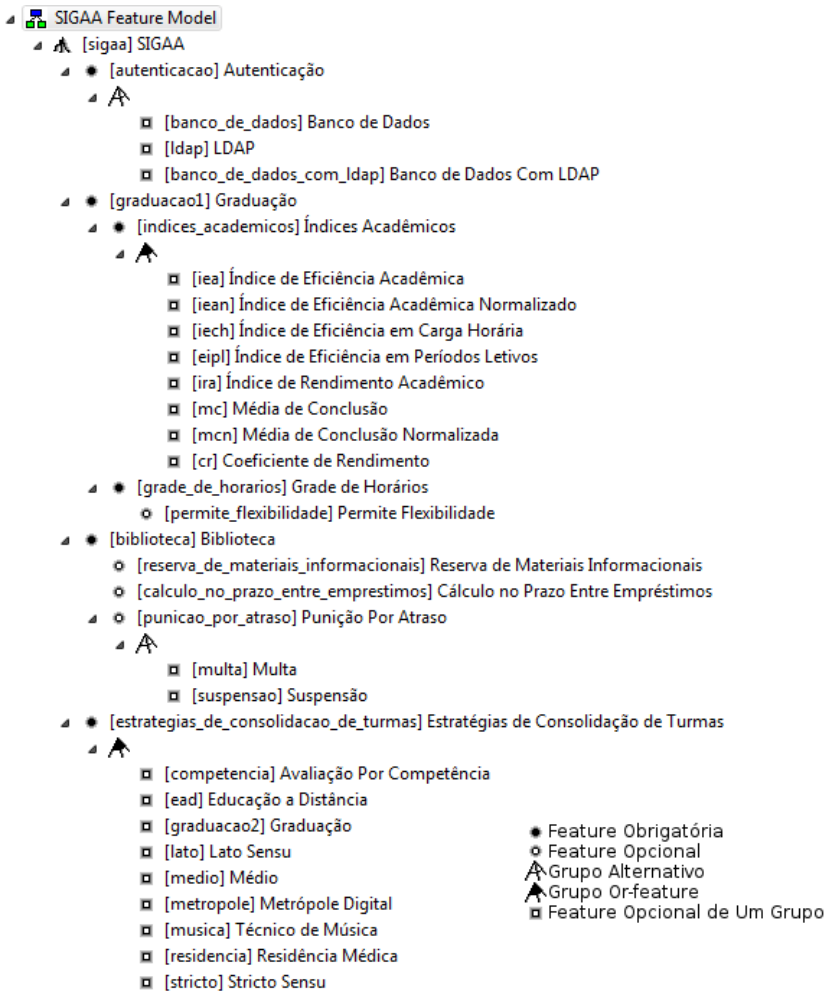

Figura 1. Modelo de *features* parcial da LPS SIGAA.

De forma a atender tal requisito, técnicas de modularização baseadas em padrões de projeto foram aplicadas para implementar os diferentes tipos de variabilidades demandados para a customização de cada LPS. Porém, em determinadas situações a modularização de variabilidades é uma tarefa que pode ser bastante complexa, por exemplo, quando a variabilidade envolve trechos específicos de código espalhado em um ou mais artefatos. Em casos desse tipo, as variabilidades foram implementadas com uso de estruturas condicionais. Dessa forma, as técnicas utilizadas para modularização das LPSs podem ser divididas em duas categorias [Kästner et al 2008]: (i) composicionais – baseadas na adoção de padrões de projeto tradicionais; e (ii) anotativas – que utilizam algum mecanismo de decisão, como a compilação condicional ou a execução condicional. As subseções seguintes detalham o uso das técnicas usadas no contexto da LPS SIGAA para tratar diferentes tipos de variabilidades.

3.3.1. Técnicas Composicionais

As técnicas composicionais são baseadas na aplicação de padrões de projeto [Gamma et al 1995] para modularizar cada variabilidade, podendo ocorrer a combinação de diferentes padrões de projeto para tratar uma determinada variabilidade. Exemplos de padrões que foram usados no contexto da SINFO/UFRN para implementar variabilidades são: *Strategy*, *Chain of Responsibility*, *Template Method*, entre outros. A

Tabela **2** mostra alguns tipos de variabilidades da LPS SIGAA que são tratadas através de técnicas composicionais.

Tabela 2. Tipos de variabilidade e técnicas composicionais.

Tipo de Variabilidade	Técnica de Implementação	Exemplos da LPS SIGAA
Feature Alternativa	Padrão *Strategy*	Autenticação; processamento de multas de empréstimo de livros; estratégias de consolidação de turmas
Feature OR (*or-feature*)	Padrão *Chain of Responsibility*	Cálculo de índices acadêmicos para alunos

Padrão *Strategy*

O padrão *Strategy* foi aplicado nas variabilidades que necessitam da escolha de um comportamento dentre um conjunto de comportamentos possíveis. Dessa forma, ele é usado para modularizar *features* alternativas existentes nas LPSs da SINFO/UFRN. Neste contexto, cada comportamento alternativo é implementado por uma estratégia e somente uma estratégia deve ser selecionada, caracterizando assim uma *feature* alternativa.

A Figura 2 apresenta um diagrama de classes com a modularização da *feature* alternativa Autenticação do SIGAA usando o padrão de projeto *Strategy*. Esta *feature* oferece três possibilidades de autenticação do usuário: (i) com banco de dados; (ii) com LDAP (*Lightweight Directory Access Protocol*); e, (iii) com banco de dados e LDAP. Para implementar cada comportamento representado pelas *features* de autenticação, temos uma classe que implementa a *interface* EstrategiaAutenticacao. A estrutura das classes apresentada segue o padrão *Strategy*, sendo que a estratégia de autenticação é determinada pelo atributo ESTRATEGIA_AUTENTICACAO da classe de domínio ConstantesParametroGeral, presente na camada de negócio, cujo valor foi definido na engenharia de aplicação, ou seja, durante a customização da LPS para uma dada instituição.

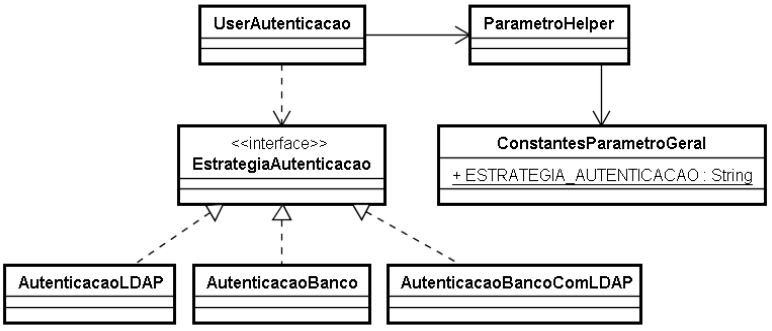

Figura 2. Implementação da *feature* associada à autenticação.

Padrão *Chain of Responsability*

O padrão *Chain of Responsability* foi aplicado nas LPSs da SINFO/UFRN para implementar algumas das variabilidades do tipo *or-feature*. Este tipo de *feature* é um caso particular de *feature* alternativa, que permite a seleção de não apenas uma das *features* pertencentes ao grupo alternativo de *features*, mas sim a seleção de mais de uma *feature* desse grupo, ou até mesmo nenhuma delas.

No contexto da LPS SIGAA, um exemplo de *or-feature* são os cálculos de índices acadêmicos. Esses cálculos podem ser compostos por diversas etapas e essas etapas podem variar de uma instituição para outra, ou dentro da mesma instituição dependendo do nível de ensino. Através da aplicação do padrão *Chain of Responsability* é possível executar uma cadeia de classes que implementam um determinado processamento. As classes são executadas em uma ordem pré-definida e ao término da execução de uma, inicia-se o processamento da classe seguinte. Para criar um novo nó na cadeia de responsabilidades, deve-se criar uma classe que estende a classe base dos nós da cadeia e definir a sua ordem de execução dentro da cadeia.

Como exemplo, temos a *feature* Índices Acadêmicos que possui um grupo *or-feature* com oito *subfeatures*: (i) Índice de Eficiência Acadêmica, (ii) Índice de Eficiência Acadêmica Normalizada, (iii) Índice de Eficiência em Carga Horária, (iv) Índice de Eficiência em Períodos Letivos, (v) Índice de Rendimento Acadêmico, (vi) Média de Conclusão, (vii) Média de Conclusão Normalizada e (viii) Coeficiente de Rendimento. Neste caso particular, temos a opção de selecionar até oito *subfeatures* do grupo.

Cada *subfeature* do grupo *or-feature* de Índices Acadêmicos representa a ativação do cálculo de um índice acadêmico diferente. Para tratar esse tipo de configuração, foi usado em conjunto com o padrão *Chain of Responsibility* parâmetros *booleanos* para indicar quais *features* de índices foram selecionadas. O cálculo de cada índice é modularizado dentro de uma classe separada, assim, em tempo de execução, verifica-se se a *feature* foi selecionada e, em caso positivo, cria-se um objeto da classe correspondente ao índice.

Uma vez que determinado conjunto de índices é ativado, a atualização dos seus valores ocorre junto com outros cálculos relacionados ao discente usando o padrão *Chain of Responsibility*. A Figura 3 mostra a aplicação de tal padrão de projeto na *feature* Índices Acadêmicos.

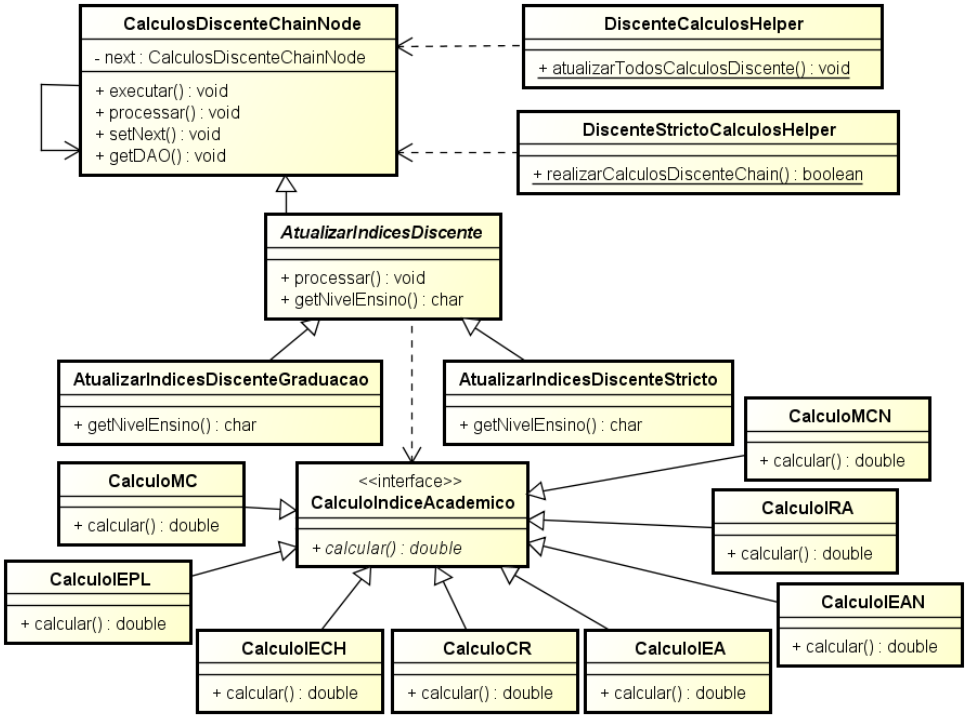

Figura 3. Implementação das variabilidades do cálculo de índices acadêmicos.

Como ilustrado na Figura 3, as classes DiscenteCalculosHelper e DiscenteStrictoCalculosHelper possuem o método para atualizar os cálculos referentes aos discentes.

Cada cálculo é processado em um objeto do tipo CalculosDiscenteChainNode que após concluir sua tarefa executa o próximo objeto, até que todos os cálculos sejam processados. A classe responsável pelos cálculos dos índices é AtualizarIndicesDiscente que estende CalculosDiscenteChainNode e possui duas especializações, uma para graduação (AtualizarIndicesDiscenteGraduação) e outra para pós-graduação (AtualizarIndicesDiscenteStricto).

Cada índice é implementado em uma classe de cálculo separada pertencente a camada de negócio da LPS. CalculosDiscenteChainNode pertence a camada de dados e permite a atualização dos novos valores por parte da classe da camada de negócio AtualizarIndicesDiscente, através de um objeto de acesso a dados fornecido pelo método getDAO().

3.3.2. Técnicas Anotativas

As técnicas anotativas são implementadas com uso de estruturas condicionais e atendem as variabilidades de granularidade fina que estão espalhadas no código de classes das LPSs da SINFO/UFRN. Tradicionalmente, técnicas anotativas fazem uso de estruturas condicionais que permitem tratar as variabilidades em tempo de compilação (compilação condicional), porém, uma abordagem dinâmica também é possível, na qual as variabilidades são tratadas apenas em tempo de execução (execução condicional).

Nas LPSs de SIs Web da SINFO/UFRN são usadas técnicas anotativas para tratar diferentes tipos de *features*, aplicando-se uma abordagem de execução condicional (na maior parte, *ifs* espalhados pelo código). Assim, o processo de decisão de cada variabilidade implementada com técnicas anotativas, ocorre em tempo de execução. Em geral, as informações usadas para verificar se determinadas variabilidades serão ativadas ou com que valores serão parametrizadas, estão armazenadas em um banco de dados. Em tempo de execução é feita uma consulta ao repositório persistente para decidir como será tratada a variabilidade. A Tabela 3 mostra alguns exemplos.

Tabela 3. Tipos de variabilidade e técnicas anotativas.

Tipo de Variabilidade	Técnica de Implementação	Exemplos da LPS SIGAA
Feature Opcional	Execução Condicional	Componentes com flexibilidade de horário; possibilidade de reserva de materiais; tratamento de irregularidades administrativas
Feature Opcional Parametrizável	Configuração de parâmetro no código	Tempo limite entre empréstimos de mesmo material

Execução Condicional

Para o caso de *features* opcionais, a técnica anotativa aplicada nas LPSs da SINFO/UFRN é a execução condicional. Dessa forma, parâmetros são verificados através de comandos condicionais que determinam, em tempo de execução, se uma determinada variabilidade foi incluída ou não. Essa solução condiciona o código opcional através de um parâmetro, em geral, do tipo *booleano*. Dessa forma, são usados comandos condicionais para verificar a configuração de tais parâmetros, e decidir em tempo de execução se a *feature* foi incluída ou não, caracterizando assim a abordagem padrão de execução condicional.

Exemplos de *features* opcionais que foram implementadas usando essa técnica no SIGAA são: (i) Permite Flexibilidade – possibilita que componentes curriculares sejam cadastrados com flexibilidade de horário, ou seja, o horário pode mudar ao longo do período de duração das turmas nas quais o componente é oferecido; (ii) Reserva de Materiais Informacionais – permite que o módulo de

biblioteca trabalhe com a possibilidade de realizar reservas para os diversos tipos de mídias, como CDs, DVDs, fitas VHS e livros; e (iii) Irregularidade Administrativa na Biblioteca – possibilita que determinadas funcionalidades sejam bloqueadas para os usuários que extrapolem o limite de dias de atraso permitidos no empréstimo de materiais informacionais da biblioteca.

Nestes casos, existe uma ou mais classes com atributos *booleanos* que parametrizam a presença ou ausência da *feature* na customização da LPS (produto) disponibilizada para cada instituição. Por exemplo, para a *feature* Permite Flexibilidade, a classe chamada HorarioTurmaMBean da camada de apresentação define um método que verifica se há flexibilidade na grade de horários. Para isso, este método verifica atributos de várias classes da camada de negócio que representam os níveis de ensino (*Stricto Senso*, *Lato Senso*, Graduação e Técnico). A Figura 4 ilustra tal situação.

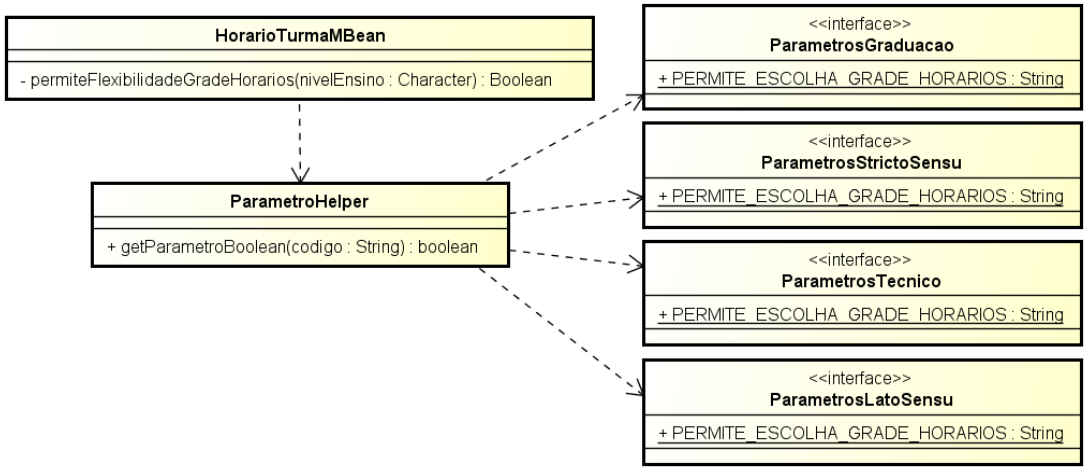

Figura 4. Modularização da variabilidade de flexibilidade na grade de horários.

Parâmetro de Configuração

Uma variação da situação apresentada anteriormente para a flexibilidade de horário ocorre quando a *feature* opcional pode ser parametrizada por um valor qualquer, por exemplo, um inteiro ou *String*. Um bom exemplo de *feature* opcional parametrizada presente no SIGAA é o Cálculo no Prazo Entre Empréstimos que determina o cálculo para o prazo mínimo que o usuário da biblioteca precisa esperar para poder realizar o empréstimo do mesmo material novamente. Esta *feature* é parametrizada por um tipo inteiro que representa o valor em horas indicando o prazo mínimo entre empréstimos do mesmo material para o mesmo usuário. Atribuindo-se um valor menor ou igual a zero para essa *feature*, ela é desativada, podendo o usuário realizar empréstimos sucessivos de um mesmo material sem precisar esperar um tempo mínimo entre eles. Esta técnica, em especial, é conhecida por parâmetro de configuração.

A estratégia em si é semelhante à execução condicional mostrada anteriormente e, inclusive, podemos dizer que esta técnica também faz uso de execução condicional, pois além de parametrizar o valor da *feature* é preciso verificar através de testes condicionais durante a execução se a variabilidade estará presente.

A Figura 5 exemplifica o diagrama de classes da *feature* Cálculo no Prazo Entre Empréstimos, ilustrando o uso de tal técnica. A classe ProcessadorRealizaEmprestimo, presente na camada de negócio, possui o método verificaTempoLimiteEntreEmprestimos() que verifica se o prazo mínimo entre empréstimos do mesmo material é maior que zero, habilitando a *feature* neste caso, e, em seguida, verifica novamente o valor da *feature* para determinar se o prazo mínimo informado foi

respeitado para o material e o usuário em questão. Caso o prazo mínimo não seja respeitado, o sistema não realiza o empréstimo do material para aquele usuário.

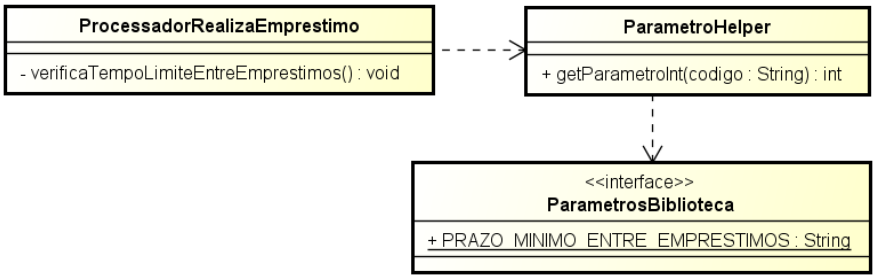

Figura 5. Modularização da variabilidade associada ao tempo limite entre empréstimos de mesmo material.

3.3.3 Integração de Técnicas

As técnicas composicionais são baseadas na aplicação de padrões de projeto para modularizar cada variabilidade, enquanto as técnicas anotativas fazem uso de estruturas condicionais. Devido à complexidade de certas variabilidades, há, em alguns casos, a necessidade de integrar diferentes técnicas para tratá-las, podendo ocorrer à combinação de diferentes padrões de projeto e/ou a aplicação combinada de técnicas composicionais e anotativas. Diversas variabilidades das LPSs de SIs Web da SINFO/UFRN combinam o uso das técnicas. A Tabela 4 apresenta exemplos no contexto do SIGAA.

Tabela 4. Tipos de variabilidade e integração de técnicas.

Tipo de Variabilidade	Técnica de Implementação	Exemplos da LPS SIGAA
Feature alternativa opcional	Padrão *Strategy* e parâmetro de configuração	Punição por atraso
Feature OR (*or-feature*)	Padrão *Strategy* e *Template Method*	Estratégias de consolidação de turmas

Padrão *Strategy* e Parâmetro de Configuração

A combinação do padrão *Strategy* com a técnica anotativa de parâmetro de configuração foi aplicada diversas vezes nas LPSs da SINFO/UFRN. Um contexto de uso desta combinação pode ser visualizado da seguinte maneira: (i) padrão *Strategy* fornece um conjunto de possibilidades de comportamentos que modularizam uma *feature*; e (ii) o parâmetro de configuração permite a injeção de parâmetros compartilhados pelos elementos do padrão *Strategy*. Diversas *features* alternativas opcionais foram modularizadas aplicando essa combinação. Os comportamentos das *features* alternativas opcionais e as *features* alternativas se assemelham, porém, nas *features* alternativas opcionais tem-se a possibilidade de que nenhuma *subfeature* seja selecionada para fazer parte do produto customizado da LPS.

Na LPS SIGAA, existem algumas *features* alternativas opcionais, por exemplo, funcionalidades de gerenciamento de empréstimo dos materiais da biblioteca (tais como, livros, periódicos, etc). Uma funcionalidade desta categoria é a punição por atraso de empréstimo, ou seja, até o término do período de empréstimo, o usuário deverá realizar a devolução e caso o período seja extrapolado, poderá existir uma punição para o usuário. Na implementação da LPS SIGAA, ocorrendo à inclusão de uma punição, existem duas modalidades: (i) pagamento em dinheiro de uma multa; e (ii) período de suspensão, tempo em que o usuário fica impossibilitado de realizar empréstimo na biblioteca. Nas duas modalidades, o tempo extrapolado, a partir da data limite prevista para devolução, determina a quantia da multa ou o tempo de suspensão. Por este motivo, observe que além da definição da estratégia da punição (padrão *Strategy*), a definição da multa ou do tempo de suspensão devem ser

definidos na LPS (parâmetro de configuração). Assim, o padrão *Strategy* implementa as possibilidades de punição (multa ou suspensão), sendo que para cada possibilidade de punição existe uma estratégia implementada.

No que diz respeito ao caráter opcional desta *feature*, na sua implementação foi aplicado um valor como parâmetro de configuração. Este parâmetro determina se algum comportamento de punição por atraso será aplicado, e, caso o parâmetro esteja habilitado, então este indicará a estratégia que implementa a regra de punição.

A Figura 6 apresenta o diagrama de classes parcial de implementação da *feature* Punição Por Atraso. As classes de negócio PunicaoAtrasoEmprestimoStrategy, SuspensaoStrategyDefault e MultaStrategy implementam os aspectos alternativos de tal *feature*. Já a classe ParametrosBiblioteca possui os parâmetros relativos a configuração de algumas funcionalidades da biblioteca do SIGAA, entre elas, SISTEMA_TRABALHA_COM_MULTA e SISTEMA_TRABALHA_COM_SUSPENSAO, responsáveis por indicar as regras de punição.

Padrão *Strategy* e *Template Method*

Um caso interessante de integração de padrões na LPS SIGAA para tratar certas características de variabilidades ocorre na consolidação de turmas. Neste caso, a *feature* Estratégias de Consolidação de Turmas define um grupo de *features* indicando as possíveis estratégias suportadas para realizar a consolidação das turmas.

As estratégias de consolidação de turmas são especificadas em um grupo *or-feature*, sendo possível escolher de uma até nove *subfeatures* do grupo, sendo elas: (i) Avaliação Por Competência, (ii) Educação a Distância, (iii) Graduação, (iv) Lato Sensu, (v) Médio, (vi) Metrópole Digital, (vii) Técnico de Música, (viii) Residência Médica; e (ix) Stricto Sensu. Em geral, temos estratégias de consolidação, diferentes para turmas de diferentes níveis de ensino, por exemplo, as *features* Graduação e Stricto Sensu, para turmas de graduação e de *Stricto Sensu*, respectivamente. Da mesma forma, podem existir estratégias que não representam um nível de ensino, mas apenas uma situação particular de consolidação. Exemplos de tal situação são as *features* Técnico de Música e Residência Médica, que representam, respectivamente, as regras para consolidação de turmas de ensino técnico de música e as regras de consolidação para residência médica.

A solução adotada pelo SINFO para lidar com a modularização de tal *feature* foi usar a integração dos padrões de projeto *Strategy* e *Template Method*. O padrão *Strategy* permite definir diferentes classes que implementam as diferentes estratégias de consolidação de turmas. O algoritmo que realiza a consolidação das turmas foi construído de forma que o mesmo trecho de código possa ser aplicado para consolidar turmas com qualquer estratégia, ocorrendo variações apenas em determinados métodos usados durante o processo de consolidação que são justamente fornecidos pelas implementações de cada *Strategy*. Em outras palavras, o método que executa a consolidação funciona como um *template method*, variando alguns métodos usados no processo de acordo com o *Strategy* escolhido, caracterizando o padrão *Template Method*. A Figura 7 apresenta o diagrama de classes parcial da implementação de tal *feature*.

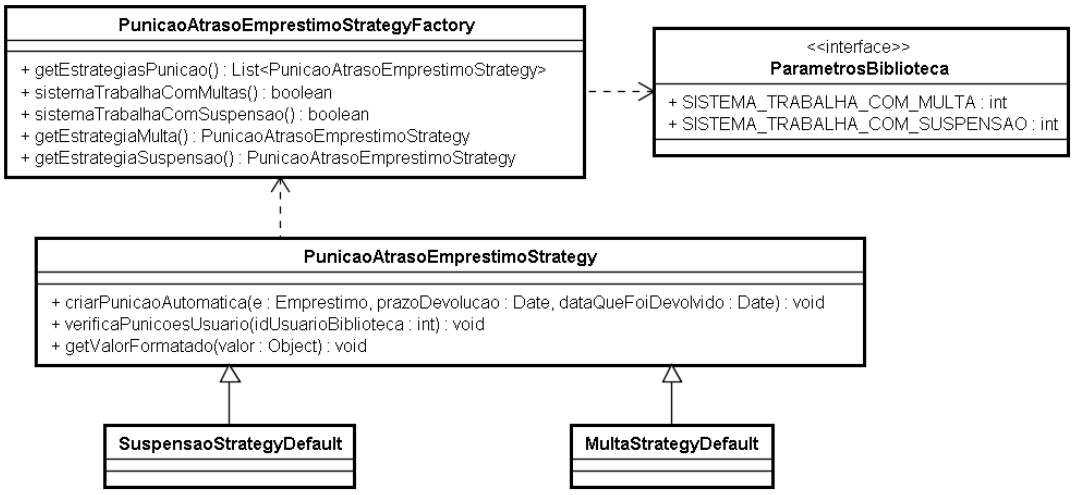

Figura 6. Modularização das variabilidades relacionadas à punição por atraso.

Conforme ilustrado na Figura 7, cada estratégia de consolidação, representada pela *interface* EstrategiaConsolidacao, é concretizada por uma classe da camada de negócio isolada (ConsolidacaoLato, ConsolidacaoGraduacao, ConsolidacaoEad, ConsolidacaoStricto, etc) que estende AbstractEstrategiaConsolidacao. Esta classe contém alguns métodos que são comuns a todas as estratégias concretas. A *interface* ProcessadorComando define os métodos execute() e validate(), sendo estes implementados na classe de negócio ProcessadorConsolidacaoTurma. O método execute() é o *template method* responsável pela execução da consolidação de qualquer turma e durante sua execução são invocados métodos definidos na *interface* EstrategiaConsolidacao e implementados por cada estratégia de consolidação, por exemplo, os métodos consolidar() e calculaMediaFinal().

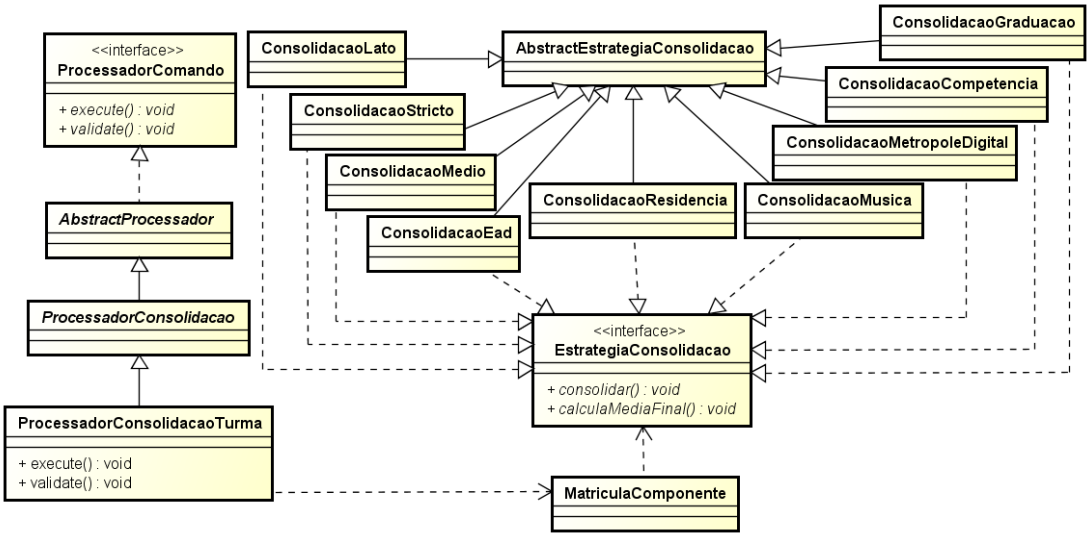

Figura 7. Implementação parcial das estratégias de consolidação de turmas.

4. CONCLUSÕES E PERSPECTIVAS FUTURAS

Este artigo apresentou a experiência da adoção de técnicas e padrões para tratar variabilidades em linhas de produtos de sistemas de informação web, no contexto das LPSs desenvolvidas pela SINFO/UFRN. Atualmente, uma das principais motivações para a SINFO estar trabalhando com

técnicas de engenharia de linhas de produto é a real necessidade de distribuir diferentes versões dos sistemas desenvolvidos para seus clientes, incluindo instituições de ensino e administrativas, que possuem diferentes necessidades relacionadas às regras de negócios implementadas.

O trabalho teve como foco principal apresentar técnicas aplicadas pela SINFO para tratar variabilidades, mostrando que a adoção e composição de padrões de projeto tradicionais são úteis para implementar diferentes tipos de variabilidades tipicamente encontradas em sistemas de informação web. Neste sentido, alguns padrões de projeto como *Strategy*, *Chain of Responsability* e *Template Method* são bastante úteis como foi relatado no artigo através da experiência prática, especificamente no SIGAA, LPS da qual os exemplos foram retirados.

Entretanto, além de abordar a aplicação de padrões de projeto, o trabalho também reporta cenários com uso de técnicas anotativas, como a execução condicional, para implementar variabilidades de granularidade fina, que são difíceis de modularizar com padrões de projeto tradicionais. Dessa forma, ficou perceptível que técnicas composicionais permitem a modularização de variabilidades comportadas, ou seja, que ocorrem isoladas e são de granularidade grossa, podendo ser encapsuladas sem produzir efeitos colaterais nos outros artefatos da linha de produto e sem causar excessivo aumento de complexidade no código da mesma. Porém, quando a variabilidade não é comportada, estando espalhada em vários locais da LPS ou mesmo quando isolada em um único ponto, mas requer tratamento de granularidade mais fina, as técnicas anotativas surgem como uma boa opção, sendo aplicadas com sucesso pela SINFO em suas linhas de produto.

A experiência prática da SINFO tem demonstrado que o uso de diferentes técnicas de implementação para tratar diferentes tipos de variabilidades é fundamental para lidar com a complexidade de modularização de variabilidades em arquiteturas em camadas de sistemas de informação. Dessa forma, algumas pesquisas [Apel and Batory 2007, Kästner et al 2008] mostram que técnicas composicionais são apropriadas para diversos domínios de LPS, mas existem diversos outros cenários que requerem o uso de técnicas anotativas, como explicado anteriormente. No caso específico das LPSs da SINFO/UFRN, a experiência tem mostrado que as duas técnicas não são exclusivas, mas sim o contrário, a combinação e integração das técnicas composicionais e anotativas é a mais adequada para tratar os tipos heterogêneos de variabilidades existentes em tais sistemas de informação.

Algumas linhas de trabalhos futuros estão sendo exploradas atualmente, de forma complementar ao uso das técnicas de modularização apresentadas no artigo, dentre elas: (i) definição de linguagens específicas de domínio (DSL – *Domain-Specific Languages*) [Czarnecki and Eisenecker 2000] para facilitar o processo de derivação automática de novos formulários web que representam implementações de processos de negócio dentro da SINFO; (ii) a definição de diretrizes e idiomas de implementação para variabilidades dinâmicas de granularidade fina ou grossa; (iii) o refinamento do processo da SINFO de forma a contemplar atividades explícitas de engenharia de linhas de produto. Atualmente, as atividades vêm sendo realizadas por uma equipe separada formada pelos arquitetos responsáveis pelas linhas de produto.

REFERENCES

Antkiewicz, M., and K. Czarnecki. 2004. “FeaturePlugin: Feature Modeling Plug-In for Eclipse”, The 2004 OOPSLA Workshop on Eclipse Technology eXchange - Eclipse '04, Vancouver, British Columbia, Canada, ACM Press, pp. 67 - 72.

Apel, S. and Batory, D. 2007. When to Use Features and Aspects? A Case Study. In Proceedings of the GPCE 2007, pages 59–68. ACM Press.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. 1996. Pattern-Oriented Software Architecture, A System of Patterns, vol. 1. Wiley.

Chastek, G., Donohoe, P., Kang, K., Thiel, S. 2001. Product Line Analysis: A Practical Introduction, Technical Report, CMU/SEI-2001-TR-001.

Clements, P., Northrop, L. 2001. “Software Product Lines: Practices and Patterns”, Addison-Wesley Professional.

Czarnecki, K., Eisenecker, U. 2000. “Generative Programming: Methods, Tools, and Applications”, Addison-Wesley.

Gamma, E., Helm, Richard., Johnson, R., Vlissides, J. 1995. “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Kästner, C., Apel, S., Kuhlemann, M. 2008. “Granularity in software product lines”. In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, 311-320.

SIG/SINFO 2012. Sistemas Institucionais Integrados de Gestão da Superintendência de Informática da UFRN 2012: Disponível em: http://www.info.ufrn.br/wikisistemas.

SINFO/UFRN 2012. Superintendência de Informática da UFRN 2012. Disponível em: http://www.info.ufrn.br.

Weiss, D., Lai, C. 1999. “Software Product-Line Engineering: A Family-Based Software Development Process”, Addison-Wesley Professional.

Svahnberg, M., Bosch, J. 2000. “Issues Concerning Variability in Software Product Lines”. In Proceedings of the International Workshop on Software Architectures for Product Families (IW-SAPF-3), Frank van der Linden (Ed.). Springer-Verlag, London, UK, 146-157.

Svahnberg, M., Gurp, J., Bosch, J. 2005. “A taxonomy of variability realization techniques”. Research Articles. Softw. Pract. Exper. 35, 8 (July 2005), 705-754. DOI=10.1002/spe.v35:8 http://dx.doi.org/10.1002/spe.v35:8.