EDSON ALVES DE OLIVEIRA JUNIOR

UM PROCESSO DE GERENCIAMENTO DE VARIABILIDADE PARA LINHA DE PRODUTO DE SOFTWARE

MARINGÁ
2005

EDSON ALVES DE OLIVEIRA JUNIOR

UM PROCESSO DE GERENCIAMENTO DE VARIABILIDADE PARA LINHA DE PRODUTO DE SOFTWARE

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

Orientadora: Profª. Drª. Itana Maria de Souza Gimenes

MARINGÁ

2005

EDSON ALVES DE OLIVEIRA JUNIOR

UM PROCESSO DE GERENCIAMENTO DE VARIABILIDADE PARA LINHA DE PRODUTO DE SOFTWARE

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

Aprovado em 15/04/2005.

BANCA EXAMINADORA

Prof. Dr. Sérgio Roberto Pereira da Silva
Universidade Estadual de Maringá – DIN/UEM

Profª. Drª. Elisa Hatsue Moriya Huzita
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Paulo César Masiero
Universidade de São Paulo – ICMC/USP

Dedico este trabalho

À minha mãe, ***Helena Barros de Oliveira***, fonte de toda minha inspiração, pela pessoa humilde, dedicada, honesta e trabalhadora que sempre foi.

AGRADECIMENTOS

Agradeço à minha mãe, **Helena Barros de Oliveira**, fonte de toda a minha inspiração, ao meu pai, **Edson Alves de Oliveira**, pelo amor e paciência demonstrados, minhas irmãs, **Eliane Marcuzo** e **Rosane Oliveira**, pelo amor, carinho e constante incentivo, e ao meu cunhado, **Itamar Marcuzo**, pelo carinho e incentivo.

Agradeço à minha orientadora **Itana Maria de Souza Gimenes**, que mais uma vez depositou sua confiança em meu trabalho, pelos seus conselhos profissionais e pelo conhecimento compartilhado comigo.

Agradeço a todos os professores que, de forma direta ou indireta, contribuíram com este trabalho, dentre eles: **Elisa Hatsue M. Huzita**, **Antonio Mendes**, **Ricardo R. Ciferri**, **Cristina A. D. Ciferri**, **Maria Madalena Dias**, **Nardênio A. Martins**, **Ronaldo A. L. Gonçalves** e **João Ângelo Martini**.

Agradeço à **Josiane Fujisawa Filus** pelo carinho, respeito, compreensão e amizade demonstrados.

Agradeço aos amigos **Fabrício Ricardo Lazilha**, **Radames Juliano Halmeman**, **Ruy Tsutomu Nishimura, Fábio Zaupa**, e **Fernanda F. Oliveira** pela amizade, respeito e contribuições.

Agradeço aos amigos **Rogério Santos Pozza**, **Igor Wiese** e **Luiz Arthur** pela forma como demonstraram e continuam demonstrando amizade e respeito.

Agradeço aos amigos da **Sérgio Yamada Computação** pelo respeito e extremo profissionalismo. Agradecimento especial ao senhor **Sérgio Yamada** pela amizade, confiança e atitude.

Agradeço aos funcionários do Departamento de Informática **Jayme F. Junior**, **Paulo C. Gonçalves**, **Marcelo R. Donato**, e, em especial à **Maria Inês Davanço**, pela amizade e paciência.

Agradeço à **Rejane Sartori** (PPG) pela amizade cultivada há vários anos.

Agradeço especialmente à **Fundação Araucária** e à **Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)**, pelo apoio financeiro despendido a este trabalho.

Agradeço aos senhores **Bruce Dickinson**, **James Hetfield**, **Humberto Gessinger** e **Herbert Viana** pelo incentivo e inspiração em suas letras e melodias.

Agradeço, finalmente, a **James Gosling**, por criar e acreditar na tecnologia Java e ao ***Java Community Process*** **(JCP)** e à ***Sun Microsystems*** por não permitirem que Java se torne um bem proprietário.

“Quanto mais aumenta nosso conhecimento, mais evidente fica nossa ignorância.”

John Fitzgerald Kennedy

RESUMO

A abordagem de linha de produto de software (LP) tem como objetivo promover a geração de produtos específicos por meio da reutilização de uma infra-estrutura central estabelecida para um determinado domínio. A abordagem de LP é adequada aos domínios em que existe demanda por produtos que possuem características comuns e pontos de variação bem definidos. Um dos grandes desafios da abordagem de LP é o gerenciamento de variabilidade que visa administrar as diferenças entre seus produtos. A correta captura e representação de variabilidades permitem ampliar o número de possíveis produtos desenvolvidos a partir de uma LP. Várias soluções para o gerenciamento de variabilidades são encontradas na literatura, porém o que se percebe é a carência de um processo que permita identificar, representar, delimitar, escolher mecanismos de implementação, monitorar e rastrear as variabilidades de uma LP. Assim, a principal contribuição deste trabalho é a especificação de um processo de gerenciamento de variabilidade para LP. O processo proposto é avaliado seguindo os conceitos de engenharia de software experimental na forma de um estudo de caso que toma como domínio de aplicação os Sistemas de Gerenciamento de *Workflow* (WfMS) e uma LP existente para WfMS. O estudo de caso mostrou que a introdução do processo proposto no desenvolvimento de LP existente permite identificar e representar um número maior de variabilidades. Os resultados do estudo de caso são apresentados graficamente na forma de histogramas.

Palavras-chaves: Engenharia de Software Experimental. Estudo de Caso. Linha de Produto de Software. Ponto de Variação. Variabilidade. Variante.

ABSTRACT

The software product line approach (PL) promotes the generation of specific products from a set of core assets for a given domain. This approach is applicable to domains in which products have well-defined commonalities and variation points. Variability management is concerned with the management of the differences between PL products. It currently represents one of the challenges of the PL approach. The correct capture and representation of variabilities increase the number possible products that can be developed from a PL. Although several partial solutions for variability management can be found in the literature, there is still a lack of an explicit process that allows: identification, representation, delimitation, selection of implementation mechanisms, monitoring and variability tracing in a PL. Thus, the main contribution of this work is the specification of a process for variability management. The proposed process was evaluated following the experimental software engineering basis. A case study was developed for the workflow management system domain (WfMS) and an existing PL for WfMS. The results of the case study have shown that the proposed process has made explicit a higher number of variabilities as compared to the existing PL process. The results of the case study are shown graphically as histograms.

Keywords: Case Study. Experimental Software Engineering. Software Product Line. Variability. Variant. Variation Point.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

ADL	Architecture Description Language
CASE	Computer-Aided Software Engineering
COTS	Commercial-Off-The-Shelf
ExPSEE	Experimental-Centered Software Engineering Environment
FAST	Feature-Oriented Abstract, Specification and Translation
FODA	Feature-Oriented Domain Analysis
FORM	Feature-Oriented Reuse Method
HTTP	Hyper-Text Transfer Protocol
IDE	Integrated Development Environment
KobrA	Komponentenbasierte Anwendungsentwicklung (Component-Based Application Development)
LP	Linha de Produto de Software
PLP	Product Line Practice
PLUS	Product Line UML-Based Software Engineering
PuLSE	Product Line Software Engineering
PV	Ponto de Variação
SEI	Software Engineering Institute
TCP	Transmission Control Protocol
USDP	Unified Software Development Process
WfMC	Workflow Management Coalition
WfMS	Workflow Management System

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

A abordagem de linha de produto de software (LP) tem como objetivo principal promover a geração de produtos específicos com base na reutilização de uma infra-estrutura central. Essa infra-estrutura, também conhecida como núcleo de artefatos, é formada, normalmente, por uma arquitetura de software e seus componentes. A abordagem de LP é adequada aos domínios em que existe uma demanda por produtos que possuem características comuns, mas que contém pontos de variação bem definidos. Atualmente, existem várias abordagens de LP (GIMENES; TRAVASSOS, 2002; CLEMENTS; NORTHROP, 2001).

Dentre os benefícios conseguidos com a abordagem de LP (HEYMANS; TRIGAUX, 2003; CLEMENTS; NORTHROP, 2001), pode-se citar: melhor compreensão do domínio em questão, maior reutilização de artefatos e diminuição do tempo para produtos chegarem ao mercado, o chamado *time-to-market*. Esses benefícios podem ser evidenciados por meio de relatos de empresas que se beneficiaram com a adoção da abordagem de LP (BASS; CLEMENTS; KAZMAN, 2003) como, por exemplo, a Nokia, que aumentou a produção de modelos diferentes de telefones celulares de 4 modelos/ano para 25-30 modelos/ano.

Grande parte da comunidade acadêmica e empresarial vem se esforçando para propor soluções que diminuam as dificuldades encontradas na adoção da abordagem de LP, tais como a reutilização e a evolução do núcleo de artefatos.

O núcleo de artefatos é uma das partes mais importantes de uma LP, pois é a partir dele que se consegue aumentar o número de produtos gerados pela LP. Para se conseguir aumentar esse número, desejável às organizações, deve-se manter os artefatos da LP os mais genéricos possíveis, adiando decisões de projeto (VAN GURP; BOSCH, 2001; SEI, 2004).

Tais decisões de projeto são tratadas como variabilidades. Porém, o que se percebe é uma grande dificuldade no gerenciamento de variabilidades e várias questões de pesquisa em aberto sobre este assunto.

Fica evidenciado, dessa forma, que existe uma carência de um processo explícito que mostre como as variabilidades devem ser identificadas, representadas, implementadas e gerenciadas durante todo o ciclo de vida de uma LP (VAN GURP; BOSCH, 2001).

Assim, este trabalho de mestrado propõe um processo que possibilita a identificação, representação, delimitação, escolha de mecanismos de implementação e monitoração das variabilidades em todo o ciclo de vida de uma LP. Com base no processo proposto, é possível rastrear variabilidades e analisar a configuração de produtos específicos de uma LP. O processo proposto utiliza como entrada artefatos comuns às abordagens de LP existentes.

Para a avaliação do processo proposto, foi realizado um estudo de caso nos moldes da Engenharia de Software Experimental (KITCHENHAM, 1996; BASILI; SELBY; HUTCHENS, 1986), uma vez que esta estratégia permite controlar as variáveis que estão sendo observadas com um custo médio. O estudo de caso tomou como base o domínio dos Sistemas de Gerenciamento de *Workflow* (WfMS - *Workflow Management System*) (WFMC, 2004). Esse domínio de aplicação vem sendo utilizado na realização de vários trabalhos de pesquisa, graduação e mestrado pelo grupo de engenharia de software da Universidade Estadual de Maringá. Dentre os principais trabalhos realizados, está o desenvolvimento de uma LP para WfMS. Essa LP possui uma documentação bem definida (OLIVIEIRA JUNIOR, 2002), uma arquitetura de LP (LAZILHA, 2002), componentes que povoam esta arquitetura (HALMEMAN, 2003) e um processo de geração de produtos específicos (NISHIMURA, 2004).

Esta dissertação está organizada da seguinte maneira: o capítulo 2 apresenta a fundamentação teórica necessária à compreensão e contextualização do trabalho desenvolvido; o capítulo 3 apresenta o processo de gerenciamento de variabilidade proposto; o capítulo 4 apresenta a avaliação do processo proposto com base em técnicas experimentais e o capítulo 5 apresenta as conclusões e trabalhos futuros.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos básicos sobre LP e variabilidade em LP. Apresenta também o processo de LP para WfMS que serviu de base para o desenvolvimento deste trabalho.

2.1 LINHA DE PRODUTO DE SOFTWARE

Uma LP representa um conjunto de sistemas que compartilham características comuns e gerenciáveis que satisfazem as necessidades de um segmento particular do mercado ou de uma missão (CLEMENTS; NORTHROP, 2001; NORTHROP, 2002). Esse conjunto de sistemas é também chamado de família de produtos. Os membros da família são produtos específicos desenvolvidos de maneira sistemática a partir de um conjunto comum de artefatos da LP.

Nos últimos anos, tem-se percebido uma crescente adoção da abordagem de LP por causa dos seus benefícios. Essa abordagem possibilita às organizações explorar as semelhanças entre seus produtos, aumentando, assim, a reutilização de artefatos e, como conseqüência, tem-se uma diminuição dos custos e do tempo no desenvolvimento (HEYMANS; TRIGAUX, 2003).

Segundo Clements e Northrop (2001) e Heymans e Trigaux (2003), os benefícios conseguidos com a adoção da abordagem de LP podem ser classificados em:

- **organizacionais:** melhor compreensão do domínio e aumento da qualidade dos produtos e da confiança do cliente;

- **engenharia de software:** melhor análise e aumento da reutilização dos requisitos e dos artefatos, melhor controle da qualidade dos produtos, estabelecimento de padrões e documentação reutilizável;
- **negócio:** redução dos gastos com teste e manutenção.

A adoção de uma abordagem de LP leva em consideração fatores organizacionais (CLEMENTS; NORTHROP, 2001; NORTHROP, 2002; SEI, 2004), a saber: a natureza dos produtos, o mercado ou missão, as metas de negócio, a estrutura organizacional, a cultura e as políticas organizacionais, as disciplinas de processo de software, a maturidade dos artefatos legados e a distribuição geográfica da equipe de trabalho.

A obtenção dos benefícios acima indicados implica em contrapartida das organizações. Um exemplo disso é a mudança da visão de desenvolvimento de produtos específicos para uma abordagem de LP que exige treinamento e adaptação de comportamento da equipe de desenvolvimento e gerenciamento.

O desenvolvimento seguindo uma abordagem de LP requer gerenciamento em longo prazo, pois os benefícios não são imediatamente visíveis. Assim, uma grande quantidade de produtos deve ser produzida para que a adoção desta abordagem possa ser realmente consolidada.

A seguir, são apresentadas as atividades essenciais e uma visão geral das abordagens existentes de LP.

2.1.1 Atividades Essenciais de Linha de Produto de Software

O SEI (*Software Engineering Institute*), através da iniciativa PLP (*Product Line Practice*), estabeleceu as atividades essenciais de uma abordagem de LP (SEI, 2004). Essas atividades são o **Desenvolvimento do Núcleo de Artefatos**, também conhecida como Engenharia de Domínio, o **Desenvolvimento do Produto,** também conhecida como

Engenharia de Aplicação**,** e o **Gerenciamento da Linha de Produto**. A Figura 1 ilustra as interações entre essas atividades.

Os três círculos da Figura 1 indicam que as atividades de uma LP são altamente interligadas e iterativas. As flechas rotativas indicam que, além dos artefatos gerados no desenvolvimento de produtos do núcleo, são também realizadas revisões dos artefatos ou até mesmo inclusão de novos artefatos.

Na medida em que os produtos são desenvolvidos, o núcleo de artefatos pode evoluir. Um exemplo disto ocorre quando se identifica um novo requisito relevante ao domínio que inicialmente não fazia parte da especificação de requisitos da LP. Assim, essa especificação evolui, incorporando o novo requisito identificado e o núcleo de artefatos é atualizado.

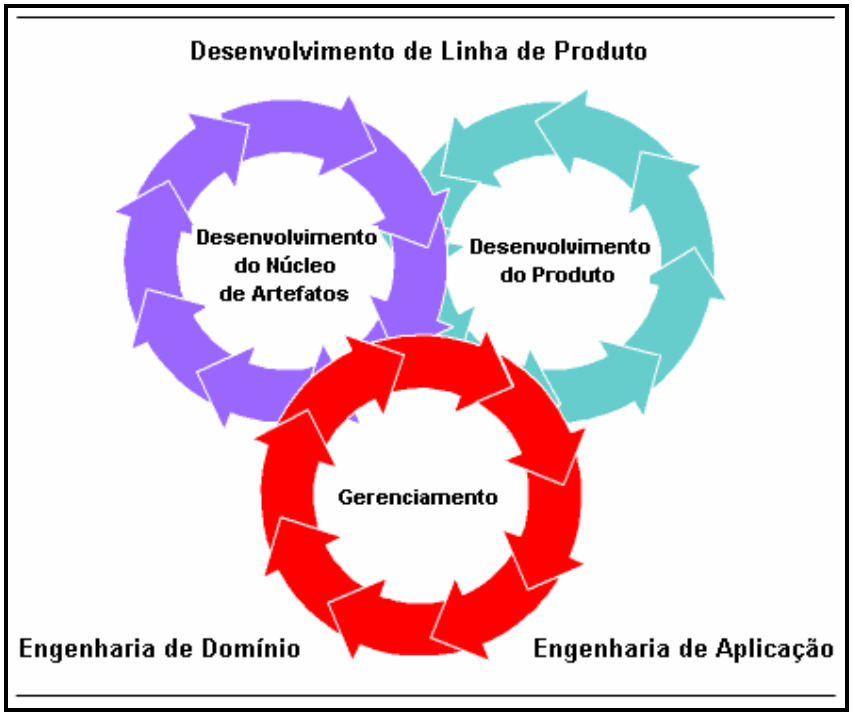

Figura 1 - Atividades essenciais de linha de produto de software
Fonte: SEI (2004).

A seguir, é apresentada uma breve descrição das atividades essenciais de LP.

2.1.1.1. Desenvolvimento do Núcleo de Artefatos

O objetivo da atividade de desenvolvimento do núcleo de artefatos é estabelecer uma infra-estrutura central que será reutilizada pelos produtos gerados a partir da LP.

Os artefatos de entrada para esta atividade são:

- as **restrições do produto:** as semelhanças e as variações entre produtos que constituirão a LP, as normas que devem ser seguidas para definir as características dos produtos e os limites de desempenho;
- os **estilos arquiteturais, padrões e *frameworks*:** utilizados para a construção da arquitetura de LP;
- as **restrições de produção:** os componentes COTS (*Commercial-Off-The-Shelf*) que serão utilizados, as normas que serão seguidas para a produção dos produtos e os componentes legados que serão reutilizados;
- a **estratégia de produção:** estratégia geral da abordagem para produzir os artefatos da LP;
- o **repositório dos artefatos pré-existentes:** os artefatos existentes como, por exemplo, componentes, especificações ou partes legadas do domínio catalogadas para a sua futura reutilização.

Esta atividade tem como principais artefatos de saída:

- o **contexto da linha de produto:** descrição dos produtos que constituirão a LP ou que esta é capaz de produzir. Esta descrição apresenta as semelhanças e as variações entre os produtos, além de incluir características e operações destes, tais como desempenho e atributos de qualidade;

- o **núcleo de artefatos da linha de produto:** base para a produção de produtos a partir da LP. Normalmente, o núcleo de artefatos inclui uma arquitetura a ser reutilizada pelos produtos, bem como os seus componentes;
- o **plano de produção dos produtos:** descrição das decisões a serem tomadas para instanciar produtos específicos a partir do núcleo de artefatos da LP.

A Figura 2 mostra os artefatos de entrada e saída da atividade de desenvolvimento do núcleo de artefatos.

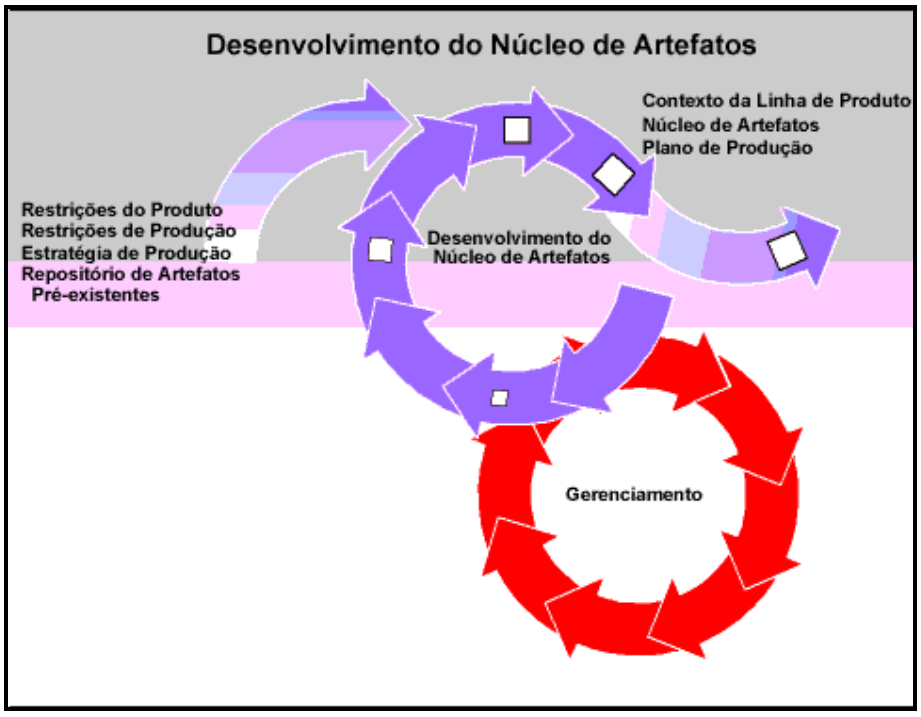

Figura 2 - Desenvolvimento do núcleo de artefatos
Fonte: SEI (2004).

2.1.1.2. Desenvolvimento do Produto

A atividade de desenvolvimento do produto tem como principal objetivo a geração de produtos de uma LP. Porém, é possível identificar requisitos que, inicialmente, não haviam sido especificados e, conseqüentemente, atualizar o núcleo de artefatos da LP.

Essa atividade depende substancialmente dos artefatos de saída da atividade anterior, que lhe servem como entrada. Esses artefatos são o **contexto da LP**, o **núcleo de artefatos** e o **plano de produção**. Além desses artefatos, são necessários também os requisitos para um produto específico.

As flechas rotativas indicam iteração e relacionamentos intrínsecos como, por exemplo, a existência e a disponibilidade de um produto específico podem afetar os requisitos de um produto subseqüente.

A Figura 3 mostra os artefatos de entrada e saída da atividade de desenvolvimento do produto.

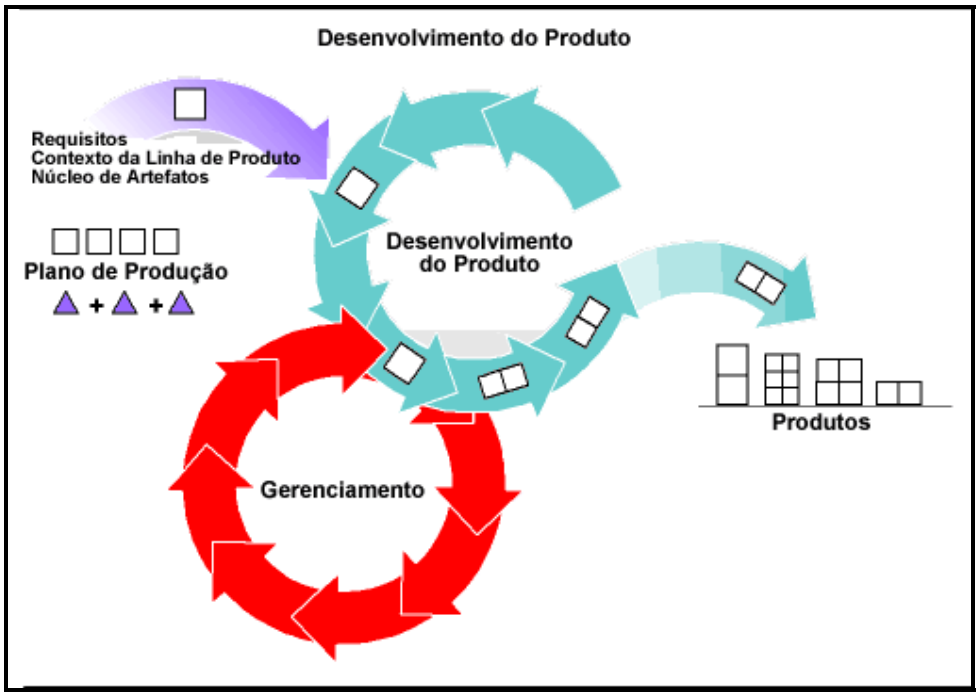

Figura 3 - Desenvolvimento do produto
Fonte: SEI (2004).

2.1.1.3. Gerenciamento da Linha de Produto

A atividade de gerenciamento de LP deve garantir que todas as atividades técnicas sejam realizadas de acordo com um planejamento coordenado.

O gerenciamento pode ser dividido em duas categorias (SEI, 2004):

- **gerenciamento técnico**: coordena as atividades de desenvolvimento do núcleo de artefatos e desenvolvimento do produto para garantir que as equipes de desenvolvimento sigam os processos definidos para a LP e coletem dados suficientes para acompanhar o progresso desta;
- **gerenciamento organizacional**: garante que as unidades organizacionais recebem os recursos corretos (ex. treinamento) em quantidades suficientes.

Uma das ações mais importantes da atividade de gerenciamento da LP é a criação de um plano de adoção que descreva o estado desejado da organização e uma estratégia para alcançar tal estado.

O gerenciamento de LP também deve estabelecer como as atividades de desenvolvimento do núcleo de artefatos e desenvolvimento do produto devem interagir para permitir a evolução da LP e o gerenciamento das semelhanças e das variabilidades de cada artefato da LP.

2.1.2 Abordagens de Linha de Produto de Software

A literatura existente apresenta algumas abordagens de LP. Dentre elas, pode-se citar algumas, como (GIMENES; TRAVASSOS, 2002): *Feature-Oriented Domain Analysis* (FODA) (KANG, 1990), Synthesis (SPC, 1993), *Family-Oriented Abstraction, Specification and Translation* (FAST) (WEISS; CHI TAU, 1999), *Product Line Software Engineering* (PuLSE) (BAYER, 1999), a abordagem proposta por Bosch (BOSCH, 2000), a iniciativa *Product Line Practice (PLP)* (CLEMENTS, NORTHROP, 2001), o método KobrA (ATKINSON et al., 2001), e mais recentemente, a abordagem *Product Line UML-Based Software Engineering* (PLUS) (GOMAA, 2005).

O FODA é um dos precursores da abordagem de LP. Ele foi desenvolvido no SEI como um método para análise de domínio. Descata-se pela introdução do modelo de *features*,

amplamente utilizado nas abordagens de LP. Em seguida, foi desenvolvida uma extensão dessa abordagem, chamada FORM (KANG et al., 1998) que inclui questões arquiteturais e de componentes.

As abordagens Synthesis e FAST tratam de questões abrangentes de LP. São abordagens precursoras que serviram de base para possibilitar a definição de um contexto mais geral para LP, como o definido na iniciativa PLP, porém não tiveram influência direta neste trabalho.

O método KobrA seguiu a abordagem Pulse caracterizando-se como uma abordagem de LP baseada em componentes. Essa abordagem é importante no contexto deste trabalho, pois o método de geração de produto da LP descrito na Seção 2.3.2 o utiliza como ponto de partida. O KobrA engloba várias tecnologias da engenharia de software como desenvolvimento baseado em componentes, *frameworks*, modelagem de processos e arquiteturas. Ele é composto por duas etapas: engenharia de *framework,* que é a representação estática de um conjunto de componentes, e engenharia de aplicação, que usa o *framework* para construir aplicações de domínio específico.

A abordagem Bosch considera uma LP em três dimensões, sendo elas: arquitetura, componente e sistema; negócios, organização, processo e tecnologia; e desenvolvimento, aplicação e evolução. Esta abordagem enfatiza a especificação de uma arquitetura de software formada por componentes reutilizáveis.

A iniciativa PLP foi tratada na seção anterior. Ela não contém um método de construção de LP, mas se destaca por caracterizar e uniformizar os vários conceitos de LP, bem como promover a sua utilização.

A abordagem PLUS (GOMAA, 2005) permite a sua integração com outros modelos de processo de software, dentre eles o Processo Unificado de Desenvolvimento de Software

(*Unified Software Development Process* - USDP). Cada fase do PLUS possui os mesmos nomes dos *workflows* do USDP. A primeira fase do PLUS, *inception*, envolve um estudo minucioso para determinar se a LP é viável ou não, o seu contexto, funcionalidades, grau de semelhanças e variabilidades e o número estimado de seus membros. Durante esta fase, são elaborados os casos de uso iniciais, um diagrama conceitual de classes e um modelo inicial de *features*. Na segunda etapa do PLUS, *elaboration*, o modelo de casos de uso e o modelo de *features* são revisados e elaborados em maiores detalhes, identificando os seus pontos de variação. Nesta fase, a arquitetura da LP é expandida e incluem-se os componentes opcionais e variantes. Na fase seguinte do PLUS, *construction*, os componentes são desenvolvidos e testados; após essa fase, inicia-se a *transition*, na qual os componentes são integrados e ficam disponíveis para os usuários poderem testar. As iterações a seguir permitem a integração de componentes opcionais e variantes. O método PLUS permite a identificação de componentes variantes através do uso de estereótipos para representar variabilidade. O processo de gerenciamento de variabilidade proposto neste trabalho de mestrado também permite a identificação de tais componentes.

2.1.3 Considerações Finais

Apesar das abordagens existentes, ainda existe uma carência por métodos de gerenciamento de LP em todos os seus aspectos. Tais aspectos vão desde o monitoramento do núcleo de artefatos, passando pela evolução deste até chegar ao gerenciamento dos artefatos variantes.

Quando se considera o gerenciamento de variabilidade em LP, percebe-se que este ainda é um assunto amplamente discutido, porém poucas são as contribuições da literatura que convergem ao ponto-chave do problema, que é a proposta de um processo eficaz e independente da abordagem de LP adotada para gerenciar variabilidades.

A seção seguinte apresenta uma revisão bibliográfica a respeito dos conceitos básicos sobre variabilidade em LP, imprescindível à compreensão deste trabalho.

2.2 VARIABILIDADE EM LINHA DE PRODUTO DE SOFTWARE

Dentre os principais aspectos que devem ser gerenciados em uma LP, estão as diferenças entre seus produtos, conhecidas como variabilidades. Variabilidade é um dos assuntos relacionados à LP que mais vem chamando a atenção tanto da comunidade acadêmica quanto da empresarial devido à complexidade do seu gerenciamento (HEYMANS; TRIGAUX, 2003). Esse fato pode ser evidenciado, por exemplo, com a existência de um *workshop* internacional específico sobre o assunto (*Workshop on Software Variability Management*) em uma das conferências internacionais mais importantes sobre LP, a *Software Product Line Conference* (SPLC, 2005).

Esta seção apresenta os conceitos básicos de variabilidade, bem como as formas de identificar, representar, implementar e gerenciar as variabilidades em LP, além de algumas ferramentas de apoio ao gerenciamento de variabilidade.

2.2.1 Definições de Variabilidade

Segundo Weiss e Chi Tau (1999), variabilidade é a forma como os membros de uma família de produtos podem se diferenciar entre si.

A variabilidade é descrita por pontos de variação e variantes. Um ponto de variação é um local específico de um artefato em que uma decisão de projeto ainda não foi resolvida. A cada ponto de variação está associado um conjunto de variantes. Cada variante corresponde a uma alternativa de projeto para instanciar uma determinada variabilidade. A resolução de um ponto de variação se dá através da escolha de uma ou mais variantes do conjunto de variantes relacionado (HEYMANS; TRIGAUX, 2003).

O conceito de variabilidade provém do desenvolvimento e instanciação de *frameworks*. Um *framework* é o esqueleto de uma aplicação formado por um conjunto de classes que contém um projeto abstrato de soluções para uma família de problemas relacionados (PARSONS; RASHID; SPECK, 1999). Um *framework* possui (PREE, 1995):

- ***frozen spots*** (pontos fixos): definem a arquitetura geral de um sistema de software - componentes básicos e seus relacionamentos. O conjunto de *frozen spots* caracteriza os chamados *frameworks blackbox*;
- ***hot spots*** (pontos variáveis): representam as partes do *framework* que são específicas de sistemas individuais. O conjunto de *hot spots* caracteriza os chamados *frameworks whitebox.*

Segundo Parsons, Rashid e Speck (1999), os *frameworks* são instanciados através dos *hot spots*. Eles podem estar localizados em interfaces ou classes abstratas. Assim, os *hot spots* representam os pontos de variação em um *framework* e podem ser adaptados às necessidades de uma aplicação (BUSCHMANN et al., 1996).

A representação explícita de variabilidade torna possível a geração de produtos específicos de uma LP. A variabilidade surge do adiamento de certas decisões fundamentais ao projeto de produtos de software. Essas decisões, quando tomadas no início do projeto, restringem o domínio no qual o sistema pode ser aplicado.

As decisões tomadas em fases iniciais do projeto de um sistema computacional correspondem à abordagem tradicional de desenvolvimento de um único produto e não à abordagem de LP. Assim, quanto maior o número de decisões de projeto adiadas, maior será o número de variabilidades de um produto de software (HALMANS; POHL, 2003).

Segundo Van Gurp e Bosch (2001), as variabilidades podem ser inicialmente identificadas por meio do conceito de *feature*[1]. Este conceito teve origem na engenharia de domínio. *Feature* pode ser definida como uma característica de um sistema que é relevante e visível para o usuário final (KANG, 1990; SIMONS et al., 1996). As *features* podem ser funcionais e não-funcionais.

Van Gurp e Bosch (2001) caracterizam *feature* como sendo uma unidade lógica de comportamento que corresponde a um conjunto de requisitos funcionais e de qualidade. Segundo Van Gurp (2000) existe uma relação n:m entre *features* e requisitos. Isto significa que um determinado requisito pode ser aplicado a várias *features* e uma *feature* em particular pode abranger mais de um requisito. Segundo Griss, Favaro e D'Alessandro (1998) e Van Gurp e Bosch (2001), as *features* podem ser classificadas como sendo:

- **obrigatórias:** estão sempre presentes e identificam um produto. Por exemplo, enviar e receber e-mail em um sistema de correio eletrônico;
- **opcionais:** podem ou não estar presentes em um produto. Quando presentes, adicionam algum valor às *features* obrigatórias de um produto. Por exemplo, a possibilidade de se adicionar assinatura ao e-mail;
- **variáveis:** possuem um conjunto de *features* relacionadas em que zero ou mais dessas *features* podem ser selecionadas para estar presentes em um produto. Por exemplo, as mensagens de um correio eletrônico podem ser enviadas na forma textual, na forma de áudio ou, ainda, de ambas as formas;

[1] Neste trabalho, o termo *feature* não é traduzido por entendermos que o correspondente, na língua portuguesa característica, não traduz o quê tecnicamente o termo significa.

- **externas:** são as *features* oferecidas pela plataforma-alvo do sistema. Por exemplo, um cliente de e-mail com capacidade de fazer conexões TCP (*Transmission Control Protocol*).

Anastasopoulos (2001) apresenta, além das *features* obrigatórias e opcionais, mais dois tipos de *features*, que são:

- **mutuamente inclusivas**: para que uma *feature* seja incluída, outras *features* específicas devem ser também incluídas e vice-versa;
- **mutuamente exclusivas:** para que uma *feature* seja incluída, outras *features* específicas não devem ser incluídas e vice-versa.

As *features* podem ser estruturadas em modelos de *features,* que são representações em árvores. Atualmente, existem várias propostas de representações de *features* na literatura, porém todas tomam como base a representação em árvore (SOCHOS; PHILIPPOW; RIEBISCH, 2004). A Figura 4 apresenta um exemplo de modelo de *features* para um cliente de e-mail. Nessa figura, pode-se observar o relacionamento entre as *features*, além do seu tempo de resolução (*runtime* e *compiletime*).

As *features* se tornam complexas quando dependem umas das outras. Van Gurp e Bosch (2000) apresentam o conceito de ***feature interaction*** para caracterizar que *features* não podem ser consideradas de forma isolada devido às possiveis interdependências. A Figura 5 apresenta o conceito de *feature interaction* em diferentes artefatos. Nessa figura, pode-se observar que uma *feature* em um artefato de mais alto nível de abstração (ex. código fonte) não pode ser considerada de forma isolada sem antes analisar as *features* dependentes em um artefato de mais baixo nível de abstração (ex. código executável).

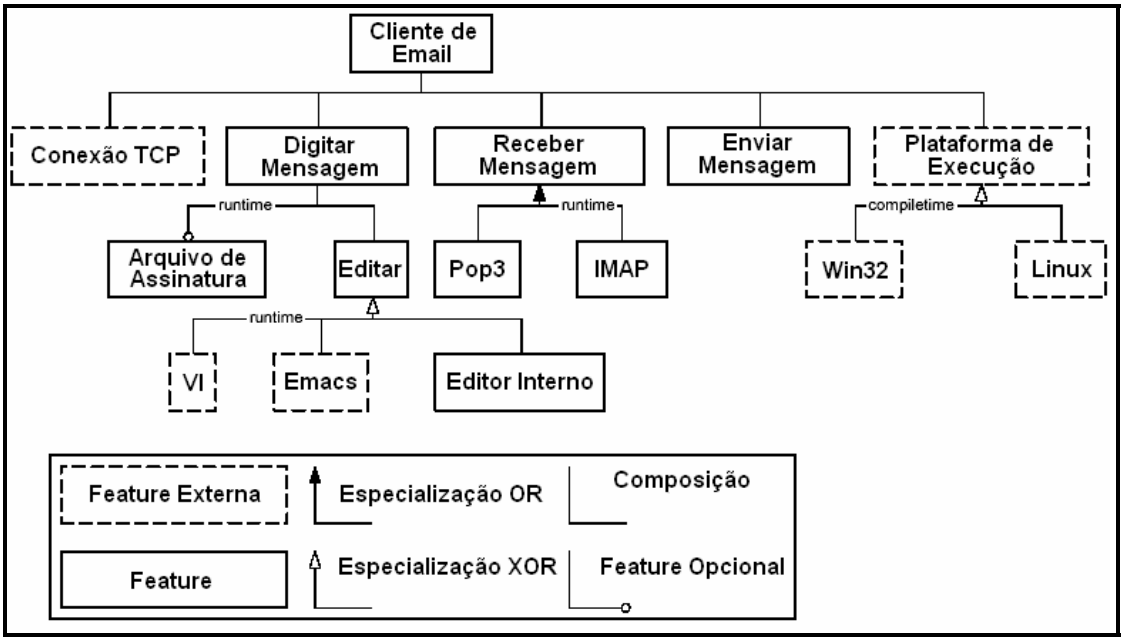

Figura 4 - Exemplo de modelo de *features*
Fonte: Van Gurp e Bosch (2001, p. 5).

As variabilidades de um domínio podem ser representadas pelo modelo de *features*. Segundo Van Gurp e Bosch (2000), as variabilidades podem estar associadas a diferentes níveis de abstração, sendo eles a descrição da arquitetura, a documentação de projeto, o código fonte, o código compilado, o código ligado e o código executável. Esses diferentes níveis de abstração podem estar associados a diferentes estágios do desenvolvimento de software. A Figura 6 apresenta o processo de representação e transformação no desenvolvimento de software. As variabilidades podem ser associadas às representações que correspondem aos artefatos de uma LP.

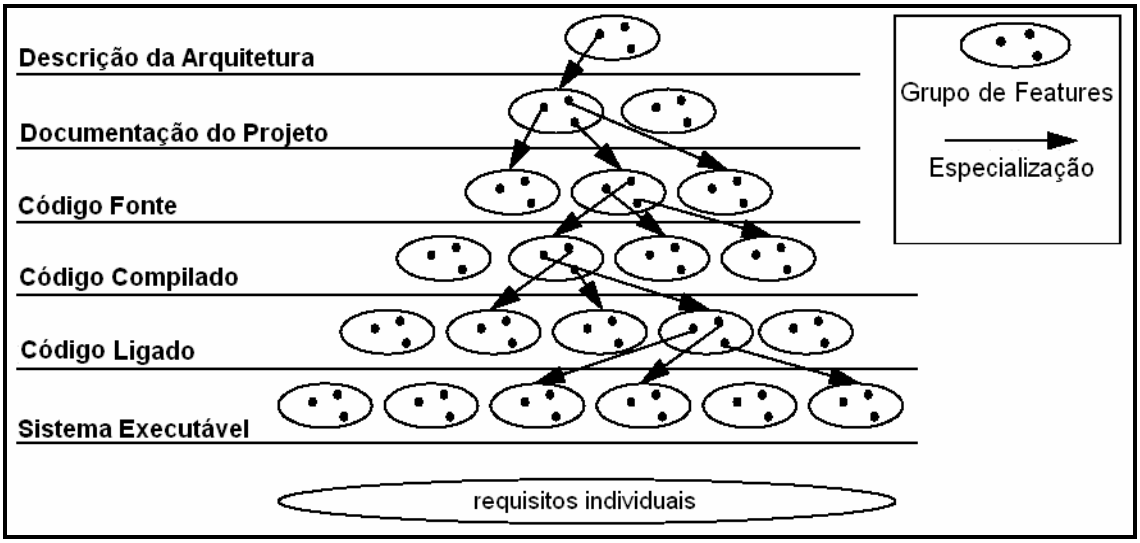

Figura 5 - *Feature interaction* em diferentes artefatos
Fonte: Van Gurp e Bosch (2000, p. 7).

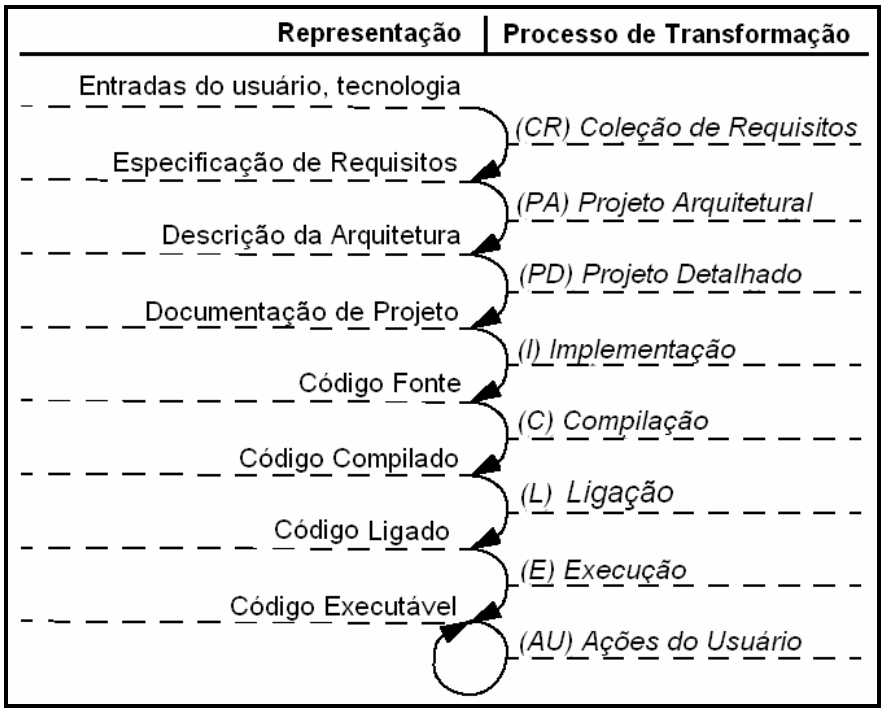

Figura 6 - Representação e transformação no desenvolvimento
Fonte: Van Gurp e Bosch (2001, p. 6).

2.2.2 Classificação de Variabilidade

As variabilidades em uma LP podem ser classificadas em relação a diferentes aspectos, sendo eles:

- **estado das decisões de projeto** (VAN GURP; BOSCH, 2001);
- **tempo e espaço** (HALMANS; POHL, 2003);
- **ponto de vista do cliente e do gerente da LP** (HALMANS; POHL, 2003);
- **adição e remoção de funcionalidades** (ANASTASOPOULOS, 2001);
- **local de ocorrência** (BACHMANN; BASS, 2001);
- **níveis** (SVAHNBERG; BOSCH, 2000).

Van Gurp e Bosch (2001) classificam as variabilidades de acordo com o estado dos seus pontos de variação, conforme segue:

- **implícito:** em etapas iniciais do desenvolvimento de uma LP, existem muitas decisões de projeto indefinidas e, conseqüentemente, existem muitas variabilidades. Contudo ainda não é possível saber se tais decisões de projeto serão adiadas ou não;
- **projetado:** quando uma decisão de projeto é adiada, o ponto de variação passa a ser projetado. Assim, é possível pensar como e quando as variantes serão selecionadas. Desse modo, pode-se fazer uma distinção entre:
  - **ponto de variação aberto:** novas variantes podem ser adicionadas;
  - **ponto de variação fechado:** novas variantes não podem ser adicionadas.
- **definido:** quando uma ou mais variantes são escolhidas para resolver um ponto de variação. A diferença de um ponto de variação definido para um ponto de variação fechado é que o primeiro não permite a adição de variantes em seu conjunto de variantes relacionado, mas não quer dizer obrigatoriamente que as variantes já tenham sido escolhidas para resolver o ponto de variação. No segundo, não se permite a adição de nenhuma nova variante ao conjunto

relacionado e todas as variantes que resolverão o ponto de variação já foram escolhidas.

Segundo Halmans e Pohl (2003), as variabilidades podem ocorrer em tempo e espaço. A **variação em tempo** trata as diferentes versões de um produto, enquanto a **variação em espaço** significa que, ao mesmo tempo, dois produtos podem conter variantes que possuem diferentes versões de implementação. Eles argumentam que as variabilidades podem ser classificadas em essenciais e técnicas, dependendo do ponto de vista de quem as trata. As **variabilidades essenciais** definem "o que implementar" e correspondem ao ponto de vista do cliente, que trata os aspectos relacionados aos requisitos funcionais e não-funcionais. Já as **variabilidades técnicas** definem "como implementar" e correspondem ao ponto de vista do gerente de LP, o qual trata os pontos de variação, as variantes e o tempo de resolução das variabilidades.

Segundo Anastasopoulos (2001), as variabilidades podem ser de vários tipos:

- **positivas:** quando adicionam funcionalidades;
- **negativas:** quando removem funcionalidades;
- **opcionais:** quando um código é adicionado;
- **alternativas:** quando um código é substituído;
- **funcionais:** quando a funcionalidade muda;
- **plataforma/ambiente:** quando a plataforma ou ambiente de execução mudam.

Bachmann e Bass (2001) classificam as variabilidades segundo o local onde elas podem ocorrer, conforme segue:

- **variação em funções**: uma função pode estar presente em um produto, mas não em outro;

- **variação em dados**: uma estrutura de dados particular pode variar de produto para produto. Muitas vezes a variação em dados é uma conseqüência da variação em função;
- **variação em fluxo de controle**: um padrão de interação pode variar entre produtos. Por exemplo, um componente envia mensagens para outro de forma seqüencial, enquanto um outro produto poderia utilizar um terceiro componente para intermediar a interação na troca de mensagens, exigindo a identificação de qualquer tipo de erro ou exceção e recuperação destes;
- **variação em tecnologia**: a plataforma (por exemplo, sistema operacional, interface gráfica com o usuário e *middleware*) pode variar da mesma forma como as funções;
- **variação em metas de qualidade**: as metas de qualidade para um produto podem variar. Por exemplo, o acoplamento entre um produtor e um consumidor de um dado pode ser conseguido via mecanismo de assinatura ou via conexão permanente;
- **variação no ambiente**: a maneira como o produto interage com o ambiente pode variar. Por exemplo, a invocação do mecanismo de *middleware* pode ser realizada por C++ ou Java.

Svahnberg e Bosch (2000) classificam as variabilidades em níveis de abstração, que são:

- **linha de produto:** como os produtos se diferenciam, indicando quais componentes serão usados para cada produto;
- **produto:** preocupa-se com a arquitetura e a escolha de componentes para um produto específico;

- **componente:** artefato em que as implementações disponíveis são selecionadas;
- **sub-componente:** preocupa-se com a remoção de *features* desnecessárias;
- **código:** onde as variabilidades entre produtos serão implementadas, seguindo as interfaces definidas no nível anterior.

2.2.3 Tempo de Resolução de Variabilidade

Um fator importante no gerenciamento de variabilidade é o seu tempo de resolução. Ele indica em qual momento uma ou mais variantes serão associadas a um determinado ponto de variação (VAN GURP; BOSCH, 2000). Como exemplo, pode-se considerar uma classe responsável por encapsular a lógica de ordenação de um vetor de elementos (CLAUß, 2001a). Os pontos de variação dessa classe podem ser o tipo de algoritmo usado, sendo as possíveis variantes *bubble_sort, quick_sort* ou *insert_sort* e o tipo de elemento que será ordenado, sendo as possíveis variantes *IntegerSort* ou *StringSort*, como mostra a Figura 7. Assim, o tempo de resolução do ponto de variação poderia ser, por exemplo, tempo de execução, o que permitiria ao usuário escolher, com a aplicação em execução, o algoritmo de ordenação a ser usado e o tipo de elemento a ser ordenado.

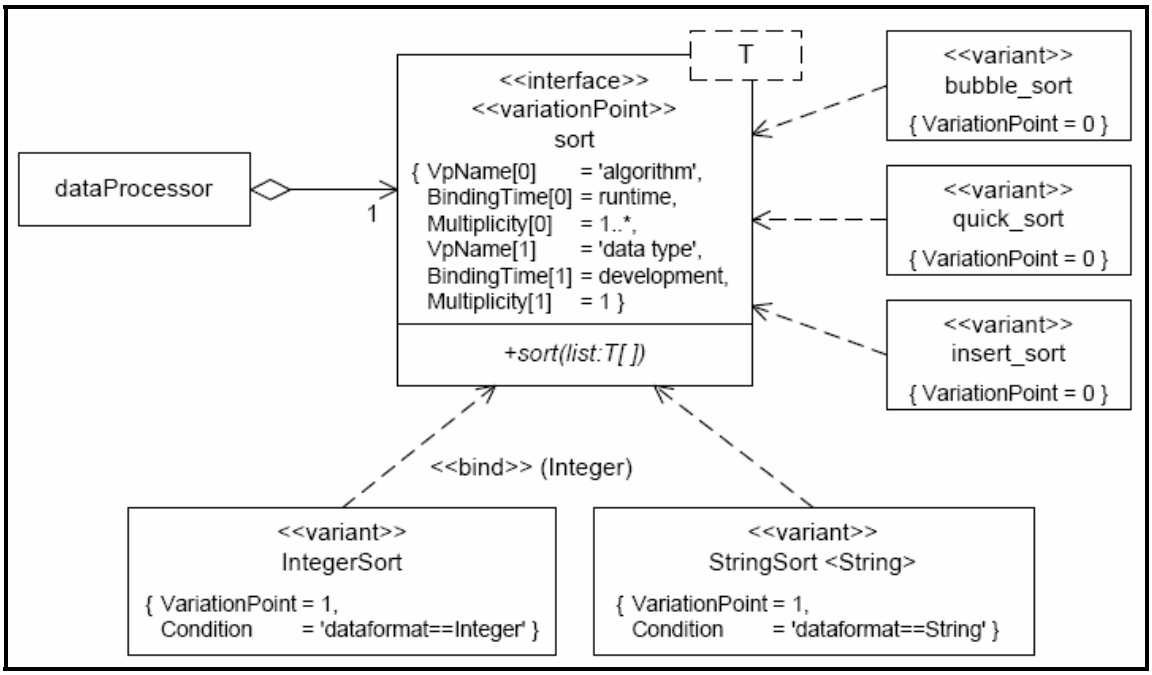

Figura 7 - Exemplo de ponto de variação e variantes
Fonte: Clauß (2001a, p. 5)

Segundo Anastasopoulos (2001), o tempo de resolução de variabilidade pode ser classificado como:

- **tempo de compilação:** a variabilidade é resolvida antes do programa ser compilado ou durante a compilação, usando diretivas de pré-processamento, por exemplo;
- **tempo de ligação:** a variabilidade é resolvida durante a ligação do módulo ou da biblioteca, selecionando diferentes bibliotecas com diferentes versões das operações exportadas;
- **tempo de execução:** a variabilidade é resolvida durante a execução do programa;
- **tempo de atualização:** a variabilidade é resolvida durante a atualização de programas ou após o início da execução destes (por exemplo, um programa de

atualização que adiciona funcionalidades como o *Windows Update* da Microsoft).

Fritsch, Lehn e Strohm (2002), assim como Anastasopoulos (2001), classificam os tempos de resolução de variabilidade como compilação e tempo de execução, além de:

- **programação:** pode ser especializado em:
  - **desenvolvimento do núcleo de artefatos**: durante a construção dos artefatos da LP;
  - **desenvolvimento do produto**: durante a instanciação da arquitetura de LP e de seus componentes.
- **integração:** pode ser especializado em:
  - **compilação;**
  - **seleção do código fonte.**
- **montagem**.

Segundo Fritsch, Lehn e Strohm (2002), o tempo de resolução de variabilidade restringe a escolha de mecanismos de implementação de variabilidade. Por exemplo, se uma variabilidade é resolvida em tempo de execução, não se pode implementá-la com um mecanismo que é definido em tempo de compilação.

2.2.4 Representação de Variabilidade

Dentre as principais soluções para representar variabilidade, estão as propostas por Jacobson, Griss e Jonsson (1997) para diagramas de casos de uso; Morisio, Travassos e Stark (2000) para diagramas de classes; Clauß (2001b) para modelos de *features* e diagramas de classes e Gomaa e Webber (2004) para artefatos da UML.

Jacobson, Griss e Jonsson (1997) propõem o uso de uma marca ("•") para representar variabilidade em casos de uso. Porém, variabilidades em atores não são tratadas. A Figura 8

apresenta um exemplo de representação de variabilidade em diagramas de casos de uso proposta por Jacobson, Griss e Jonsson em que o caso de uso “Agendar Aula” representa um ponto de variação e possui duas possíveis variantes, representadas pelos casos de uso “Agendar Aula Teórica” e “Agendar Aula Prática”. Em diagramas de casos de uso, a variabilidade é representada com o auxílio do estereótipo <<extend>>, além da marca “•”.

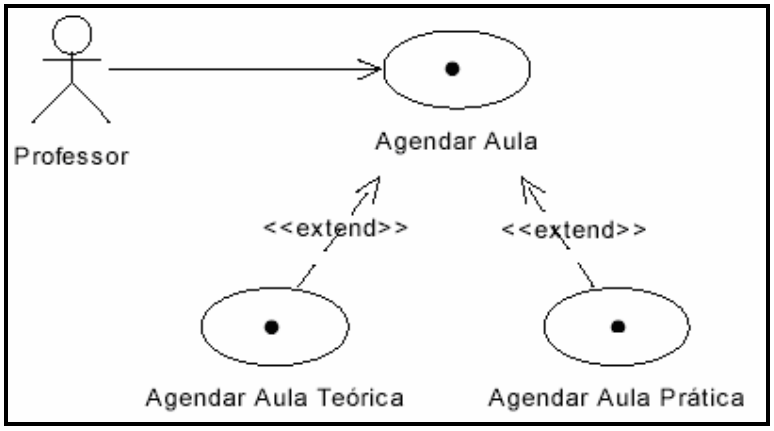

Figura 8 - Exemplo de representação de variabilidade proposta por Jacobson, Griss e Jonsson
Fonte: Lazilha (2002, p. 66).

Morisio, Travassos e Stark (2000) propõem a utilização dos estereótipos <<V>> e <<xorV>> para representar variabilidade em diagramas de classes. O estereótipo <<V>> também é utilizado para representar variabilidade em operações de uma determinada classe. A Figura 9 apresenta um exemplo da representação proposta por Morisio, Travassos e Stark (2000), em que a classe “Recurso” representa um ponto de variação e possui duas possíveis variantes, representadas pelas classes “Assistente” e “AudioVisual”. Em diagramas de classes, a variabilidade é representada utilizando-se generalização/especialização, além dos esterótipos <<V>> e <<xorV>>.

Clauß (2001b) apresenta três extensões da UML para representar variabilidades, sendo uma para modelos de *features* e duas para modelar variabilidade em projeto de software. A primeira delas se preocupa em representar os pontos de variação na forma de modelos de

features, porém usando diagramas de classes da UML. A segunda extensão representa, de forma explícita, os pontos de variação na análise e no projeto de software. A terceira se preocupa em representar elementos opcionais em situações em que não se aplicam pontos de variação como, por exemplo, em operações, atributos, associações, entre outros.

Gomaa e Webber (2004) apresentam quatro formas de representar variabilidade em LP, sendo elas a parametrização, a ocultação de informação, a herança e os pontos de variação. Essas formas podem ser representadas utilizando-se o modelo *Variation Point Model*, proposto por Gomaa e Webber (2004).

Um estudo comparativo entre as principais formas de representação de variabilidade propostas para modelos de *features*, diagramas de casos de uso e diagramas de classes é apresentado por Trigaux e Heymans (2003).

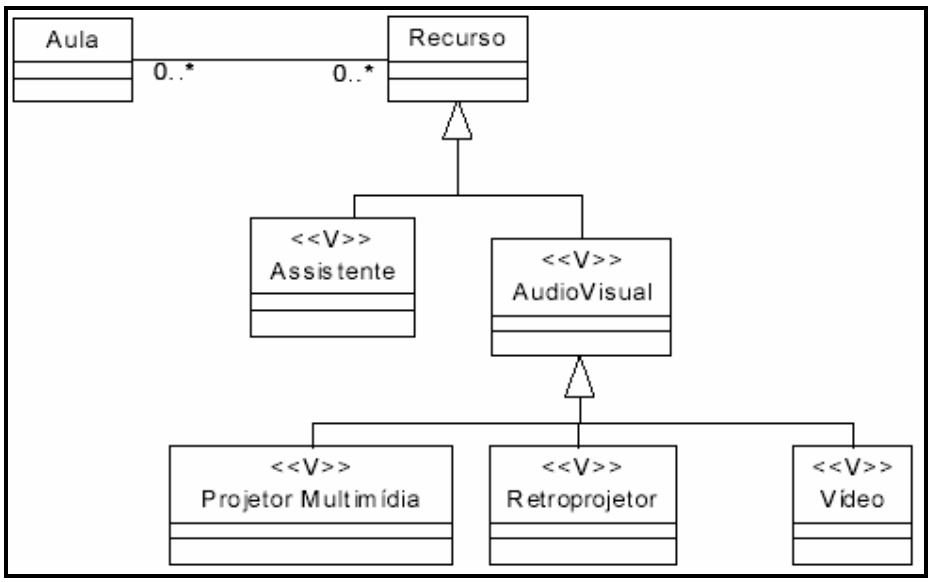

Figura 9 - Exemplo de representação de variabilidade proposta por Morisio, Travassos e Stark
Fonte: Lazilha (2002, p. 66).

2.2.5 Implementação de Variabilidade

Anastasopoulos (2001) baseia-se em técnicas de implementação de variabilidade, propostas por Jacobson, Griss e Jonsson (1997). Segundo o autor, a implementação das

variabilidades depende muito da escolha de uma linguagem de programação e de uma documentação muito bem definida dos artefatos da LP. Dentre as técnicas de implementação de variabilidade propostas por Anastasopoulos (2001), pode-se citar:

- **Agregação/Delegação:** permite que objetos deleguem funcionalidades. A variabilidade pode ser tratada colocando-se a funcionalidade obrigatória no objeto que delega e a variante no objeto delegado. Esta técnica é adequada para *features* opcionais, mas não é satisfatória para *features* alternativas. Um problema conhecido é quando o número de variantes cresce, pois o número de objetos delegados também cresce. Isto piora quando delegações já existentes são combinadas;
- **Herança:** adiciona funções básicas às superclasses e funções especializadas às subclasses;
- **Parametrização:** representa software reutilizável como uma biblioteca de componentes parametrizados. O comportamento do componente é determinado pelos valores dos parâmetros definidos. Isto evita replicação de código por centralizar decisões de projeto sobre um conjunto de variáveis. Por exemplo, uma pilha, na qual o tipo dos elementos é definido por um parâmetro. A parametrização pode melhorar a reutilização em LP e também facilitar o rastreamento das decisões de projeto;
- **Sobrecarga:** esta técnica utiliza o mesmo nome de um elemento para operar de maneiras diferentes. Isto ocorre em tempo de execução;
- **Carga Dinâmica de Classe:** todas as classes são carregadas na memória assim que estas são necessárias. Essa técnica é interessante para LP, pois, desta

maneira, um produto pode pesquisar seu contexto e decidir, em tempo de execução, qual classe carregar;

- **Compilação Condicional:** possibilita o controle sobre os segmentos de código a serem incluídos ou excluídos da compilação de um programa. Diretivas marcam os locais de variação no código. Uma das maiores vantagens desta técnica é o encapsulamento de múltiplas implementações em um simples módulo. A implementação desejada é selecionada pela definição dos símbolos condicionais apropriados. A compilação condicional é conseguida antes do tempo de compilação;
- **Reflexão:** é a habilidade de um programa manipular, na forma de dados, algo que representa o estado de um programa durante sua própria execução. Essa técnica está relacionada fortemente à meta-programação, na qual objetos em altos níveis de abstração representam entidades como sistemas operacionais e linguagens de programação;
- **Padrões de Projeto:** muitos dos padrões de projeto podem variar e fornecer soluções para o gerenciamento de variabilidade.

Svahnberg, Van Gurp e Bosch (2002) apresentam um conjunto de técnicas de implementação de variabilidade. Dentre elas, podemos citar a Reorganização Arquitetural, o Componente Variante de Arquitetura, o Componente Opcional de Arquitetura, a Substituição Binária e a Especialização de Componentes Variantes.

2.2.6 Gerenciamento de Variabilidade

O gerenciamento de variabilidades é uma das etapas que compõem a atividade de gerenciamento de LP (Seção 2.1.1.3).

Segundo Fritsch, Lehn e Strohm (2002) e Becker (2003), o gerenciamento de variabilidade está relacionado a todas as atividades do processo de desenvolvimento de LP.

Van Gurp e Bosch (2001) sugerem um conjunto de atividades para o gerenciamento de variabilidade:

- **identificação das variabilidades:** identificar onde a variabilidade é necessária, utilizando-se modelos de *features* ou especificações de requisitos;
- **delimitação das variabilidades:** delimitar os pontos de variação, o que envolve atividades como definição do tempo de resolução para cada ponto de variação, decisão de como e quando as variabilidades serão adicionadas ao sistema e a escolha da representação para realizar um ponto de variação;
- **implementação das variabilidades:** escolha de uma técnica para implementar as variabilidades;
- **gerenciamento das variantes:** controle das variantes de pontos de variação dependendo do fato destes estarem abertos ou não.

O gerenciamento das variabilidades deve contar com um apoio automatizado (VAN DER HOEK, 2000; SUCCI; YIP; PEDRYCZ, 2001; VAN GURP; BOSCH, 2001; BACHMANN; BASS, 2001; BEUCHE; PAPAJEWSKI; SCHRÖDER-PREIKSCHAT, 2003; HEYMANS; TRIGAUX, 2003; HALMANS; POHL, 2003; SEI, 2004).

Existem algumas ferramentas que auxiliam no gerenciamento de variabilidades como ***Ménage*** (VAN DER HOEK, 2000), ***Holmes*** (SUCCI; YIP; PEDRYCZ, 2000) e ***pure:variants*** (BEUCHE; PAPAJEWSKI; SCHRÖDER-PREIKSCHAT, 2003).

Ménage baseia-se na linguagem de descrição de arquitetura (ADL) xADL 2.0 que permite armazenar e recuperar arquiteturas de LP (GARG et al., 2003). Além disso, a ferramenta possui um ambiente visual, sendo possível criar componentes, portas e conectores

de arquiteturas de LP. A ferramenta conta também com controle de configuração e versão. A Figura 10 apresenta o ambiente gráfico da ferramenta *Ménage*.

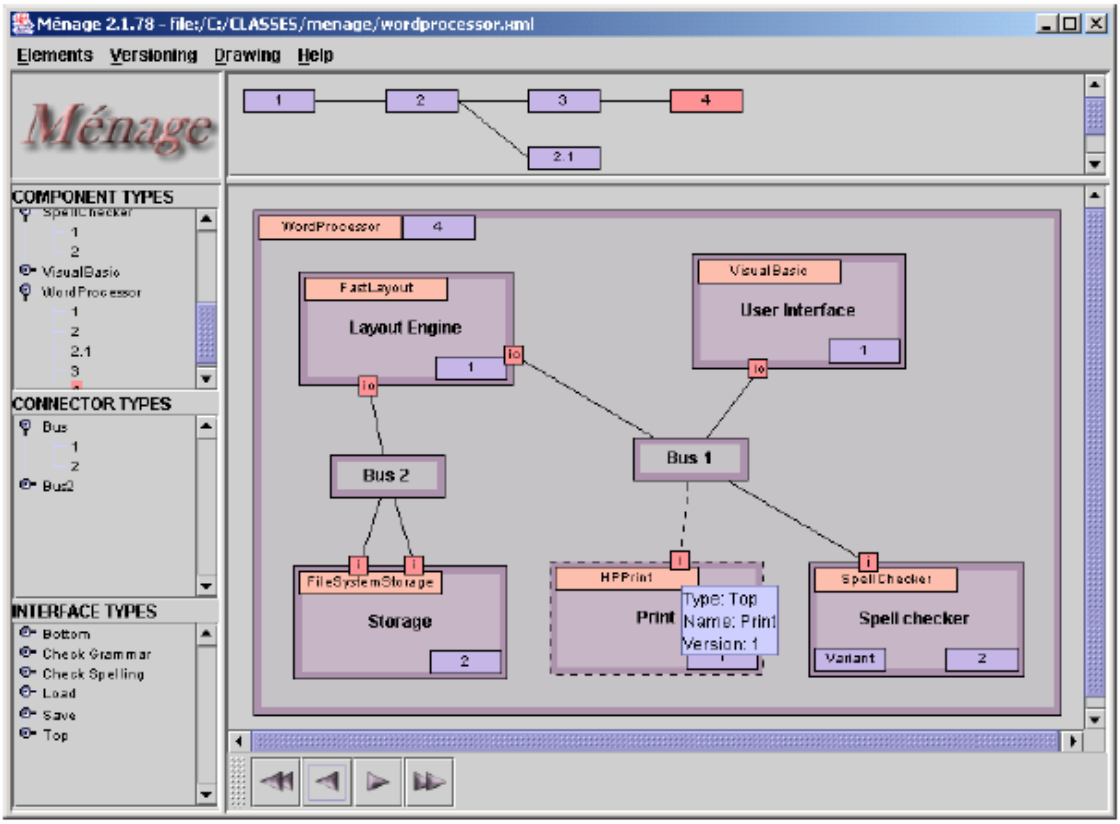

Figura 10 - Ambiente gráfico da ferramenta *Ménage*
Fonte: Garg et al. (2003, p. 5).

Holmes é um ambiente que integra várias ferramentas específicas às atividades de desenvolvimento de LP, incluindo definição do domínio, caracterização do domínio e definição do contexto da LP. Para a modelagem do domínio, é usada a ferramenta IBM/Rational Rose (IBM, 2004) e, para o desenvolvimento do *framework* do domínio, é realizada a integração com várias ferramentas CASE (*Computer-Aided Software Engineering*) de modelagem e IDEs (*Integrated Development Environment*) (SUCCI; YIP; PEDRYCZ, 2000).

pure:variants é uma ferramenta que permite a especificação de uma LP a partir de suas *features*. Assim, é possível criar famílias de sistemas e, em seguida, especificar configurações de produtos específicos da família. Isto auxilia na análise de impacto ao adicionar ou remover *features* a um produto específico da família especificada. A ferramenta está disponível na forma de *plug-in* para o IDE Eclipse (ECLIPSE, 2004). A Figura 11 apresenta o ambiente gráfico da ferramenta.

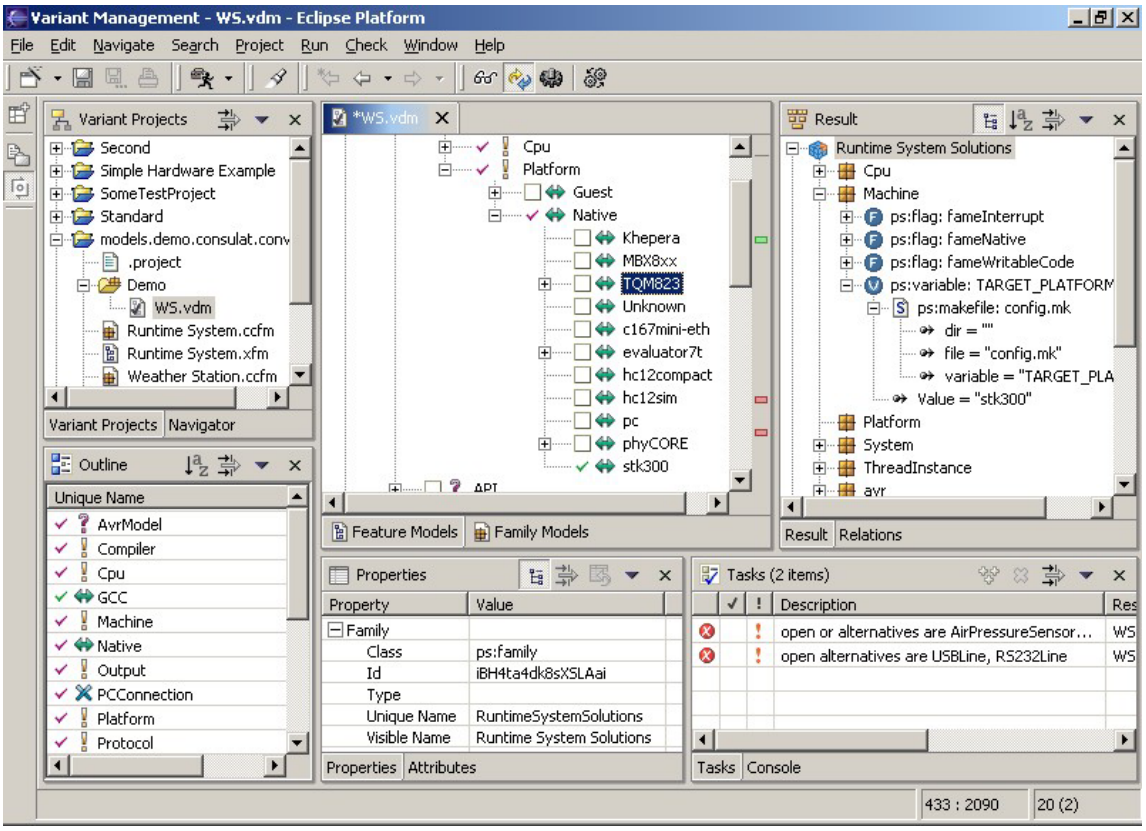

Figura 11 - Ambiente gráfico da ferramenta *pure-variants*
Fonte: Pure-Systems (2004).

Apesar da existência dessas ferramentas, percebe-se que elas não permitem gerenciar as variabilidades em todos os artefatos de uma LP, como proposto em SEI (2004).

Becker (2003) propõe um meta-modelo genérico de variabilidade que pode ser utilizado na construção de ferramentas de apoio ao gerenciamento de variabilidade. Esse

modelo permite o rastreamento das variabilidades pelos artefatos de uma LP. A Figura 12 apresenta esse modelo. Nesta figura, pode-se observar que o elemento "*Asset*", que representa um artefato de uma LP, está associado ao elemento "*VariationPoint*" e este está associado ao elemento "*Variability*". Dessa forma, é possível fazer o rastreamento das variabilidades de uma LP, pois cada artefato pode ser constituído de um ou mais pontos de variação e estes pontos são utilizados para descrever as variabilidades de artefatos da LP.

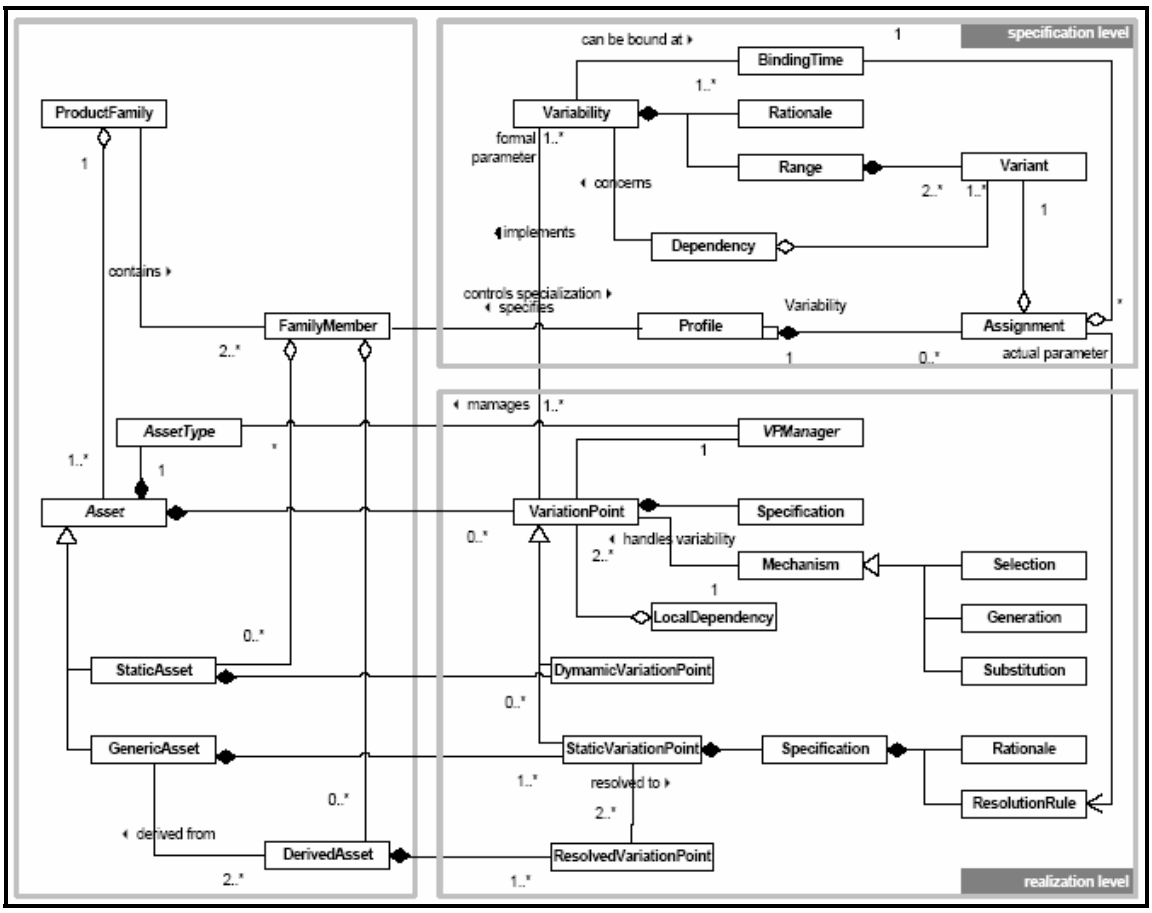

Figura 12 - Meta-modelo genérico de variabilidade
Fonte: Becker (2003, p. 6).

Segundo Becker (2003), a utilização deste meta-modelo possibilita maior facilidade e flexibilidade no desenvolvimento de ferramentas de apoio ao gerenciamento de variabilidades.

2.2.7 Considerações Finais

Variabilidade é um conceito que não está presente somente quando consideramos a abordagem de LP. *Frameworks* são excelentes exemplos de como as decisões adiadas de projeto permitem aumentar o número de possíveis sistemas computacionais que podem ser produzidos e de artefatos que podem ser reutilizados.

Gerenciar variabilidades requer saber exatamente como identificá-las, representá-las, delimitá-las, implementá-las e, o mais importante, como monitorar a evolução dos artefatos variantes de um sistema computacional.

Atualmente, o que se percebe é que várias são as propostas de gerenciamento de variabilidade existentes, contudo elas são realizadas para resolver problemas específicos de abordagens adotadas pelos seus autores.

Dessa forma, existe uma carência de um processo de gerenciamento de variabilidade que permita guiar o gerente de LP a gerenciar as variabilidades de uma LP da forma mais genérica possível, sem levar em consideração a abordagem necessária ao gerenciamento.

2.3 PROCESSO DE DESENVOLVIMENTO DE LINHA DE PRODUTO DE SOFTWARE

Esta seção apresenta o processo de desenvovimento de LP que vem sendo proposto e utilizado pelo grupo de engenharia de software da UEM. Atualmente, este processo é formado pelas atividades de desenvolvimento da LP e de geação de produtos.

A seguir, o processo é ilustrado com o desenvolvimento de uma LP para WfMS. Os conceitos relativos à tecnologia de *workflow* encontram-se no Apêndice A, uma vez que vêm sendo repetidos em vários trabalhos realizados no grupo (LAZILHA, 2002; OLIVEIRA JUNIOR, 2002; HALMEMAN, 2003; NISHIMURA, 2004).

A arquitetura da LP tomou como base a arquitetura genérica e o modelo de referência da WfMC (WfMC, 1995; WfMC 1999) e o ExPSEE (*Experimental Process-centred Software Engineering Environment*) (GIMENES, 2002).

As seções seguintes apresentam o desenvolvimento da LP para WfMS.

2.3.1 Desenvolvimento da Linha de Produto

O desenvolvimento da LP para WfMS tomou como base o método Catalysis (D'SOUZA, 1999), portanto utiliza a UML como notação. Assim, os estágios do processo de desenvolvimento da LP são **Especificação de Requisitos**, **Especificação do Sistema**, **Projeto Arquitetural** e **Projeto Interno dos Componentes**. Além disso, mecanismos de representação de variabilidade foram utilizados com base nas propostas de Jacobson, Griss e Jonsson (1997), para diagramas de casos de uso, e Morisio, Travassos e Stark (2000), para diagramas de classes.

A seguir, são apresentados os estágios mencionados acima e a descrição dos artefatos produzidos.

2.3.1.1. Especificação de Requisitos

A especificação de requisitos identificou três atores envolvidos em um WfMS, que são ***WorkflowArchitectureManager***, responsável por criar e manter arquiteturas de *workfow* que são futuramente reutilizadas em vários projetos; ***WorkflowManager***, responsável por instanciar uma arquitetura para um determinado projeto e programar as tarefas para a execução de um determinado *workflow*; e ***WorkflowUser***, responsável por executar as tarefas programadas para um *workflow*.

Os artefatos resultantes deste estágio são o Modelo de Casos de Uso do Negócio da LP para WfMS, apresentado na Figura 13, e os diagramas de casos de uso para cada um dos atores.

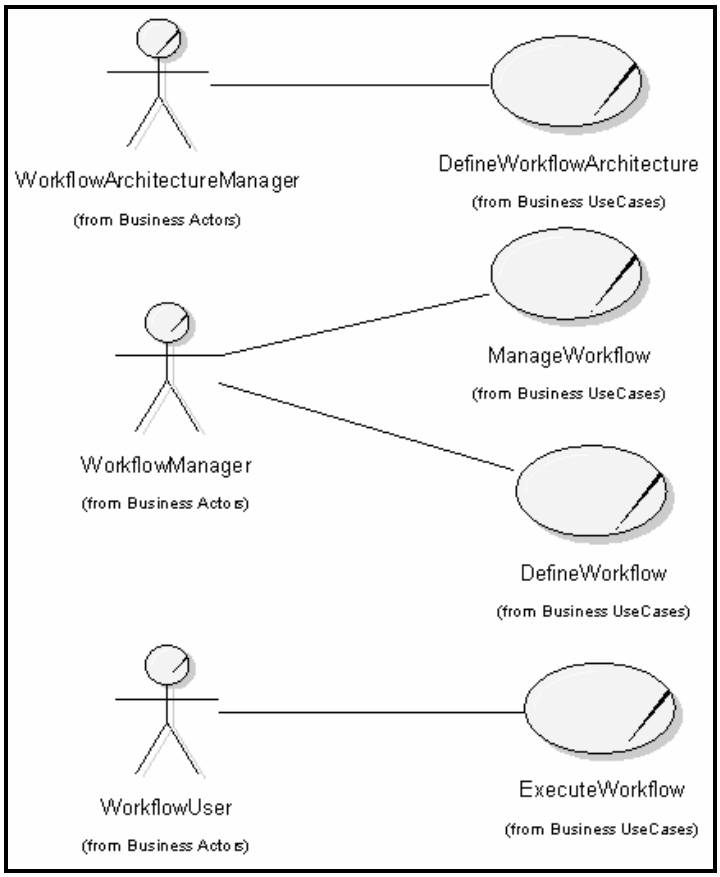

Figura 13 - Modelo de casos de uso de negócio para WfMS
Fonte: Nishimura (2004, p.113).

O caso de uso ***DefineWorkflowArchitecture*** define os elementos de uma arquitetura de *workflow*. O caso de uso ***DefineWorkflow*** instancia uma arquitetura de *workflow*, programa as tarefas de um *workflow* e, através do caso de uso ***ManageWorkflow***, é realizado o gerenciamento do progresso das tarefas de um *workflow*. O casos de uso ***ExecuteWorkflow*** executa as tarefas de um *workflow* utilizando os recursos definidos para estas.

A Figura 14, a Figura 15 e a Figura 16 apresentam os casos de uso para cada um dos atores de um WfMS.

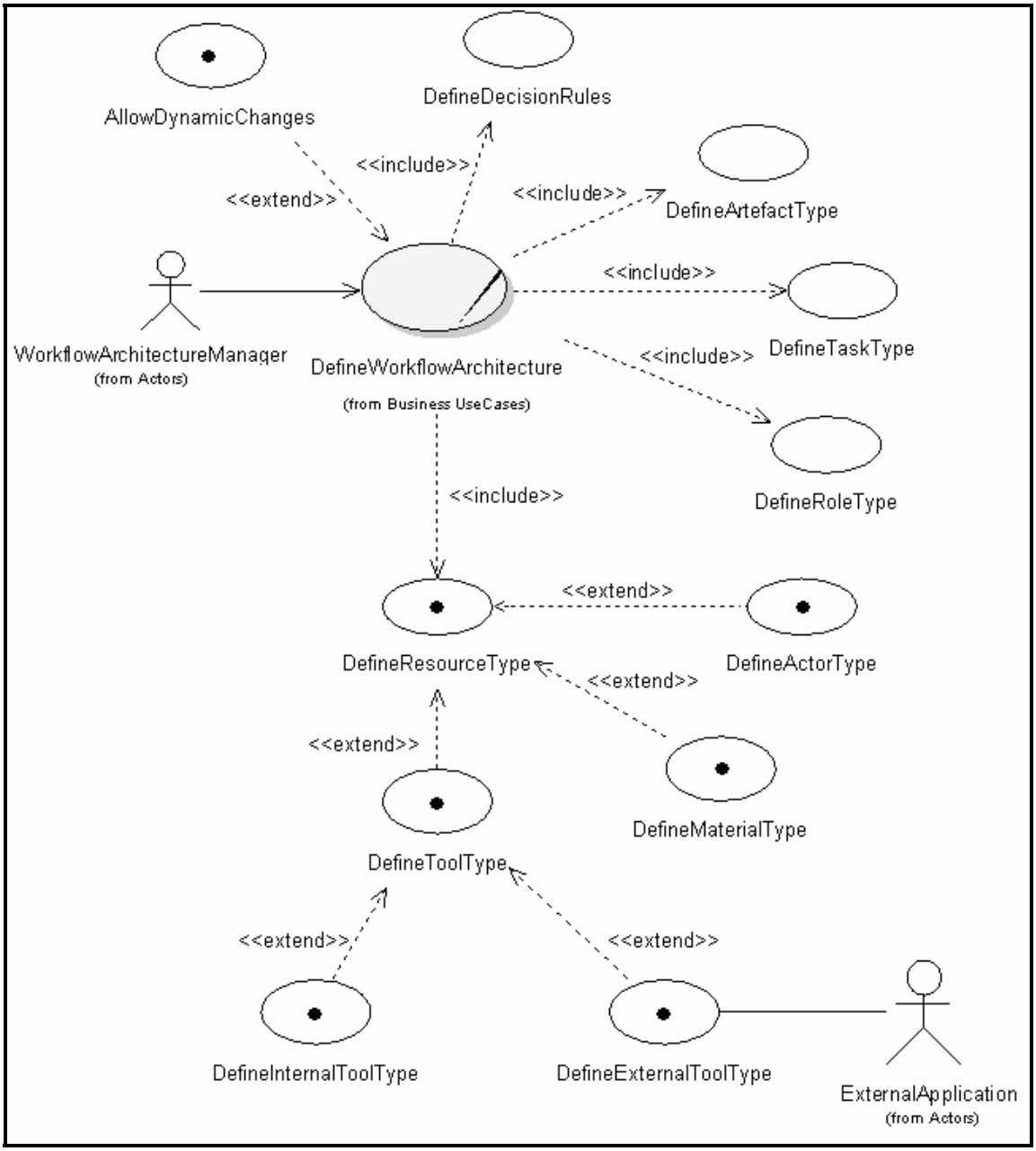

Figura 14 - Diagrama de casos de uso para o ator *WorkflowArchitectureManager*

Fonte: Nishimura (2004, p.115).

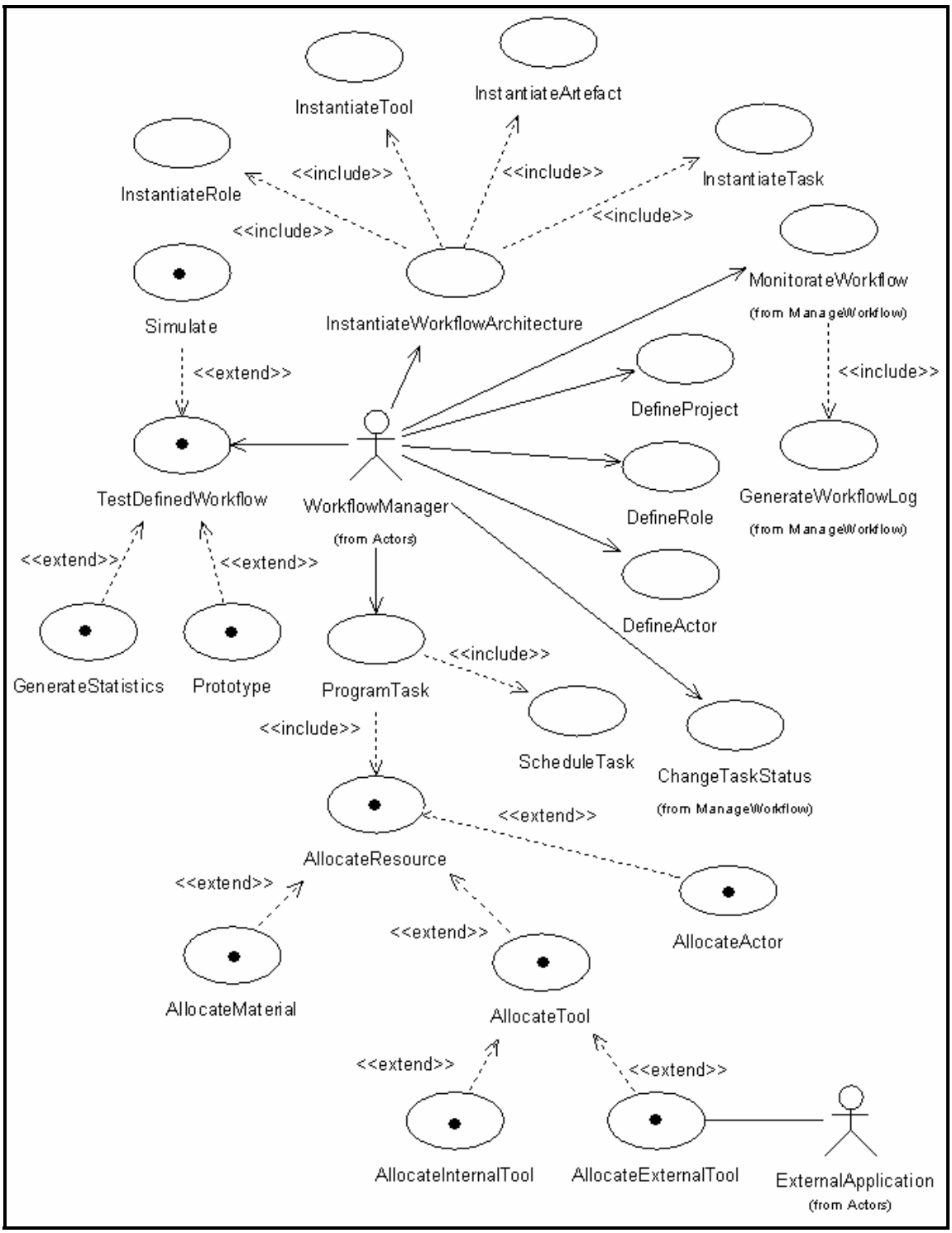

Figura 15 - Diagrama de casos de uso para o ator *WorkflowManager*
Fonte: Nishimura (2004, p. 116).

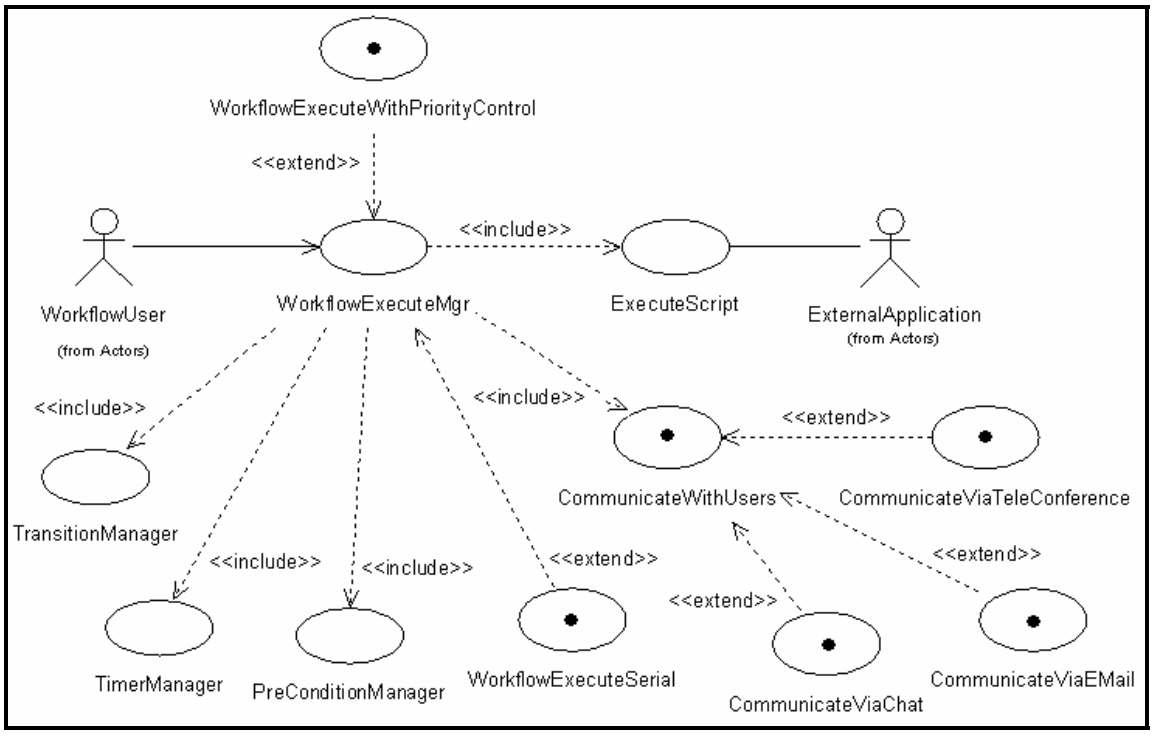

Figura 16 - Diagrama de casos de uso para o ator *WorkflowUser*
Fonte: Nishimura (2004, p. 117).

A representação de variabilidade nos casos de uso é feita utilizando-se a proposta de Jacobson, Griss e Jonsson (1997), cuja idéia é marcar os casos de uso com a marca "•", indicando que o caso de uso marcado possui variação. Por exemplo, na Figura 14, a definição de um tipo de ferramenta sofre variação e é realizada por meio do caso de uso ***DefineToolType***, em que as possíveis variantes são os casos de uso ***DefineInternalToolType*** e ***DefineExternalToolType***. A utilização da marca "•" é extremamente restrita. Assim, para auxiliar na representação das variantes de um caso de uso, além da marca "•", também é utilizado o estereótipo da UML <<extend>>.

Um dos pontos fracos na proposta de Jacobson, Griss e Jonsson (1997) é a falta de informação no caso de uso que possui variação em relação às suas variantes. Tal representação não permite responder a questões como:

- Qual a cardinalidade do ponto de variação?

- Qual o tipo de relação existente entre o ponto de variação e as suas variantes?
- As variantes são mutuamente exclusivas?
- Qual o tempo de resolução do ponto de variação?

2.3.1.2. Especificação do Sistema

Neste estágio, é representada a solução de software identificada a partir do modelo de domínio. Analisando as ações do sistema, foram identificados os tipos e ações relacionadas (NISHIMURA, 2004).

Os artefatos produzidos nesse estágio foram o modelo estático de tipos para WfMS e os diagramas de seqüência para os casos de uso do negócio. A Figura 17 apresenta o modelo estático de tipos para WfMS. Os diagramas de seqüência não são apresentados, pois estes não possuem variabilidades representadas.

A representação das variabilidades em diagramas de classes é feita através do uso do estereótipo <<V>>, proposto por Morisio, Travassos e Stark (2000). Para exemplificar o seu uso, na Figura 17 a classe ***Tool*** representa um ponto de variação e possui o estereótipo <<V>>. As suas possíveis variantes são as classes ***InternalTool*** e ***ExternalTool***. As classes que representam as variantes, além de possuírem o estereótipo para variabilidade, possuem, também, uma relação de herança com o ponto de variação.

Assim como acontece com a proposta de representação de variabilidade em casos de uso (JACOBSON; GRISS; JONSSON, 1997), a representação em classes, proposta por Morisio, Travassos e Stark (2000), possui alguns pontos fracos com relação às informações sobre a variabilidade como, por exemplo, informações sobre a cardinalidade da variação, o tipo de relação entre o ponto de variação e suas variantes e o tempo de resolução do ponto de variação.

Outro ponto fraco é quando são tratados pontos de variação múltiplos, ou seja, quando um ponto de variação possui mais de um tipo de variabilidade, como ilustrado na Figura 7. Desse modo, a representação proposta por Morisio, Travassos e Stark (2000) se torna falha por não informar quais variantes fazem parte e quais não fazem parte de uma determinada variação em um mesmo ponto de variação.

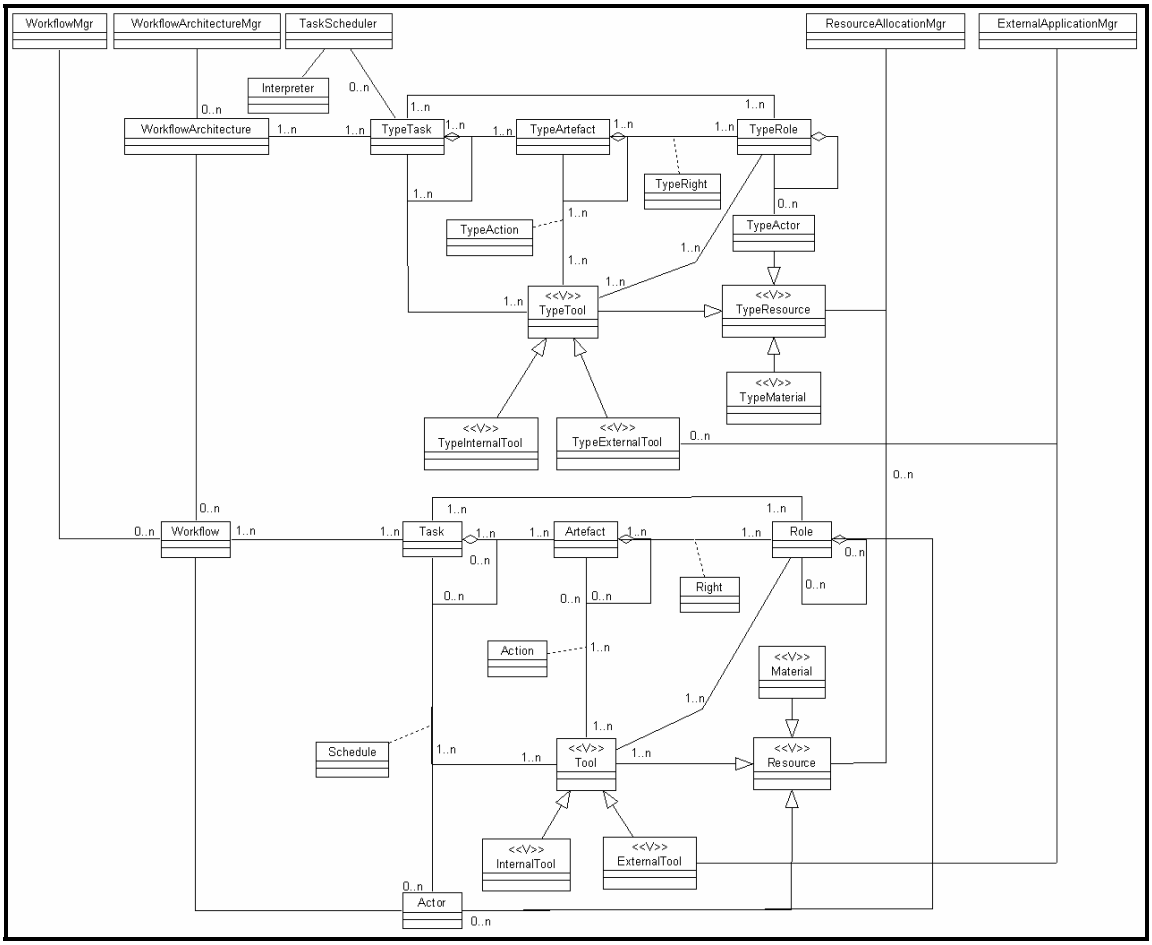

Figura 17 - Diagrama estático de tipos para WfMS

Fonte: Nishimura (2004, p. 118).

2.3.1.3. Projeto Arquitetural

Neste estágio, é realizado um refinamento desde os artefatos de mais alto nível até à arquitetura do sistema. O Catalysis considera os pacotes como a unidade de decomposição de mais alto nível, sendo que eles podem ser considerados partes do sistema que podem ser

tratados como unidades independentes e candidatas a se tornarem componentes no estágio seguinte.

O principal artefato produzido nesse estágio é o diagrama de camadas verticais de alto nível, apresentado na Figura 18.

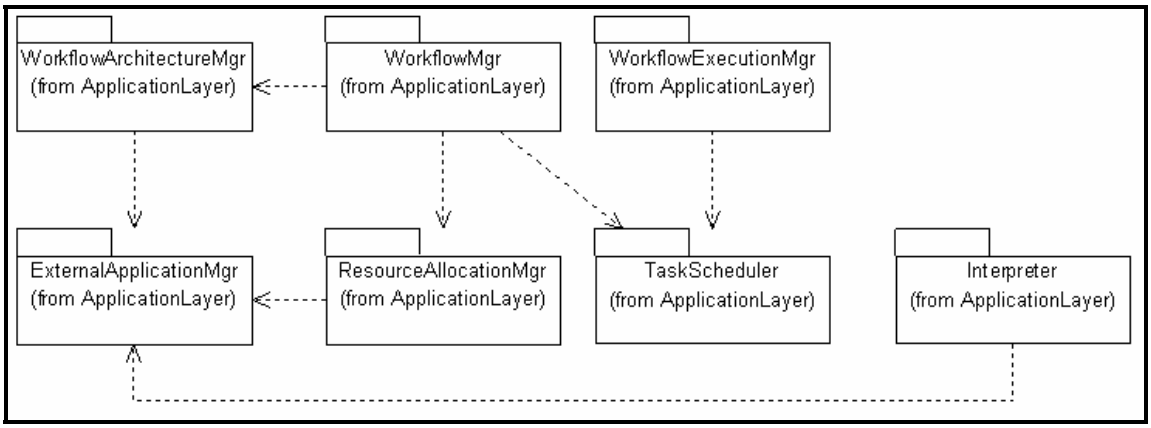

Figura 18 - Diagrama de camadas verticais de alto nível para WfMS
Fonte: Nishimura (2004).

2.3.1.4. Projeto Interno dos Componentes

Neste estágio, são especificados os componentes que formam a arquitetura do sistema. A especificação é realizada tomando como base os artefatos produzidos nos estágios anteriores, assim como o diagrama de camadas verticais de alto nível.

Os componentes são, então, especificados e as suas interfaces são definidas. O artefato resultante desse estágio é o diagrama de componentes para WfMS, mostrado na Figura 19.

O diagrama da Figura 19 não apresenta nenhuma representação de variabilidade, porém, com base nos diagramas de casos de uso e diagramas de classes, sabe-se que existem variabilidades em alguns componentes. Assim, questões relacionadas à variabilidade não ficam claras nesse tipo de artefato.

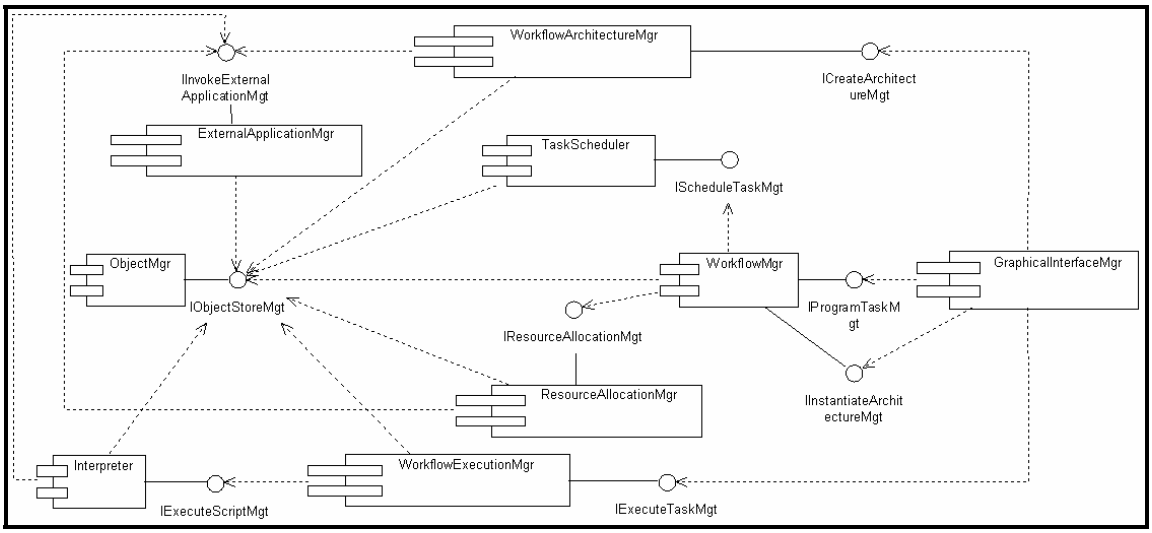

Figura 19 - Arquitetura de componentes para WfMS
Fonte: Nishimura (2004, p. 41).

A seguir, é apresentada uma breve descrição dos componentes que formam a arquitetura de LP para WfMS da Figura 19 (NISHIMURA, 2004):

- ***GraphicalInterfaceMgr*:** este componente é responsável pelo gerenciamento da interface com o usuário do sistema. O ponto de variação deste componente é a interface com o usuário, que pode ocorrer através da interface gráfica convencional ou por um navegador HTTP (*HyperText Transfer Protocol*);
- ***WorkflowArchitectureMgr*:** este componente é responsável pelo controle e gerenciamento da construção e manutenção de arquiteturas de *workflow*, suportando as funções relacionadas à definição de arquiteturas de *workflow* e dos tipos de objetos relacionados a esta. Os pontos de variação associados a este componente indicam o tipo de recurso que pode ser utilizado, podendo ser ator, ferramenta e material. O tipo de ferramenta pode ser especializado em interna e externa;
- ***WorkflowMgr*:** este componente é responsável pela criação e gerenciamento de projetos que incorporam *workflows*. Os projetos incluem a instanciação e

execução de arquiteturas de *workflow*. A cada projeto existe um *workflow* associado. Para cada tipo de objeto existente na arquitetura, instancia-se um objeto no *workflow*. Em seguida, as tarefas do projeto são executadas e gerenciadas, fazendo-se a alocação de recursos e demais tomadas de decisão;

- ***WorkflowExecutionMgr*:** este componente é responsável pela execução das tarefas de um *workflow*;
- ***TaskScheduler*:** este componente é responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas. O gerenciador de tarefas permite que os usuários identifiquem as tarefas geradas pelo gerenciador de *workflow*. Os pontos de variação deste componente estão associados aos tipos de recursos que podem ser utilizados (material, ferramenta e ator). A ferramenta utilizada pode ser do tipo interna ou externa. As tarefas podem ter prioridades de execução diferentes e os algoritmos utilizados para escalonamento das tarefas também podem variar;
- ***ResourceAllocationMgr*:** este componente é responsável pela alocação de recursos (atores, ferramentas ou material). Além da variabilidade associada ao tipo de recurso e ao tipo de ferramenta, as políticas de alocação de recurso podem variar;
- ***ExternalApplicationMgr*:** este componente é responsável pelo gerenciamento das aplicações externas invocadas durante a definição do processo e execução das tarefas. Pontos de variação incluem as diferentes formas de adaptação da aplicação externa ao WfMS;
- ***ObjectMgr*:** este componente é responsável pelo relacionamento com os mecanismos de armazenamento de objetos manipulados pelo sistema. Suas

funções trabalham com objetos, que podem ser considerados desde dados de controle do processo, dados relevantes do processo ou até mesmo instâncias de *workflow* e tarefas. Todos os componentes pertencentes à arquitetura utilizam seus serviços. Sua presença torna o restante dos componentes independentes de uma implementação particular de um sistema gerenciador de objetos, garantindo flexibilidade e portabilidade para toda a coleção de componentes existentes. Pontos de variação deste componente incluem os adaptadores para os sistemas de gerenciadores de bancos de dados;

- ***Interpreter*:** uma linguagem para programação de processos é necessária para que o processo possa ser programado e executado pelo gerenciador de *workflow*. As tarefas que compõem o *workflow* são então programadas utilizando uma linguagem de programação de processos e recursos que permitem a execução cooperativa dessas tarefas. Para a execução dessas tarefas, o gerenciador de tarefas realiza chamadas ao interpretador da linguagem para que este possa executá-las.

Informações mais detalhadas sobre o processo de desenvolvimento da arquitetura de LP podem ser encontradas em Lazilha (2002).

2.3.2 Geração de Produtos Específicos em LP

A geração de produtos específicos em LP (NISHIMURA, 2004) utiliza alguns artefatos do *Unified Software Development Process* (USDP) (KRUCHTEN, 2000), o processo de análise e projeto do Catalysis (D´SOUZA, 1999), o conceito de geração de produtos da abordagem PLP/SEI (CLEMENTS; NORTHROP, 2001) e alguns artefatos de tratamento de variabilidade do método KobrA (ATKINSON et al., 2001).

A Figura 20 apresenta as fases do processo de geração de produtos em LP, que corresponde à atividade de desenvolvimento do produto, proposta pelo SEI (2004).

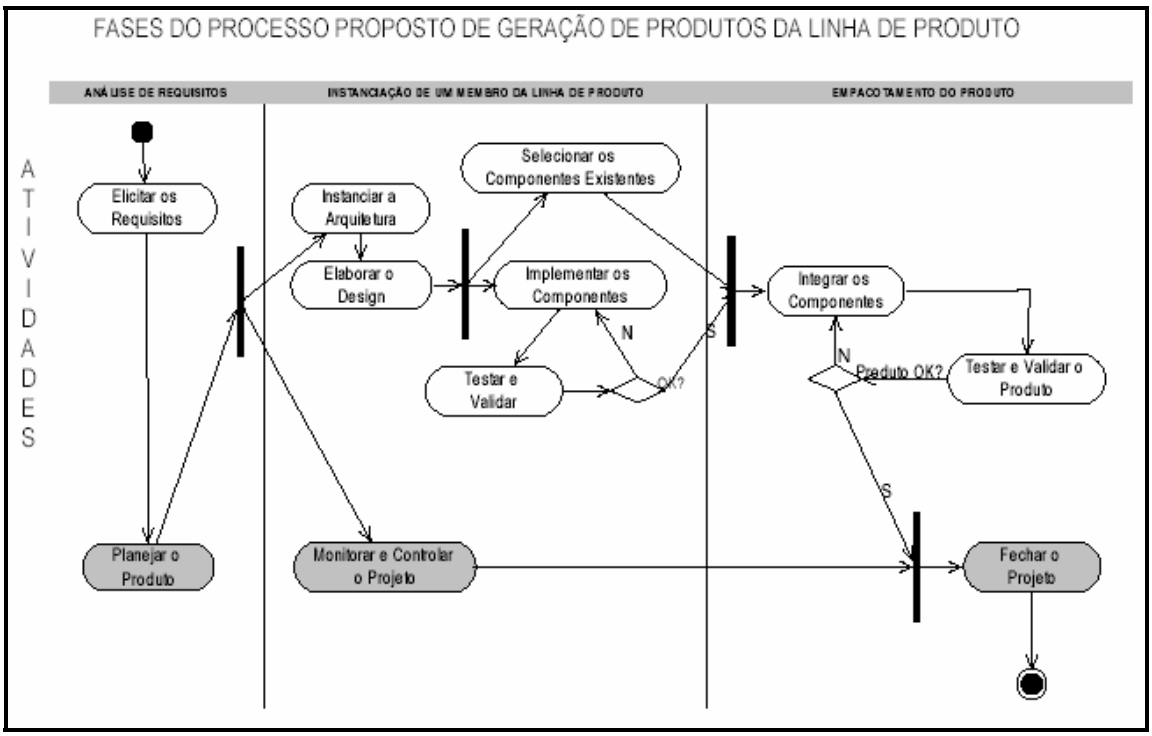

Figura 20 - Fases do processo de geração de produtos em LP
Fonte: Nishimura (2004, p.44).

Os itens a seguir apresentam uma breve descrição das fases do processo de geração de produtos, apresentado na Figura 20 (NISHIMURA, 2004):

- **Análise de Requisitos:** permite a identificação dos requisitos específicos a um produto. Os requisitos identificados podem se tornar requisitos da LP. Grande parte das variabilidades da LP é resolvida durante esta fase, que é composta por elicitação de requisitos e planejamento da geração do produto. A atividade de elicitação de requisitos é guiada pelo documento de requisitos da LP. A atividade de planejamento da geração do produto elabora o cronograma, constitui a equipe de desenvolvimento, calcula o custo do projeto e define a forma de acompanhamento e controle da geração do produto;

- **Instanciação de um Membro da LP:** esta fase é composta pelas atividades de instanciar a arquitetura, de elaborar o *design*, de selecionar os componentes e de implementar os novos componentes. Nessa fase, todas as variabilidades pendentes são resolvidas para atender às especificações do novo produto. Para isto, é utilizado um modelo de decisão que é um documento contendo as opções visíveis para o gerente de LP das características de um produto específico (Figura 21). O documento de resolução do modelo de decisão contém as variabilidades do modelo de decisão, resolvidas para um determinado produto (Figura 22);
- **Empacotamento do Produto:** esta fase é composta pelas atividades de integração dos componentes, de realização de testes, de validação da integração e de fechamento do produto. Os componentes implementados e os componentes selecionados do repositório de componentes são integrados para formar o produto específico.

ID	Questão	Ponto de Variação	Alternativas	Efeitos
WA1	Definição dos Tipos de Materiais?	DefineMaterialType	Sim ou Não	Sim: 1) retirar o estereótipo "Variability Use Case" do caso de uso "DefineMaterialType"; 2) resolver a decisão de ID "WF1" com a resposta "Sim"; 3) executar o script "VariabilityHandler.ebs"; Não: 1) alterar o estereótipo do caso de uso "DefineMaterialType" para "Variability Use Case Deleted"; 2) resolver a decisão de ID "WF1" com a resposta "Não"; 3) executar o script "VariabilityHandler.ebs"; 4) remover a classe "TypeMaterial"; 5) remover o caso de uso "DefineMaterialType".

Figura 21 - Exemplo de modelo de decisão

Fonte: Nishimura (2004, p. 57).

ID	Questão	Ponto de Variação	Resolução
WA1	Definição dos Tipos de Materiais?	DefineMaterialType	Sim

Figura 22 - Exemplo de resolução de modelo de decisão
Fonte: Nishimura (2004, p. 57).

A Figura 23 apresenta a seqüência das atividades do processo de geração de produtos em LP (NISHIMURA, 2004).

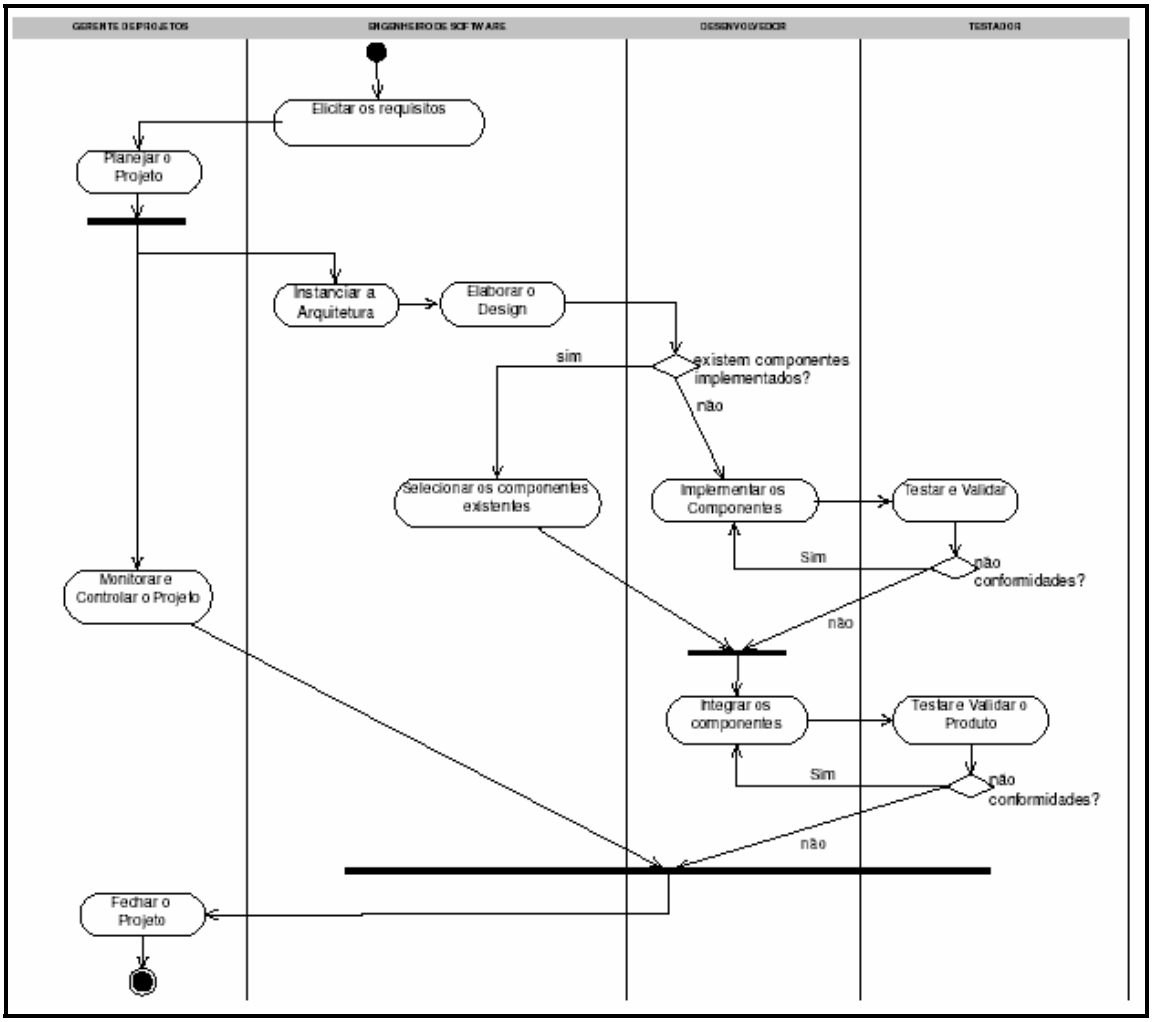

Figura 23 - Seqüência de atividades do processo de geração de produtos em LP
Fonte: Nishimura (2004, p. 48).

Informações mais detalhadas sobre o processo de geração de produtos em LP podem ser encontradas em Nishimura (2004).

2.3.3 Considerações Finais

O desenvolvimento da LP para WfMS seguiu o processo de desenvolvimento de LP existente com base no método Catalysis e na UML como notação.

Vários trabalhos de conclusão de curso, monografias de especialização e dissertações de mestrado foram realizados com base na LP para WfMS. Dentre os principais trabalhos, estão os desenvolvidos por Lazilha (2002), Oliveira Junior (2002), Halmeman (2003) e Nishimura (2004).

A LP para WfMS é fundamental para a realização deste trabalho, pois é a base para a realização do estudo de caso para a avaliação do processo proposto de gerenciamento de variabilidade.

CAPÍTULO 3

PROCESSO DE GERENCIAMENTO DE VARIABILIDADE PARA LINHA DE PRODUTO DE SOFTWARE

Este capítulo apresenta o processo de gerenciamento de variabilidade para LP, bem como as justificativas para a sua proposta, as contribuições deste em relação à literatura existente e as atividades e os artefatos que compõem tal processo.

3.1 CARACTERIZAÇÃO DO PROCESSO

A adoção de um processo de gerenciamento de variabilidade permite um melhor gerenciamento de LP, pois contribui para aumentar tanto a qualidade dos artefatos produzidos quanto o número de possíveis produtos a serem construídos a partir de um domínio. Para tanto, as atividades deste processo devem estar bem documentadas, assim como os seus artefatos.

A elaboração do processo de gerenciamento de variabilidade proposto neste trabalho de mestrado tomou como base as atividades de gerenciamento de variabilidade, propostas por Van Gurp e Bosch (2001), a construção de modelos de *features*, proposta por Griss, Favaro e D'Alessandro (1998) e a construção de um modelo genérico de variabilidades, proposta por Becker (2003). O processo de gerenciamento de variabilidade contribui para apoiar o processo de LP existente (Seção 2.3) em suas atividades de desenvolvimento de LP e de geração de produtos específicos.

O processo de gerenciamento de variabilidade é composto por atividades e artefatos como mostra a Figura 24. O processo é reaizado ao final de cada atividade do

desenvolvimento de LP, tornando iterativo. Ele também é um processo incremental, uma vez que a cada nova iteração o número de variabilidades aumenta.

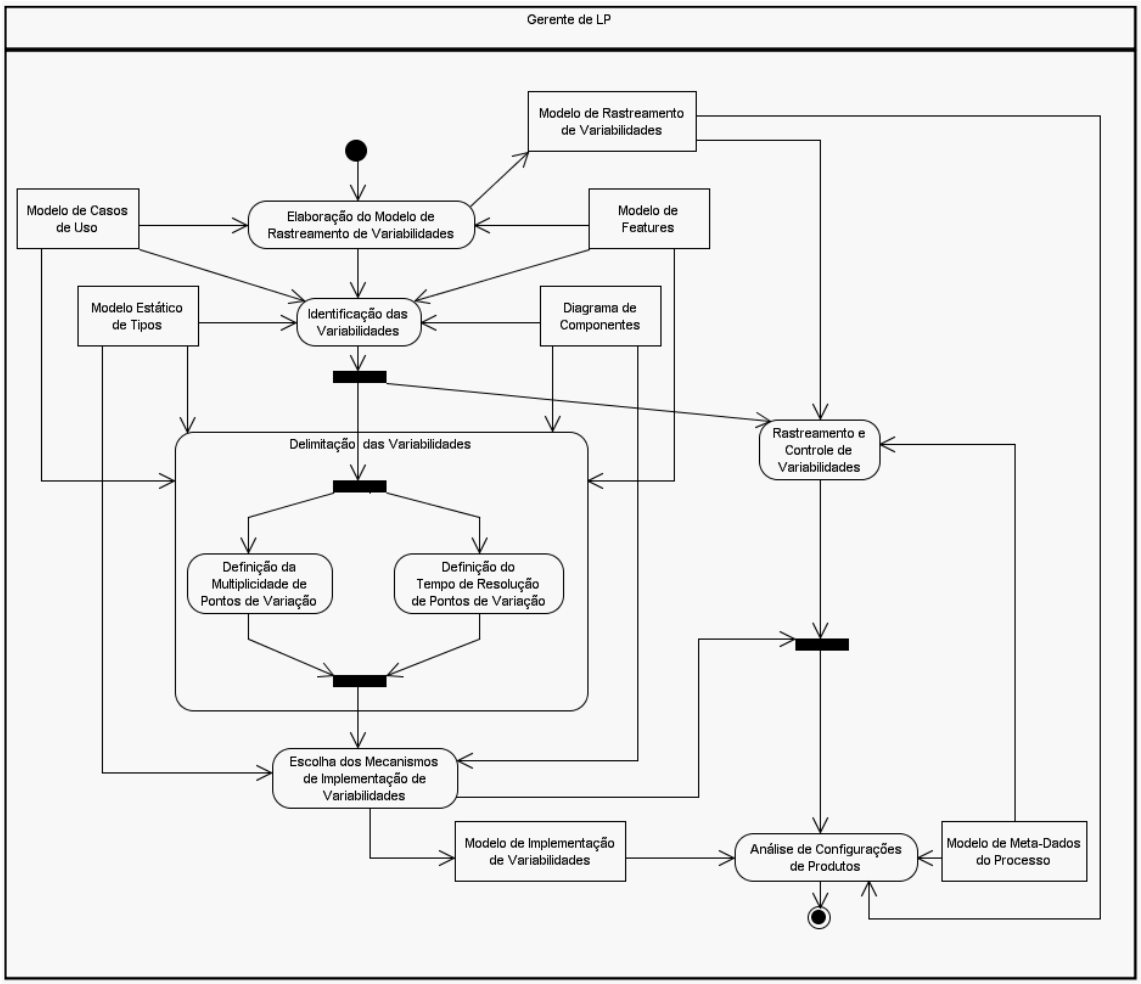

Figura 24 – Atividades do processo de gerenciamento de variabilidade para LP.

No diagrama de atividades (Figura 24) que representa o processo proposto, as atividades são representadas pelas elipses e os artefatos de entrada e saída são representados pelos retângulos. Todas as atividades do processo são realizadas pelo Gerente de LP.

O processo é visto como um elemento que consome informações provenientes da atividade de desenvolvimento da LP e também produz informações para tal atividade como, por exemplo, o modelo de casos de uso e o modelo estático de tipos com as variabilidades identificadas e delimitadas. O processo também produz informações para a atividade de

geração de produtos específicos como, por exemplo, o modelo de rastreamento de variabilidades e o modelo de implementação de variabilidades. A Figura 25 mostra o relacionamento entre o gerenciamento de variabilidade e as atividades de desenvolvimento de LP e geração de produtos. Nesta figura, o processo de gerenciamento de variabilidade está representado pelo retângulo hachurado em cinza e corresponde a um dos elementos da atividade de gerenciamento de LP propostas pelo SEI (Figura 1).

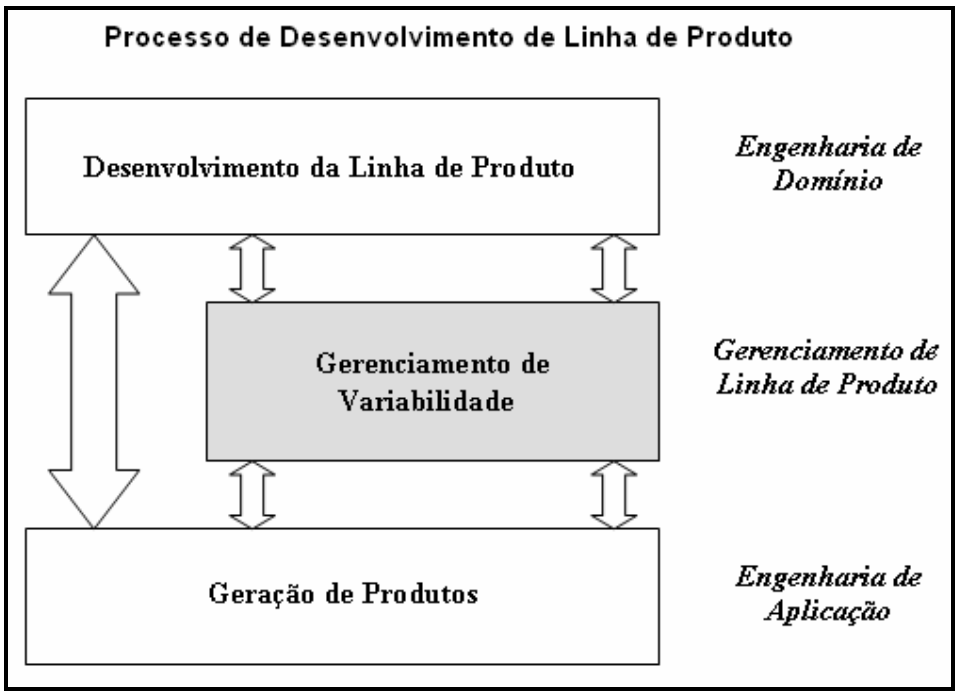

Figura 25 - Processo de desenvolvimento de LP existente com gerenciamento de variabilidade.

A Figura 26 mostra a interação entre as atividades de desenvolvimento da LP, realizadas pelo engenheiro de LP, e as do gerenciamento de variabilidade, realizadas pelo gerente de LP. A figura mostra que ao final de cada atividade do desenvolvimento da LP é realizada uma iteração do gerenciamento de variabilidade. Comparando esta figura com a abordagem PLP ilustrada na Figura 1, as atividades alinhadas verticalmente à esquerda correspondem ao ciclo superior esquerdo das atividades propostas pelo SEI (Figura 1), enquanto as demais atividades correspondem ao ciclo inferior.

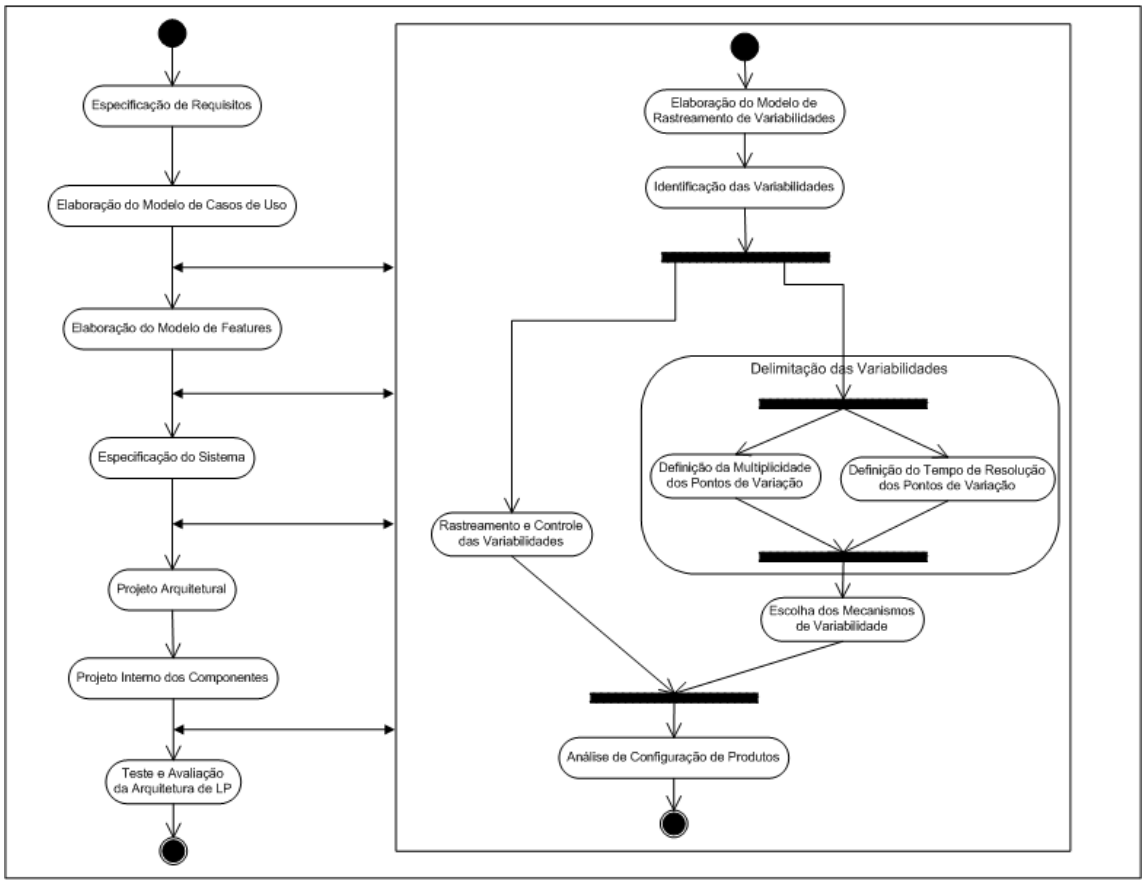

Figura 26 – Interação entre as atividades de desenvolvimento da LP e as do gerenciamento de variabilidade.

A concepção do processo de gerenciamento de variabilidade permitiu analisar o processo de LP existente, o que mostrou a necessidade de inclusão de algumas atividades nesse processo.

As seções a seguir apresentam a descrição das atividades incorporadas ao processo de LP existente e a descrição das atividades que compõem o processo de gerenciamento de variabilidade, bem como os artefatos utilizados e os papéis que realizam tais atividades. A descrição dessas atividades utiliza a UML como notação. Um exemplo básico de compra via Internet é utilizado para ilustrar as atividades.

3.2 ATIVIDADES INCORPORADAS AO PROCESSO DE DESENVOLVIMENTO DE LP EXISTENTE

O processo de gerenciamento de variabilidade exige alguns artefatos de entrada que não faziam parte do processo de LP existente como, por exemplo, o modelo de *features*. Assim, uma das contribuições deste trabalho é a adaptação do processo existente às necessidades do processo de gerenciamento de variabilidades. A atividade de "Elaboração do Modelo de *Features*" foi incluída no processo de LP existente. Além dessa atividade, também se faz necessária uma atividade para testar e avaliar a arquitetura de LP. Tal atividade demanda um estudo minucioso sobre as técnicas de teste e avaliação existentes na literatura. Neste trabalho não é realizado tal estudo, portanto a proposta desta atividade ficará como trabalho futuro.

As seções a seguir apresentam a atividade de "Elabotação do Modelo de Casos de Uso" somente para ilustrar o exemplo de compra via Internet e a atividade de "Elaboração do Modelo de *Features*", incorporada ao processo de LP existente. Para ilustrar o exemplo de compra via Internet são utilizados casos de uso de sucesso representando refinamentos de casos de uso de mais alto nível de abstração.

3.2.1 Elaboração do Modelo de Casos de Uso

O modelo de casos de uso é fundamental para o desenvolvimento de LP, pois é o ponto de partida para a identificação das *features* e, como conseqüência, das variabilidades de uma LP.

Esta atividade tem como artefato de entrada o documento de especificação de requisitos da LP, pois é a partir dele que se constrói o modelo de casos de uso, que é o seu

artefato de saída. Este modelo mostra os casos de uso da LP e os atores envolvidos, em um alto nível de abstração.

A Figura 27 apresenta um modelo de casos de uso para um processo de compra via Internet.

O modelo de casos de uso serve como artefato de entrada para a atividade de "Elaboração do Modelo de *Features*" e para o processo de gerenciamento de variabilidade. Além disso, esse modelo evolui na medida em que as variabilidades vão sendo identificadas e incorporadas ao modelo através da notação proposta (Seção 3.3).

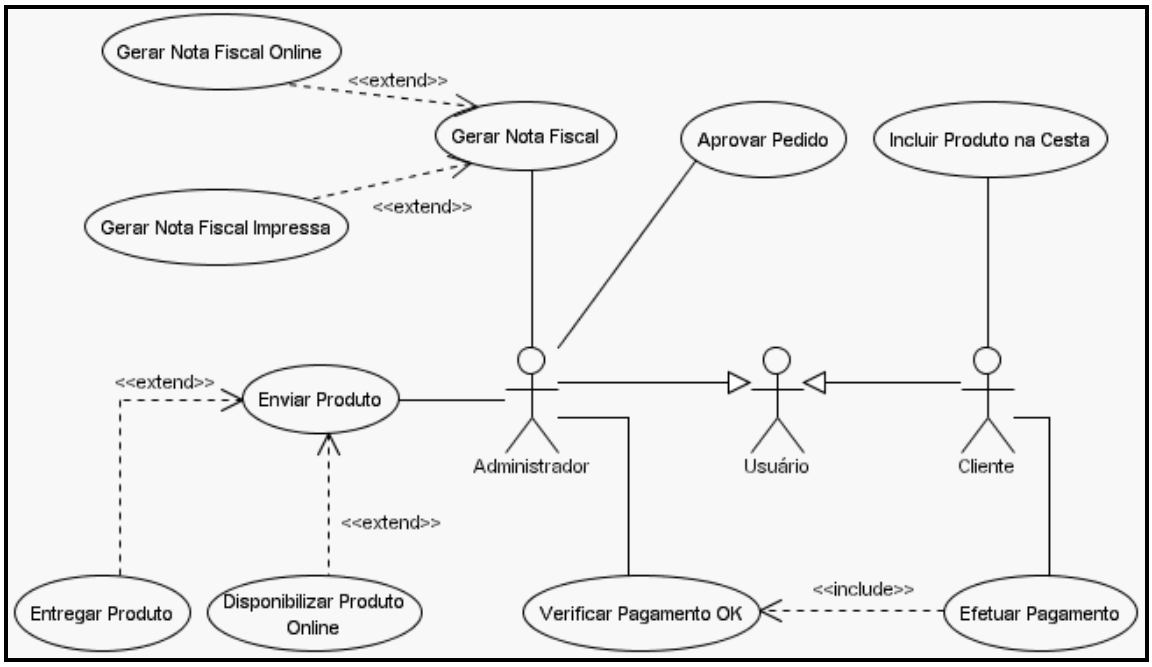

Figura 27 - Modelo de casos de uso de um processo de compra via Internet.

3.2.2 Elaboração do Modelo de *Features*

Esta atividade visa construir o modelo de *features* da LP a partir do modelo de casos de uso. A elaboração do modelo de *features* segue as etapas de identificação das *features*, definição do tipo das *features* e representação gráfica das *features*. A seguir, são apresentadas tais etapas de uma forma mais detalhada.

3.2.2.1. Identificação das *Features* de uma LP

A identificação das *features* de uma LP tem como base a proposta de Griss, Favaro e D'Alessandro (1998). Como os requisitos de uma LP são representados pelo modelo de casos de uso e as *features* são as abstrações dos requisitos de uma LP, é possível identificar *features* a partir do modelo de casos de uso.

Para tanto, deve-se proceder da seguinte maneira:

1. identificar grupos de casos de uso, relacionados entre si ou através de um ator, que representam pontos de reutilização da LP. Por exemplo, os casos de uso "Gerar Nota Fiscal", "Gerar Nota Fiscal *Online*" e "Gerar Nota Fiscal Impressa" (Figura 27) podem ser agrupados, pois podem ser vistos como pontos de reutilização pela forma como a geração de uma nota fiscal pode variar: *online* e/ou impressa;
2. para cada grupo de casos de uso identificado, criar uma ou mais *features*. O agrupamento apontado no item anterior poderia gerar a *feature* "Nota Fiscal", que pode ser composta pelas *features* "*Online*" e "Impresso".

O modelo de *features* permite representar *features* funcionais, abstrações dos requisitos funcionais e *features* não-funcionais, abstrações dos requisitos não-funcionais.

Os grupos de casos de uso e as *features* são utilizados na elaboração do modelo de rastreamento de variabilidades na atividade seguinte. A representação das *features* utilizando diagramas de classes é apresentada na Seção 3.2.2.3.

3.2.2.2. Definição do Tipo das *Features*

Nesta etapa, é definido o tipo de cada *feature* identificada na etapa anterior.

Para isso, deve-se classificá-las como sendo:

- **obrigatórias**: são as *features* que obrigatoriamente devem estar presentes em um produto da LP. Por exemplo, considerando o exemplo do processo de compra via Internet, tem-se as *features* “Envio”, “Pagamento” e “Nota Fiscal”;
- **opcionais**: são as *features* que podem ou não estar presentes em um produto da LP. Um exemplo seria a *feature* “Cesta de Compras” que, para alguns produtos, pode existir e, para outros, não;
- **alternativas**: conjunto de duas ou mais *features* que podem ser classificadas como:
    - **inclusivas**: indica que zero ou mais *features* de um conjunto podem fazer parte de um produto da LP. As *features* alternativas inclusivas são representadas pela relação OR da lógica booleana. Um exemplo poderia ser o conjunto formado pelas *features* “*Online*” e “Impresso” como alternativas à *feature* “Nota Fiscal”. Um produto, nesse caso, poderia conter a emissão de nota fiscal de forma *online*, impressa ou ambas;
    - **exclusivas**: possibilita que uma e somente uma *feature* de um conjunto possa fazer parte de um produto da LP. As *features* alternativas exclusivas são representadas pela relação XOR (*Exclusive OR*) da lógica booleana. Um exemplo poderia ser o conjunto formado pelas *features* “Boleto Bancário”, “Cartão de Crédito” e “Pagamento na Entrega” como alternativas à *feature* “Pagamento”. Um produto, nesse caso, poderia conter o pagamento ou na forma de boleto bancário ou de cartão de crédito ou realizar o pagamento no ato da entrega do produto, porém nunca duas ou mais formas em um mesmo produto da LP;

- **externas**: são as *features* relacionadas aos requisitos não-funcionais como, por exemplo, a plataforma de execução do produto, a quantidade mínima de memória exigida e o tipo de servidor de banco de dados. Um exemplo poderia ser a *feature* "Plataforma" que determina em qual plataforma o produto será executado.

Além dos tipos de cada uma das *features*, é preciso definir, também, as relações entre as *features*, as quais podem ser de dois tipos:

- ***requires***: essa relação indica que para uma *feature* pertencer a um produto da LP, uma ou mais *features* também devem estar presentes. Um exemplo poderia ser a relação existente entre as *features* "Cartão de Crédito", que é uma alternativa à *feature* "Pagamento", e "*Online*", que é uma alternativa à *feature* "Nota Fiscal". Para um determinado produto, o pagamento via cartão de crédito pode exigir que o comprovante (nota fiscal) seja gerado de forma *online*.

- ***mutex***: essa relação indica que para uma *feature* fazer parte de um produto da LP, uma ou mais *features* não podem estar presentes no mesmo produto. Um exemplo poderia ser a relação existente entre as *features* "Pagamento na Entrega", que é uma alternativa à *feature* "Pagamento", e "Eletrônico" que é uma alternativa à *feature* "Envio". Para um determinado produto, o pagamento na entrega não faz sentido se o produto for entregue de forma eletrônica.

3.2.2.3. Representação Gráfica do Modelo de *Features*

Nesta etapa, as *features* identificadas e definidas nas etapas anteriores e as relações entre elas são representadas graficamente.

Para isso, utiliza-se um diagrama de classes da UML, em que as *features* são representadas por classes, como proposto por Clauß (2001). Além das classes, são utilizados três tipos de relações da UML entre as classes que representam as *features,* que são composição, generalização/especialização e dependência.

A relação de composição indica que uma *feature* é formada por outras *features*, independentes dos seus tipos (obrigatória, opcional, alternativa ou externa). Um exemplo seria a *feature* "Envio", que pode ser composta pelas *features* "Entrega" e "Eletrônico".

A relação de generalização/especialização é usada única e exclusivamente para representar a relação entre as *features* alternativas exclusivas. Um exemplo seria a relação entre as *features* "Pagamento" e as suas alternativas "Boleto Bancário", "Cartão de Crédito" e "Pagamento na Entrega".

A relação de dependência é utilizada para representar as relações *requires* e *mutex* entre *features*.

A representação do modelo de *features* por meio de diagramas de classes utiliza, também, estereótipos para representar o tipo de cada *feature,* que são <<mandatory>>, para as *features* obrigatórias, <<optional>>, para as *features* opcionais, <<alternative_OR>>, para as *features* alternativas inclusivas, <<alternative_XOR>>, para as *features* alternativas exclusivas, e <<external>>, para as *features* externas. As relações *requires* e *mutex* utilizam os estereótipos <<requires>> e <<mutex>>, respectivamente, nas relações de dependência da UML.

A identificação das *features* alternativas que pertencem a um mesmo conjunto é feita utilizando-se uma nota da UML que indica {or} se a relação é inclusiva ou {xor} se a relação é exclusiva. Um exemplo é a utilização de uma nota com {or} indicando que as *features* alternativas inclusivas "Entrega" e "Eletrônico" pertencem a um mesmo conjunto.

No trabalho realizado por Gomaa (2005), a representação de variabilidade em modelos de *features* é feita utilizando estereótipos - por exemplo, <<zero-or-more-of feature group>>, indicando que zero ou mais *features* do grupo de *features* devem ser escolhidas, <<exactly-one-of feature group>> indicando que somente uma *feature* do grupo de *features* deve ser escolhida - que representam muito bem se as *features* variantes possuem relações inclusivas ou exclusivas. Porém, tal representação não permite dizer de fato quantas variantes serão escolhidas e qual o tempo de resolução de uma *feature* que representa um ponto de variação. A proposta de utilização de notas da UML deste trabalho permite a representação deste tipo de informação possibilitando maior compreensão das relações entre as *features*, além do tempo de resolução de cada uma destas.

A Figura 28 apresenta o modelo de *features* para um processo de compra via Internet. Além disso, essa figura mostra um exemplo de como representar as *features* identificadas e definidas nas etapas anteriores, utilizando, para isso, a notação proposta neste trabalho.

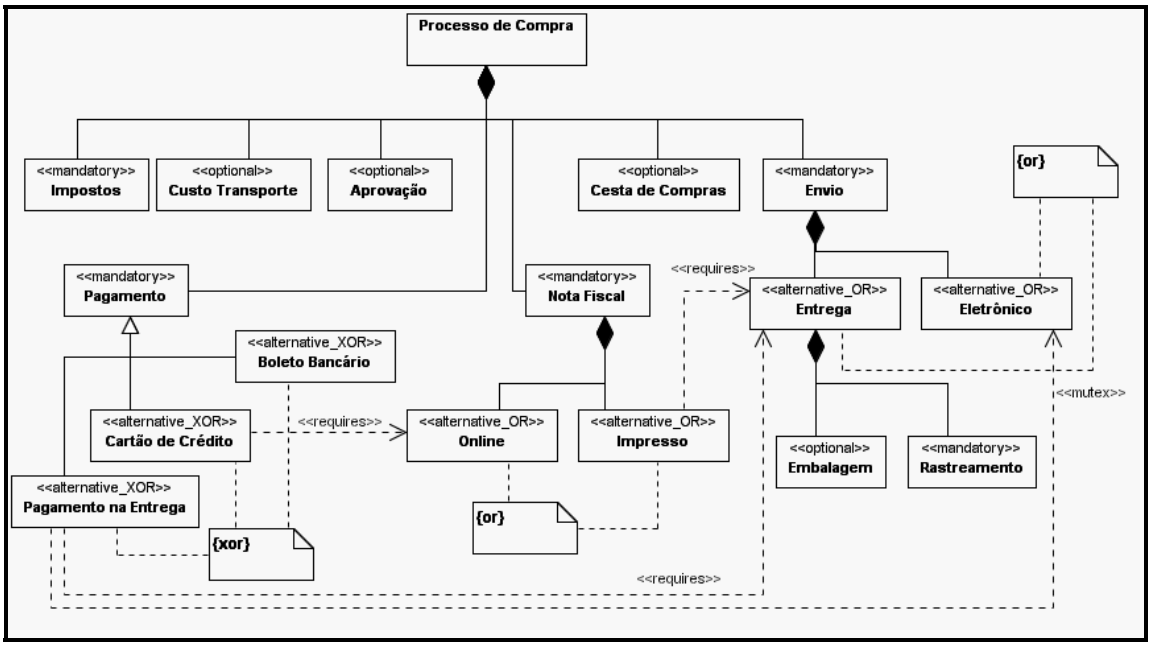

Figura 28 - Modelo de *features* de um processo de compra via Internet.

3.3 ATIVIDADES DO PROCESSO DE GERENCIAMENTO DE VARIABILIDADE

Esta seção apresenta as atividades e os artefatos de entrada e saída (vide Apêndice B) que compõem o processo proposto para gerenciamento de variabilidade em LP.

3.3.1 Elaboração do Modelo de Rastreamento de Variabilidades

Esta atividade tem como objetivo criar um modelo de rastreamento de variabilidades entre casos de uso e *features*. Tal modelo permite relacionar as variabilidades entre os artefatos da LP.

A elaboração do modelo de rastreamento de variabilidades toma como base o modelo utilizado para rastrear *features* e casos de uso da ferramenta IBM/Rational RequisitePro (IBM, 2004), em que se definem quais são os casos de uso que estão relacionados a quais *features* e vice-versa.

A atividade de elaboração do modelo de rastreamento de variabilidades tem como artefato de entrada o modelo de casos de uso e o modelo de *features*, construídos durante o desenvolvimento da LP, e obedece à relação n:m entre casos de uso e *features*. Assim, um caso de uso pode envolver várias *features* e uma *feature* em particular pode envolver mais de um caso de uso.

O Quadro 1 apresenta o modelo de rastreamento de variabilidades para um processo de compra via Internet. As linhas representam as *features* identificadas no modelo de *features*, enquanto as colunas representam os casos de uso identificados no modelo de casos de uso.

***Features* / Casos de Uso**	**Efetuar Pagamento**	**Enviar Produto**	**Gerar Nota Fiscal**
Pagamento	•		
Cartão de Crédito	•		
Boleto Bancário	•		
Pagamento na Entrega	•		
Nota Fiscal			•

Online	•		•
Impresso			•
Envio		•	
Entrega	•	•	•
Eletrônico	•	•	
Rastreamento		•	
Embalagem		•	

Quadro 1 - Modelo de rastreamento parcial de um processo de compra via Internet.

Para construir o modelo de rastreamento de variabilidades, deve-se proceder da seguinte maneira:

1. listar todas as *features* da LP;
2. listar todos os casos de uso da LP;
3. para cada *feature* listada, identificar os casos de uso que originam tal *feature*;
4. marcar com o símbolo "•" o cruzamento entre as *features* e os casos de uso relacionados. Por exemplo, o caso de uso "Gerar Nota Fiscal" origina a *feature* "Nota Fiscal" (Quadro 1).

Com este modelo, é possível, a partir de uma determinada *feature*, rastrear as variabilidades até chegar aos casos de uso e aos demais elementos da UML como, por exemplo, outros casos de uso, classes e componentes.

Dessa forma, é possível identificar quais são os casos de uso e os demais artefatos da LP afetados durante a escolha de determinadas *features* para um produto específico. Isto permite, por exemplo, simular a configuração de produtos específicos a partir do modelo de *features* da LP e analisar o impacto nos demais artefatos ao adicionar ou remover determinada *feature*.

3.3.2 Identificação das Variabilidades

Esta atividade tem como objetivo identificar as variabilidades de uma LP em diagramas de casos de uso, diagramas de classes e diagramas de componentes. Além disso, essa atividade visa, também, a representação gráfica das variabilidades.

Esta atividade tem como artefato de entrada o modelo de casos de uso, além do diagrama de classes e de componentes da LP. É realizada percorrendo-se os artefatos citados acima em busca de pontos de variação.

As seções a seguir apresentam suas etapas.

3.3.2.1. Identificação de Pontos de Variação, Variantes e de seus Relacionamentos

A identificação de pontos de variação em diagramas de casos de uso depende da habilidade do gerente de LP em reconhecer quais são os casos de uso que podem representar pontos de variação e quais casos de uso podem representar as variantes de um ponto de variação. Para isso, neste processo são propostas algumas orientações que podem ser seguidas:

- as variantes do tipo alternativas inclusivas ou exclusivas são normalmente identificadas através da relação com o estereótipo <<extends>>. Assim, os casos de uso, que representam variantes, são identificados com um dos estereótipos <<alternative_OR>> ou <<alternative_XOR>> enquanto os pontos de variação são representados com o estereótipo <<variationPoint>>. Por exemplo, na Figura 27, pode-se identificar o caso de uso “Gerar Nota Fiscal” como um ponto de variação e os casos de uso “Gerar Nota Fiscal *Online*” e “Gerar Nota Fiscal Impressa” como variantes alternativas inclusivas indicando que zero ou mais destas variantes podem ser escolhidas para resolver o ponto de variação;

- as variantes dos tipos opcionais e obrigatórias são normalmente identificadas como uma simples relação de associação. Por exemplo, na Figura 27, pode-se identificar o caso de uso “Incluir Produto na Cesta” como opcional, uma vez que um produto de software pode ou não conter a inclusão de um produto na cesta de compras, e o caso de uso “Aprovar Pedido” como obrigatório;
- existem casos em que é necessário representar as relações entre variantes, sendo estes quando uma variante exige a existência de outra e quando uma variante exige que outra variante não exista. Estas relações são representadas sempre através da relação de dependência da UML, com os estereótipos <<requires>> e <<mutex>>, respectivamente.

O mesmo acontece na identificação de pontos de variação em diagramas de classes, em que normalmente classes com a relação de herança representam variantes alternativas inclusivas ou exclusivas, enquanto classes com outros tipos de relações da UML (associação, agregação, composição e dependência) representam variantes do tipo opcionais ou obrigatórias. Um exemplo poderia ser uma classe “TipoEmbalagem” como sendo um ponto de variação e tendo como variantes alternativas inclusivas as classes “EmbalagemPapel” e “EmbalagemAcolchoada”.

A identificação de variabilidades em diagramas de componentes depende do conhecimento e experiência do gerente de LP e das classes que formam o componente. Os componentes formados por classes que possuem variação são também considerados como pontos de variação. Um exemplo poderia ser o componente “GerenciadorEnvio”, formado pela classe “TipoEmbalagem”, que possui duas variantes, o componente “GerenciadorEntregaOnline” e o componente “GerenciadorEntregaDomicílio”.

3.3.2.2. Representação Gráfica das Variabilidades

A representação gráfica das variabilidades em diagramas de casos de uso, diagramas de classes e diagramas de componentes tem como base alguns trabalhos como o de Clauß (2001), o de Bühne, Halmans e Pohl (2003), o de Gomaa e Webber (2004), e o de Gomaa (2005), e segue o Quadro 2.

Esta representação exige a evolução dos artefatos (diagramas de casos de uso, diagramas de classes e de componentes). Assim, tais artefatos devem ser modificados no sentido de incluir elementos que possibilitem a representação de variabilidades.

A Figura 29 apresenta a evolução do modelo de casos de uso da Figura 27 para um processo de compra via Internet em que as variabilidades são identificadas. A diferença na representação de variabilidade em casos de uso proposta neste trabalho com relação ao trabalho de Gomaa (2005) é que o autor utiliza o estereótipo <<kernel>> para representar variantes obrigatórias, enquanto neste trabalho é utilizado o estereótipo <<mandatory>> (Quadro 2). Além disso, Gomaa (2005) não utiliza estereótipos para representar as relações do tipo alternativas inclusivas e alternativas exclusivas, o que impossibilita identificar tais relações em diagramas de casos de uso. Outra contribuição importante deste trabalho na representação de variabilidade em casos de uso é a utilização dos estereótipos <<mutex>> e <<requires>> que representam relações de dependência entre casos de uso, o que não é abordado em Gomaa (2005).

	Diagramas de Casos de Uso		Diagramas de Classes		Diagramas de Componentes	
	Relação da UML	**Estereótipo do Elemento**	**Relação da UML**	**Estereótipo do Elemento**	**Relação da UML**	**Estereótipo do Elemento**
Ponto de Variação	--------	<<variationPoint>>	--------	<<variationPoint>>	Dependência	<<variable>>
Variante Obrigatória	Associação Agregação Composição Dependência	<<mandatory>>	Associação Agregação Composição Dependência	<<mandatory>>	Dependência	<<mandatory>>
Variante Opcional	Associação Agregação Composição Dependência	<<optional>>	Associação Agregação Composição Dependência	<<optional>>	Dependência	<<optional>>
Variante Alternativa Inclusiva	Esp. / General. com o estereótipo <<extend>>	<<alternative_OR>>	Herança	<<alternative_OR>>	Dependência	<<alternative_OR>>
Variante Alternativa Exclusiva	Esp. / General. com o estereótipo <<extend>>	<<alternative_XOR>>	Herança	<<alternative_XOR>>	Dependência	<<alternative_XOR>>
Variante Mutuamente Exclusiva	Dependência	<<mutex>>	Dependência	<<mutex>>	Dependência	<<mutex>>
Variante Inclusiva	Dependência	<<requires>>	Dependência	<<requires>>	Dependência	<<requires>>

Quadro 2 - Relações e estereótipos usados na representação gráfica de pontos de variação e variantes.

Observa-se que os casos de uso da Figura 29 são os mesmos da Figura 27, porém com os pontos de variação e variantes identificados. Além dos estereótipos propostos para representar variabilidades, são usadas notas da UML para representar informações adicionais sobre os pontos de variação e suas variantes como, por exemplo, o nome da variação e o seu tipo, sendo que:

- {} significa que a variante é obrigatória ou opcional;
- {or} significa que a variante é alternativa inclusiva;
- {xor} significa que a variante é alternativa exclusiva.

Nota-se, também, a existência de três "?". Estes símbolos indicam que as variabilidades ainda não foram delimitadas, o que será feito na próxima atividade do processo.

As informações sobre os pontos de variação na forma de notas da UML são uma contribuição deste trabalho com relação à literatura existente. Isto se deve ao fato de nenhuma notação possibilitar a representação de tais informações em diagramas de casos de uso, diagramas de classes e diagramas de componentes com elementos da UML. Assim, nas notações existentes na literatura, tais informações ficam omitidas nos diagramas da UML, o que prejudicava a correta interpretação da resolução dos pontos de variação e de suas variantes. Além disso, por ser um elemento pertinente à UML e ser representado por um meta-modelo (OMG, 2005), as notas, assim como os outros elementos da UML, podem ser lidas por qualquer ferramenta de modelagem baseada nos meta-modelos da UML.

A Figura 30 apresenta um exemplo de identificação de variabilidades em diagrama de classes para um processo de compra via Internet.

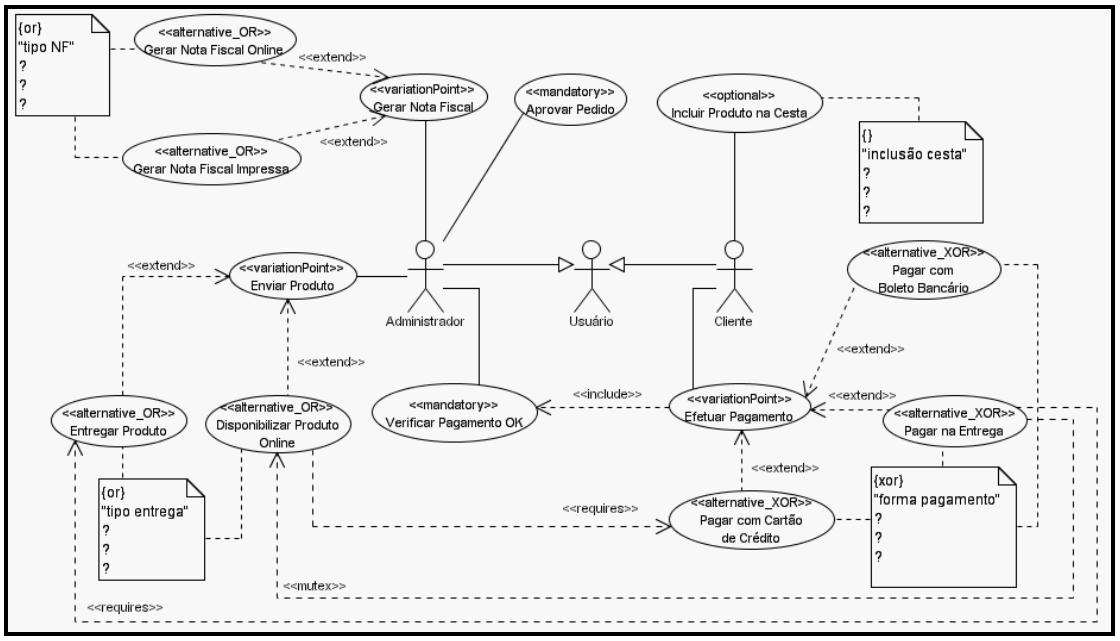

Figura 29 - Identificação de variabilidades em modelo de casos de uso de um processo de compra via Internet.

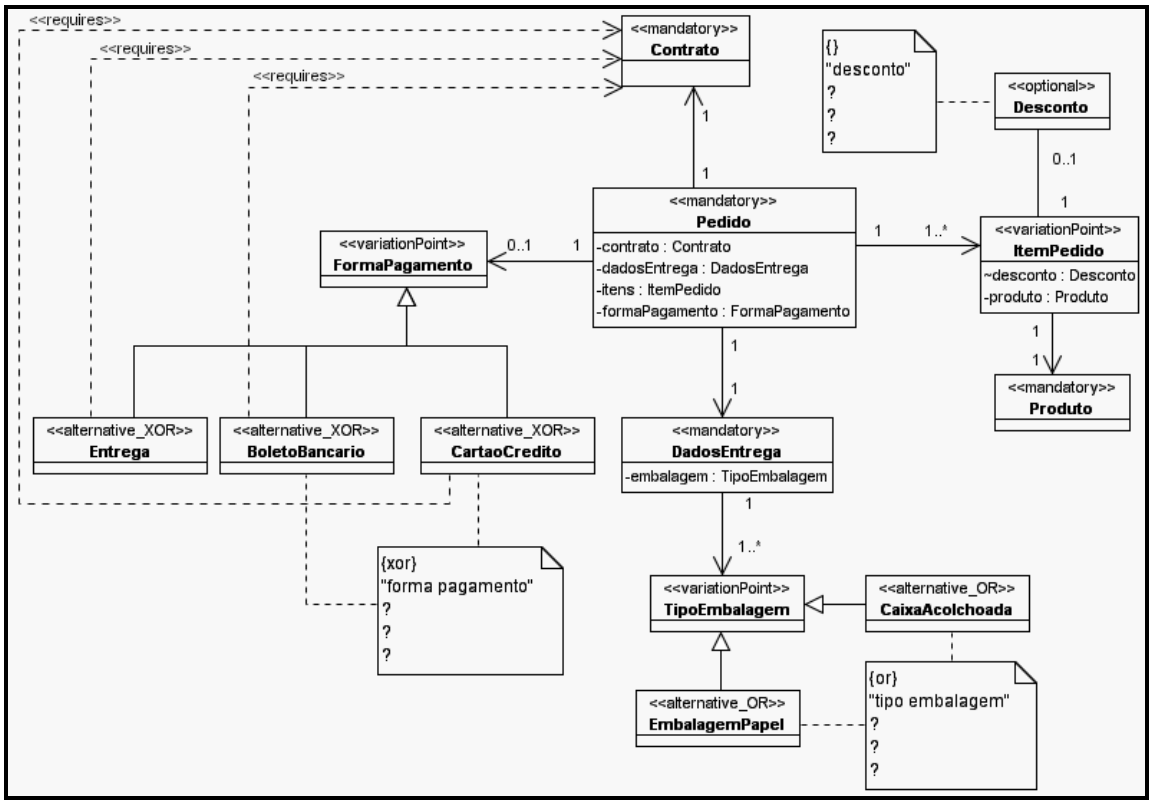

Figura 30 - Identificação de variabilidades em diagrama de classes de um processo de compra via Internet.

Assim como em casos de uso, a representação de variabilidade em classes é feita com a utilização dos mesmos estereótipos. No trabalho de Gomaa (2005), o estereótipo utilizado para representar variantes alternativas é <<variant>>, porém, assim como em casos de uso, não é possível saber se a relação entre as classes é do tipo inclusiva ou exclusiva. Dessa forma, para representar relações alternativas inclusivas e exclusivas entre classes, são utilizados, nesse trabalho, os estereótipos <<alternative_OR>> e <<alternative_XOR>>, respectivamente.

Na Figura 30, pode-se observar, por exemplo, que a classe "ItemPedido" representa um ponto de variação e as classes "Desconto" e "Produto" são variantes do tipo opcional e obrigatória, respectivamente.

A representação de variabilidade em diagramas de componentes utiliza os mesmos estereótipos utilizados em casos de uso e classes, porém o estereótipo <<variationPoint>> é substituído pelo estereótipo <<variable>>, uma vez que um componente é formado por classes que possuem pontos de variação. Assim, diz-se que o componente é variável ao invés de dizer que o componente é um ponto de variação.

A Figura 31 apresenta um exemplo de identificação de variabilidades em diagrama de componentes para um processo de compra via Internet. Observa-se que o componente "GerenciadorEnvio", com o estereótipo <<variable>>, possui variação e existe uma relação do tipo alternativa inclusiva, representada pelo estereótipo <<alternative_OR>>, entre este componente e as suas variantes "GerenciadorEntregaOnline" e "GerenciadorEntregaDomicílio".

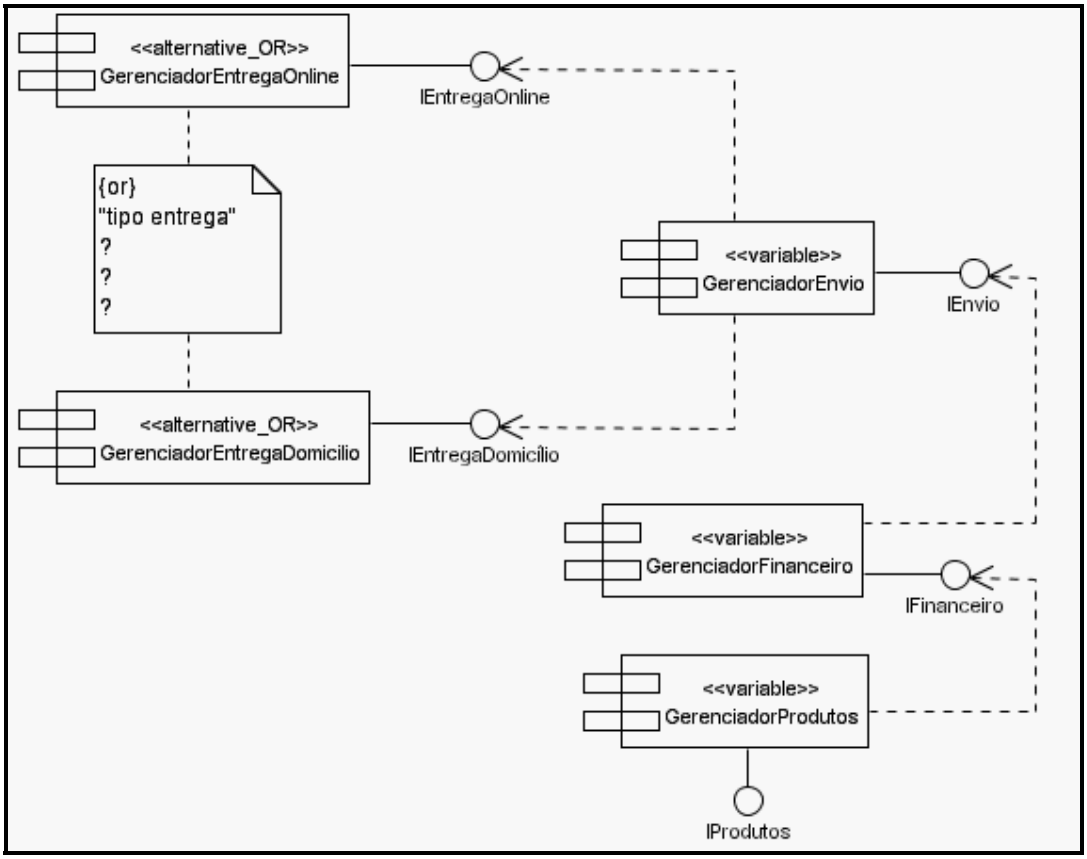

Figura 31 – Identificação de variabilidades em diagramas de componentes de um processo de compra via Internet.

3.3.3 Delimitação das Variabilidades

Esta atividade tem como objetivo definir:

- a multiplicidade de cada ponto de variação;
- o tempo de resolução de cada ponto de variação;
- se os pontos de variação podem aceitar a adição de mais variantes.

Os artefatos de entrada para esta atividade são diagramas de casos de uso, diagramas de classes e diagramas de componentes com variabilidades identificadas.

As seções a seguir apresentam as etapas desta atividade.

3.3.3.1. Definição da Multiplicidade dos Pontos de Variação

A multiplicidade de um ponto de variação indica quantas variantes do conjunto associado ao ponto de variação, no mínimo, devem ser escolhidas. A definição da multiplicidade deve considerar o tipo de relação existente entre o ponto de variação e as suas variantes.

Pontos de variação com **relações do tipo opcional** sempre terão a sua multiplicidade definida como 0 (zero), uma vez que a variante pode ou não ser escolhida.

Pontos de variação com **relações do tipo obrigatória** sempre terão suas multiplicidades definidas como 1 (um).

Assim como as relações obrigatórias, os pontos de variação com **relações do tipo alternativa exclusiva** sempre terão sua multiplicidade definida como 1 (um), pois, dentre as possíveis variantes, uma e somente uma deverá ser escolhida.

Pontos de variação com **relações do tipo alternativa inclusiva** podem variar sua multiplicidade de 0 (zero) até o número de variantes associadas ao ponto de variação, pois 0 (zero) ou mais variantes podem ser escolhidas.

O Quadro 3 apresenta um resumo da multiplicidade de acordo com o tipo de relação existente entre pontos de variação e suas variantes.

	Obrigatória	**Opcional**	**Alternativa Inclusiva**	**Alternativa Exclusiva**
Relação entre Ponto de Variação e Variante(s)	1:1	1:1	1:N	1:N
Multiplicidade	1	0	0 .. N	1

Quadro 3 - Resumo relação/multiplicidade para pontos de variação e variantes.

3.3.3.2. Definição do Tempo de Resolução dos Pontos de Variação

O tempo de resolução de um ponto de variação indica em que momento as variantes do conjunto associado ao ponto de variação serão escolhidas.

Para tanto, deve-se considerar os seguintes tempos de resolução:

- **projeto -** o ponto de variação é resolvido durante o desenvolvimento da LP ou dos produtos;
- **implementação -** o ponto de variação é resolvido em tempo de programação, antes da compilação;
- **compilação -** o ponto de variação é resolvido durante a compilação;
- **ligação -** o ponto de variação é resolvido durante a ligação de módulos ou de bibliotecas;
- **execução -** o ponto de variação é resolvido durante a execução do programa;
- **atualização -** o ponto de variação é resolvido durante a atualização do programa, o que acontece durante a sua execução. Um exemplo, seria incorporar um novo módulo disponível durante a execução do programa.

A definição do tempo de resolução dos pontos de variação é fundamental na escolha dos mecanismos de implementação de variabilidades (Seção 3.3.4).

3.3.3.3. Representação Gráfica de Multiplicidade e Tempo de Resolução

A representação gráfica da multiplicidade e do tempo de resolução de pontos de variação é realizada nos diagramas de casos de uso, diagramas de classes e diagramas de componentes.

Para isto, utiliza-se a nota da UML, presente nos diagramas provenientes da atividade anterior. Nesses diagramas, as notas da UML contêm o tipo da relação existente ({}, {or} ou {xor}), o nome da variação e três "?". Os símbolos "?" são utilizados nesta atividade para indicar, respectivamente, a multiplicidade do ponto de variação, o tempo de resolução do ponto de variação e se o ponto de variação permite a adição de mais variantes (*true* ou *false*).

A Figura 32 apresenta a evolução do modelo de casos de uso da Figura 29, incluindo a representação gráfica de multiplicidade e tempo de resolução dos pontos de variação.

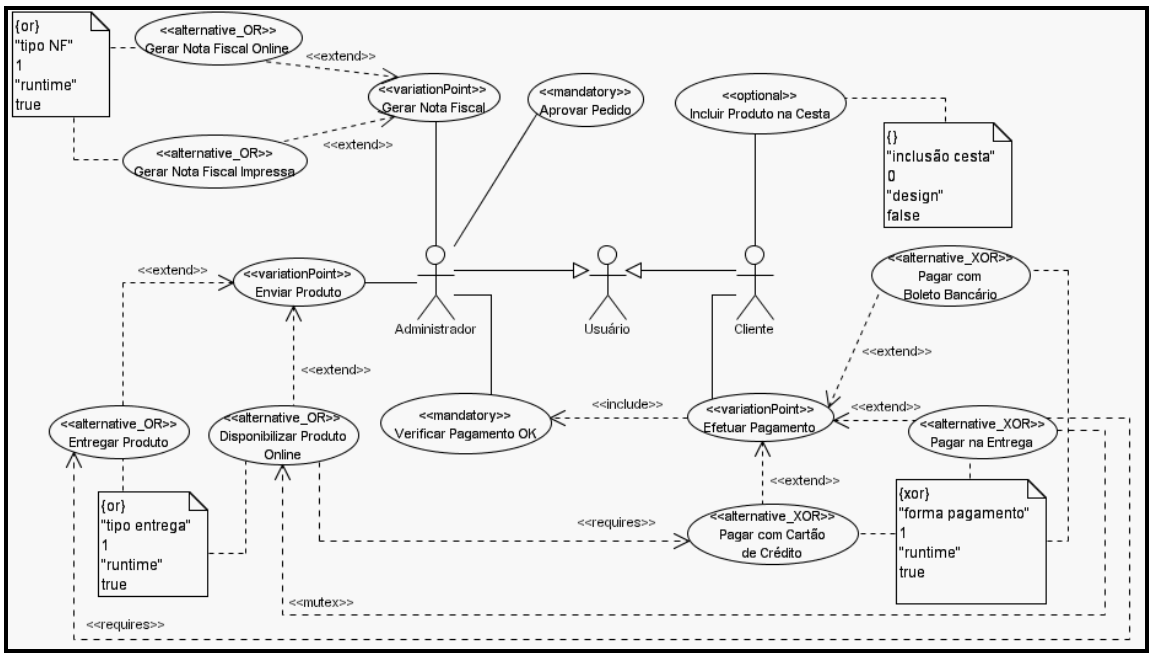

Figura 32 - Representação de multiplicidade e tempo de resolução de pontos de variação em diagrama de casos de uso de um processo de compra via Internet.

Na Figura 32, podemos notar que o caso de uso “Efetuar Pagamento” é um ponto de variação e os casos de uso “Pagar com Boleto Bancário”, “Pagar com Cartão de Crédito” e “Pagar na Entrega” são as suas variantes. Para este ponto de variação, a multiplicidade definida, de acordo com o Quadro 3, é 1 (um). O tempo de resolução definido é execução (“*runtime*”) e, ainda, o ponto de variação aceita que outras formas de pagamento sejam adicionadas durante a execução do programa.

A Figura 33 apresenta a evolução do diagrama de classes da Figura 30 para um processo de compra via Internet com a representação de multiplicidade e do tempo de resolução dos pontos de variação.

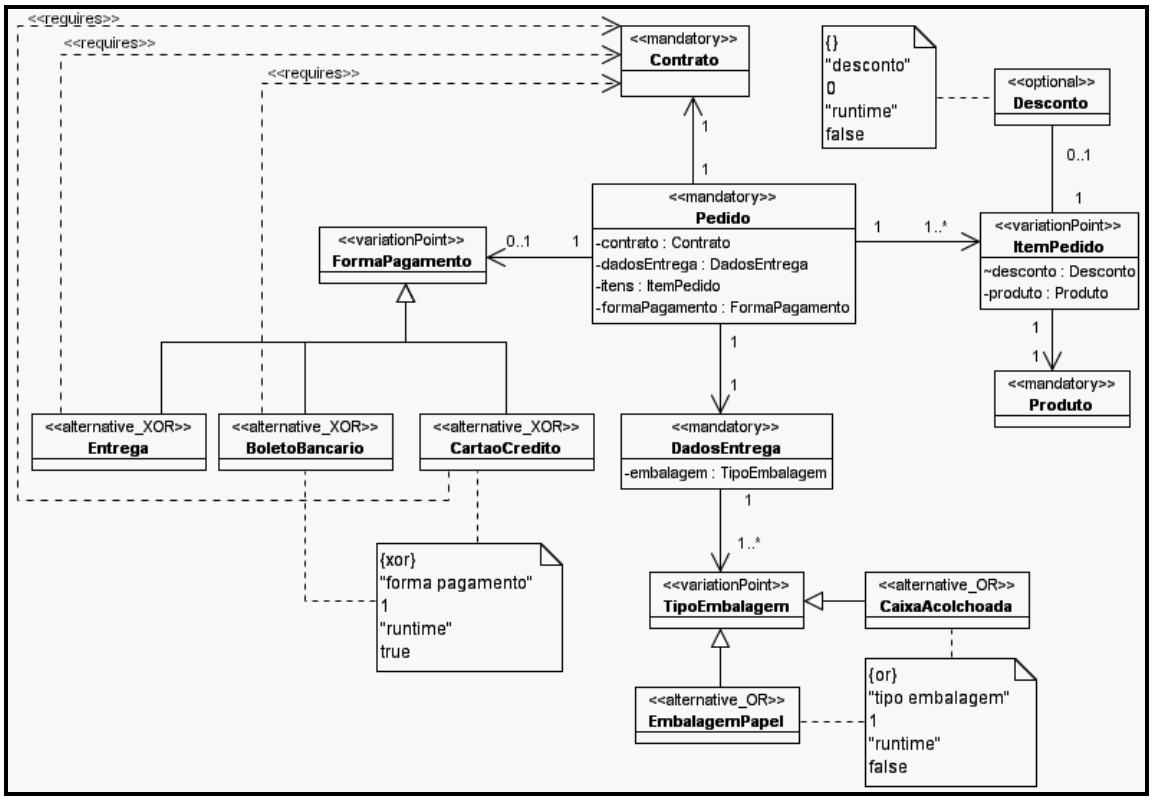

Figura 33 - Representação de multiplicidade e tempo de resolução de pontos de variação em diagrama de classes de um processo de compra via Internet.

Na Figura 33, pode-se observar que a classe "ItemPedido" representa um ponto de variação e a classe "Desconto" representa a sua variante do tipo opcional. Para este ponto de variação, a multiplicidade foi definida como 0 (zero), de acordo com o Quadro 3, e o seu tempo de resolução foi definido como sendo execução ("*runtime*"). Isto indica que a variante "Desconto" pode ou não estar presente no programa e esta decisão será tomada em tempo de execução do programa. Ainda, considerando a variação "desconto" (nota da UML), o ponto de variação não permite a adição de mais variantes, pois foi definido como "*false*".

A Figura 34 apresenta a evolução do diagrama de componentes da Figura 31 para um processo de compra via Internet com a representação de multiplicidade e tempo de resolução dos pontos de variação.

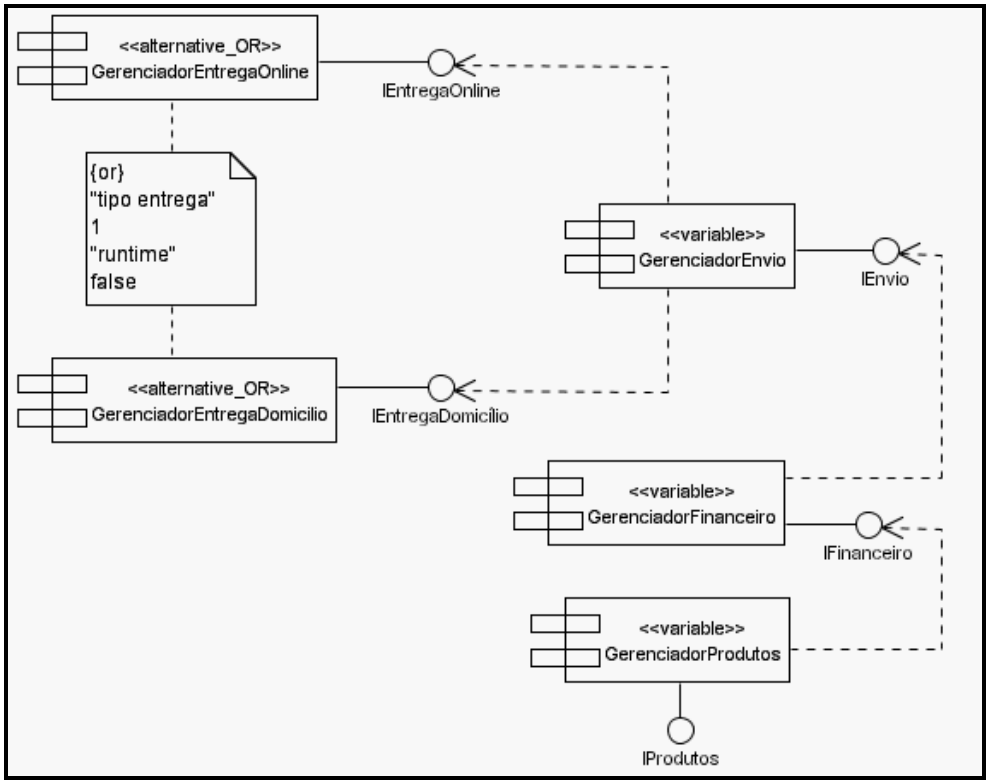

Figura 34 - Representação de multiplicidade e tempo de resolução de pontos de variação em diagrama de componentes de um processo de compra via Internet.

Na Figura 34, o componente "GerenciadorEnvio" representa um ponto de variação em que a sua multiplicidade foi definida como 1, de acordo com o Quadro 3, e o seu tempo de resolução foi definido como execução ("*runtime*"). Isto indica que as suas variantes, componentes "GerenciadorEntregaOnline" e "GerenciadorEntregaDomicilio" serão escolhidas durante a execução do programa e que nenhuma outra variante (componente) pode ser adicionada à variação "tipo entrega" (nota da UML).

3.3.4 Escolha dos Mecanismos de Implementação de Variabilidade

Esta atividade tem como objetivo escolher os mecanismos que serão utilizados para implementar as variabilidades de uma LP. Estes mecanismos devem ser definidos no que diz respeito às classes e componentes.

Os artefatos que servem de entrada para esta atividade são os diagramas de classes e os diagramas de componentes com suas variabilidades representadas e delimitadas.

Um modelo de implementação de variabilidades é gerado como saída desta atividade. Este modelo é construído com base nas técnicas de implementação de variabilidades propostas por Svahnberg, Van Gurp e Bosch (2002). Estas técnicas se baseiam nos mecanismos de implementação de variabilidade identificados por Jacobson, Griss e Jonsson (1997) e Anastasopoulos (2001) como herança, extensão, parametrização, configuração e geração. O Quadro 4 apresenta as técnicas de implementação de variabilidades de acordo com o tempo de resolução dos pontos de variação (SVAHNBERG; VAN GURP; BOSCH, 2002).

<table>
<tr><th rowspan="2">Entidades de Software Envolvidas</th><th colspan="4">Tempo de Resolução</th></tr>
<tr><th>Desenvolvimento da Arquitetura do Produto</th><th>Compilação</th><th>Ligação</th><th>Execução</th></tr>
<tr><td rowspan="3">Componentes

Frameworks</td><td>Reorganização Arquitetural</td><td rowspan="3">----------</td><td rowspan="2">Substituição Binária - Diretivas de Ligação</td><td rowspan="3">Arquitetura Orientada a Infra-estrutura</td></tr>
<tr><td>Componente Variante de Arquitetura</td></tr>
<tr><td>Componente Opcional de Arquitetura</td><td>Substituição Binária - Física</td></tr>
<tr><td rowspan="2">Implementação de Frameworks

Classes</td><td>Especializações de Componente Variantes</td><td rowspan="2">Superposição de Fragmento de Código</td><td>Substituição Binária - Diretivas de Ligação</td><td>Especializações de Componentes Variantes de Tempo de Execução</td></tr>
<tr><td>Especializações de Componente Opcionais</td><td>Substituição Binária - Física</td><td>Implementações de Componentes Variantes</td></tr>
<tr><td rowspan="2">Linhas de Código</td><td rowspan="2">----------</td><td>Condição em Constante</td><td rowspan="2">----------</td><td></td></tr>
<tr><td>Superposição de Fragmento de Código</td><td>Condição em Variável</td></tr>
</table>

Quadro 4 - Técnicas de implementação de variabilidades de acordo o tempo de resolução
Fonte: Svahnberg, Van Gurp e Bosch (2002, p.8).

O Quadro 5 apresenta um resumo de cada uma das técnicas de implementação de variabilidades, apresentadas por Svahnberg, Van Gurp e Bosch (2002).

Técnica	Tempo de Introdução	Adição de Variantes	Conjunto de Variantes	Tempo de Resolução	Funcionalidade para Resolução
Reorganização Arquitetural	Projeto Arquitetural	Projeto Arquitetural	mplícito	Desenvolvimento da Arquitetura do Produto	Externa
Componente Variável de Arquitetura	Projeto Arquitetural	Projeto Detalhado da Arquitetura	Implícito	Desenvolvimento da Arquitetura do Produto	Externa
Componente Opcional de Arquitetura	Projeto Arquitetural	Projeto Arquitetural	Implícito	Desenvolvimento da Arquitetura do Produto	Externa
Substituição Binária - Diretivas de Ligação	Projeto Arquitetural	Ligação	Implícito ou Explícito	Ligação	Externa ou Interna
Substituição Binária - Física	Projeto Arquitetural	Após a Compilação	Implícito	Antes da Execução	Externa
Arquitetura Orientada à Infra-estrutura	Projeto Arquitetural	Projeto Arquitetural Ligação Execução	Implícito ou Explícito	Compilação Execução	Interna
Especializações de Componentes Variantes	Projeto Detalhado	Projeto Detalhado	Implícito	Desenvolvimento da Arquitetura do Produto	Externa
Especializações de Componentes Opcionais	Projeto Detalhado	Projeto Detalhado	Implícito	Desenvolvimento da Arquitetura do Produto	Externa
Especializações de Componentes Variantes de Tempo de Execução	Projeto Detalhado	Projeto Detalhado	Explícito	Execução	Interna
Implementações de Componentes Variantes	Projeto Arquitetural	Projeto Detalhado	Explícito	Execução	Interna
Condição em Constante	Implementação	Implementação	Implícito	Compilação	Interna ou Externa
Condição em Variável	Implementação	Implementação	Implícito ou Explícito	Execução	Interna
Superposição de Fragmento de Código	Compilação	Compilação	Implícito	Compilação Execução	Externa

Quadro 5 – Resumo das técnicas de implementação de variabilidade

Fonte: Svahnberg, Van Gurp e Bosch (2002, p.17).

As técnicas apresentadas no Quadro 4 e no Quadro 5 levam em consideração os aspectos das variantes associadas a um ponto de variação como número de entidades

envolvidas, quando a variante deve ser introduzida, quando é possível adicionar novas variantes e quando é necessário resolver um determinado ponto de variação.

Com base nos diagramas de classes, diagramas de componentes, no Quadro 4 e no Quadro 5 deve-se construir o modelo de implementação de variabilidades. Esse modelo mostra a variação nos artefatos da LP, bem como o nível onde se encontram tais variações, o seu tempo de resolução, o mecanismo de implementação escolhido e a estratégia de implementação de acordo com o mecanismo escolhido.

O Quadro 6 apresenta o modelo de implementação de variabilidades para um processo de compra via Internet.

Variação	Nível	Tempo de Resolução	Mecanismo de Implementação	Estratégia de Implementação
tipo entrega	Componente	Projeto	Componente Opcional de Arquitetura	Duas formas: a) **componente que chama**: delegar ao mecanismo introduzido em fases posteriores; b) **componente chamado**: criar um componente nulo que responde com valores que o componente que chama deve ignorar.
forma pagamento	Classe	Execução	Especializações de Componentes Variantes em Tempo de Execução	Utilizar os *patterns Strategy* e *Template*
tipo embalagem	Classe	Execução	Especializações de Componentes Variantes em Tempo de Execução	Utilizar os padrões de projeto *Strategy* e *Template*
desconto	Classe	Execução	Especializações de Componentes Variantes em Tempo de Execução	Utilizar os padrões de projeto *Strategy* e *Template*

Quadro 6 - Modelo de implementação de variabilidades de um processo de compra via Internet.

3.3.5 Rastreamento e Controle das Variabilidades

Esta atividade visa rastrear e controlar as variabilidades identificadas em uma LP. Para isto, utiliza-se um modelo de meta-dados para o processo. Esse modelo permite que os

artefatos de uma LP possam ser relacionados, possibilitando, assim, que se faça o rastreamento e o controle das variabilidades a partir de qualquer artefato da LP que possua variação.

A Figura 35 apresenta o modelo de meta-dados do processo para o rastreamento de variabilidades. Esse modelo tem como base o modelo de variabilidade proposto por Becker (2003) e serve também como um modelo de apoio à construção de ferramentas de suporte ao gerenciamento de variabilidades em LP, fundamental à próxima atividade.

Na Figura 36, pode-se perceber, por exemplo, que uma variabilidade (classe "*Variability*") é descrita por um ou mais pontos de variação (classe "*VariationPoint*"). Cada ponto de variação possui zero ou mais variantes (classe "*Variant*"), sendo que ambas as classes são especializações da classe "*VariantArtifact*". Ainda, um ponto de variação possui um tempo de resolução (classe "*ResolutionTime*") associado e um ou mais mecanismos de implementação (classe "*ImplementationMechanism*").

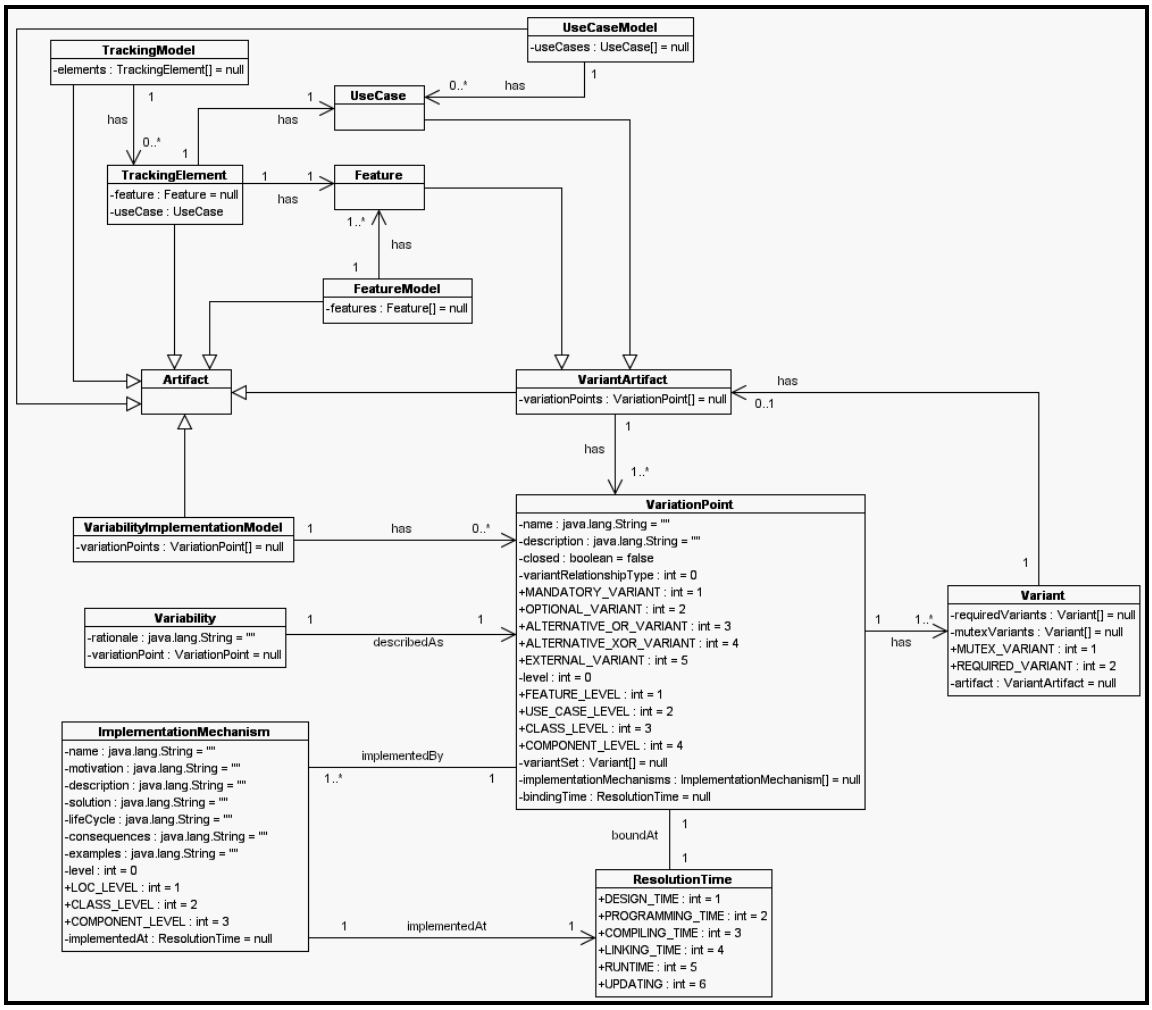

Figura 35 - Modelo de meta-dados do processo de gerenciamento de variabilidade para LP.

3.3.6 Análise de Configurações de Produtos Específicos

Esta atividade visa à especificação de diferentes produtos de uma LP com o objetivo de verificar se os artefatos da LP necessitam ser modificados ou se são identificadas novas variabilidades. Entende-se por configuração de um produto a especificação de quais *features* são escolhidas para o produto.

A partir do modelo de *features* de uma LP, criam-se diferentes configurações de produtos para a LP escolhendo quais as *features* necessárias para tais produtos. Assim, para cada produto especificado, deve-se analisar o impacto na adição, modificação ou remoção de

cada *feature* com relação aos demais artefatos da LP. Isto é possível utilizando-se o modelo de meta-dados do processo e o modelo de rastreamento de variabilidades.

Dessa forma, é possível comparar os produtos e verificar se a LP precisa ser modificada para atender à exigência dos novos produtos. Além disso, é possível, a partir das especificações dos produtos e dos dados de cada uma delas, realizar análises estatísticas, bem como realizar experimentos formais.

Porém, para alcançar o resultado esperado desta atividade, é fundamental a existência de uma ferramenta automatizada de suporte ao gerenciamento de variabilidades que permita realizar as especificações de uma LP e dos seus produtos. O projeto e a implementação de uma ferramenta para apoiar esta atividade estão fora do contexto deste trabalho, podendo ser realizados como um trabalho futuro.

3.4 CONSIDERAÇÕES FINAIS

A proposta do processo de gerenciamento de variabilidade permitiu contribuir com o processo de LP existente com relação à incorporação de atividades no desenvolvimento de LP. Tais atividades não haviam sido consideradas em trabalhos anteriores, mas se mostraram fundamentais para o gerenciamento de variabilidade.

O processo proposto neste trabalho possui atividades que, em conjunto, apóiam a construção de uma LP, dão suporte à geração de produtos específicos e fornecem uma visão gerencial de uma LP. Acredita-se, assim, que a estrutura do processo é genérica o suficiente para ser adotada, independente da abordagem de LP que esteja sendo seguida.

CAPÍTULO 4

AVALIAÇÃO DO PROCESSO PROPOSTO

Este capítulo apresenta a avaliação do processo de gerenciamento de variabilidade para LP proposto neste trabalho de mestrado. Essa avaliação baseia-se nos conceitos de engenharia de software experimental e em estudos sobre a avaliação empírica de ferramentas e métodos de engenharia de software.

As seções seguintes apresentam o estudo de caso realizado, precedido por um resumo dos fundamentos de engenharia de software experimental utilizados.

4.1 ENGENHARIA DE SOFTWARE EXPERIMENTAL

Um experimento se caracteriza por possibilitar a verificação de teorias, explorar os fatores críticos e permitir que novas teorias possam ser formuladas e corrigidas (TRAVASSOS; GUROV; AMARAL, 2002). A experimentação segue um modo sistemático, disciplinado, computável e controlado para avaliação da atividade humana (ZELKOWITZ; WALLACE, 1998).

Segundo Travassos, Gurov e Amaral (2002), novos métodos, técnicas, linguagens e ferramentas não deveriam ser apenas sugeridos e publicados sem experimentação.

Para a realização de estudos experimentais, deve-se estar familiarizado com a terminologia associada aos estudos experimentais, bem como familiarizado com os métodos experimentais existentes. Além disso, faz-se necessário um bom planejamento para que o estudo possa gerar os resultados esperados.

As seções seguintes apresentam a terminologia associada aos estudos experimentais, os métodos de avaliação experimental existentes e o planejamento de um estudo experimental.

4.1.1 Terminologia Associada aos Estudos Experimentais

Segundo Kitchenham, Pickard e Pfleeger (1995) e Kitchenham (1996), existe uma terminologia associada aos estudos experimentais, independente da estratégia experimental a ser utilizada. Tal terminologia consiste em:

- **Tratamento:** método ou ferramenta que está sendo avaliada;
- **Controle:** método ou ferramenta atualmente utilizada ou com a qual o tratamento será comparado;
- **Objeto Experimental:** é o elemento no qual o tratamento está sendo aplicado;
- **Sujeito:** indivíduo ou equipe que aplica o tratamento;
- ***Baseline*:** descreve os dados médios de projetos de uma organização e é utilizada como referência, com a finalidade de comparação com outros projetos da organização;
- **Hipótese:** especificação quantitativa de um estudo experimental que pode ser medida, permitindo que o estudo produza informações que possam ser medidas para testar tal hipótese;
    - **Hipótese Nula:** especifica que não há diferença entre o tratamento e o controle;
    - **Hipótese Alternativa:** especifica que existe alguma diferença significativa entre o tratamento e o controle;
- **Variável de Resposta:** é a variável que se espera que mude como resultado da aplicação do tratamento;
- **Variável de Estado:** fator que caracteriza o estudo e pode influenciar os resultados da avaliação;

- **Projeto Piloto:** projeto real utilizado em um estudo e aplicado ao objeto experimental.

Com base na terminologia apresentada, é possível entender e analisar os métodos de avaliação existentes. A seção seguinte apresenta os métodos experimentais existentes.

4.1.2 Métodos Experimentais

Segundo Kitchenham (1996), existem nove métodos experimentais, classificados em quantitativos, qualitativos e híbridos.

A seguir, é apresentada uma breve descrição de cada um dos métodos, de acordo com a sua classificação. Informações mais detalhadas são encontradas em Kitchenham (1996), Kitchenham, Pickard, Pfleeger (1995), Basili, Selby, Hutchend (1986), Pfleeger (1995), Travassos, Gurov, Amaral (2002) e Amaral (2003).

Neste trabalho, foi utilizado o estudo de caso quantitativo como método experimental.

4.1.2.1. Métodos Quantitativos

Os métodos quantitativos permitem a identificação de propriedades mensuráveis que se espera que mudem ao longo do estudo. Estes tipos de métodos podem ser organizados em três formas diferentes: estudo de caso quantitativo, *survey* quantitativo e experimento formal quantitativo (KITCHENHAM, 1996).

Os itens a seguir apresentam uma breve descrição de cada um dos métodos experimentais quantitativos (KITCHENHAM, 1996).

Estudo de Caso Quantitativo

Os estudos de caso envolvem o estudo de um método ou de uma ferramenta após estes terem sido utilizados em um projeto de software real. Isto significa que é possível ter certeza

de que os efeitos do método ou ferramenta se aplicam a uma determinada organização e estão de acordo com o tipo de projeto que está sendo considerado.

As vantagens de um estudo de caso são:

- poder ser incorporado às atividades de desenvolvimento de software;
- já fazer parte do ciclo de vida de desenvolvimento de software, se realizados em projetos reais;
- permitir determinar se os efeitos esperados se aplicam a uma determinada organização ou não.

Com relação às desvantagens, os estudos de caso se caracterizam por:

- possuirem pouca ou nenhuma capacidade de replicação, o que pode fornecer resultados inapropriados;
- não existir garantia de que resultados similares serão encontrados em outros projetos;
- não existirem normas e procedimentos para conduzi-los. Diferentes disciplinas abordam e usam o termo com diferentes objetivos.

Survey Quantitativo

Surveys são utilizados em estudos em que vários métodos ou ferramentas são utilizados em uma organização. Neste tipo de estudo, os usuários fornecem informações sobre as propriedades dos métodos ou ferramentas.

Como vários usuários fornecem tais informações, é utilizada alguma técnica estatística para avaliar as diferenças entre os métodos ou ferramentas com relação às propriedades de interesse. A grande diferença entre os *surveys* e os experimentos formais é que os *surveys* são menos controlados que os experimentos formais.

Assim, as vantagens dos *surveys* são:

- baseiam-se em experiências existentes como, por exemplo, os dados sobre um método ou ferramenta;
- confirmam que um efeito é generalizado a qualquer projeto/organização;
- usam técnicas de análise estatística.

Porém, as suas desvantagens são:

- confirmam somente a associação e não a causa desta;
- podem gerar diferenças dependendo de quem fornece as informações sobre um método ou ferramenta.

Experimento Formal Quantitativo

Os experimentos formais têm como objetivo observar uma ou mais variáveis e manter outras fixas, medindo o efeito do resultado (TRAVASSOS; GUROV, AMARAL, 2002; AMARAL, 2003).

Normalmente, esses experimentos são realizados com a utilização de recursos que permitem o acompanhamento e medição do produto que está sendo desenvolvido (em laboratório), uma vez que existe a necessidade do controle total sobre tais experimentos.

4.1.2.2. Métodos Qualitativos

Os métodos qualitativos são aplicados fazendo-se uma análise sobre as características de um método ou ferramenta. Essas características são identificadas através dos requisitos dos usuários para um determinado método ou ferramenta (KITCHENHAM, 1996).

Essa análise pode ser realizada por uma única pessoa que identifica os requisitos, mapeia-os para as características e, então, avalia os métodos ou ferramentas.

Os métodos qualitativos podem ser classificados em (KITCHENHAM, 1996):

- **Estudo de Caso Qualitativo:** análise baseada em características de um método ou ferramenta após este ter sido usado na prática ou em projetos reais;
- ***Survey* Qualitativo**: análise baseada em características de um método ou ferramenta, realizada por pessoas que possuem experiência ou estudam o método ou ferramenta. Deve ser organizado na forma de questionários sobre as características do método ou ferramenta;
- **Experimento Formal Qualitativo:** análise baseada em características de um método/ferramenta realizada por um grupo de potenciais usuários. Deve ser organizado na forma de um experimento formal, escolhendo potenciais usuários de forma aleatória;
- **Seleção Qualitativa (*screening*):** análise baseada em características de um método/ferramenta, realizada por um único indivíduo que não só determina as características a serem avaliadas e suas escalas, mas também realiza a avaliação.

4.1.2.3. Métodos Híbridos

Os métodos híbridos se caracterizam por possuírem elementos quantitativos e qualitativos. Existem dois métodos híbridos, conforme descrito a seguir (KITCHENHAM, 1996).

- **Análise Qualitativa de Efeitos:** baseia-se na opinião de um membro da equipe de desenvolvimento de software. Este método fornece uma avaliação subjetiva do efeito quantitativo de métodos/ferramentas. Para isto, deve estar disponível uma *baseline* sobre a opinião dos membros da equipe. Assim, é possível comparar os resultados obtidos com os resultados armazenados na *baseline*;

- ***Benchmark*:** é o processo de execução de testes usando métodos ou ferramentas alternativas para avaliar o desempenho de cada um deles. A seleção de um teste específico é subjetiva, porém os aspectos de desempenho medidos podem ser objetivos.

4.2 ESTUDO DE CASO

A avaliação do processo proposto neste trabalho foi realizada na forma de um estudo de caso quantitativo.

A escolha do estudo de caso como método experimental tem como base o Quadro 7 e se resume ao fato de que:

- os estudos de caso possuem controle sobre a medição, enquanto que os *surveys* não possuem;
- os estudos de caso possuem um controle da investigação em nível médio, enquanto os *surveys* possuem nível baixo;
- os estudos de caso possuem um custo médio, enquanto os experimentos formais possuem um custo alto para a sua realização.

Além disso, outros fatores contribuíram nesta escolha:

- o estudo de caso pode ser realizado por uma única pessoa;
- o estudo de caso não precisa envolver a aplicação de questionários, reduzindo o trâmite de documentos;
- o estudo de caso não requer a existência de uma *baseline* para comparar os dados obtidos.

Fator	*Survey*	Estudo de Caso	Experimento
O controle da execução	Nenhum	Nenhum	Tem
O controle da medição	Nenhum	Tem	Tem
O controle da investigação	Baixo	Médio	Alto

Facilidade da repetição	Alta	Baixa	Alta
Custo	Baixo	Médio	Alto

Quadro 7 - Comparação entre métodos experimentais mais utilizados
Fonte: Travassos, Gurov e Amaral (2002, p. 16).

Os itens a seguir apresentam a elaboração e a execução do estudo de caso realizado e a análise dos resultados obtidos na avaliação do processo proposto neste trabalho.

4.2.1 Elaboração do Estudo de Caso

A elaboração do estudo de caso seguiu a abordagem adotada por Kitchenham, Pickard e Pfleeger (1995), Pfleeger (1995) e Kitchenham (1996) para a avaliação de métodos e ferramentas de engenharia de software.

Em termos gerais, a abordagem é formada pelas seguintes etapas:

1. identificação do contexto do estudo de caso;
2. definição das hipóteses;
3. seleção do projeto piloto;
4. identificação do método de comparação;
5. redução dos efeitos de fatores de distorção;
6. planejamento do estudo de caso;
7. execução do estudo de caso;
8. análise dos resultados do estudo de caso.

As seções a seguir apresentam as etapas do estudo de caso.

4.2.1.1. Identificação do Contexto do Estudo de Caso

Esta etapa tem o objetivo de identificar a terminologia e as metas do estudo de caso, bem como a relevância e as restrições deste para a organização.

Os itens a seguir apresentam a terminologia utilizada no estudo de caso:

- **Tratamento:** processo de desenvolvimento de LP com o processo de gerenciamento de variabilidade proposto;
- **Controle:** processo de desenvolvimento de LP sem gerenciamento de variabilidade;
- **Objeto Experimental:** LP para WfMS;
- **Sujeito:** Edson Alves de Oliveira Junior;
- **Projeto Piloto:** desenvolvimento de uma LP para WfMS.

Os itens a seguir apresentam as restrições deste estudo de caso:

- **Patrocinador:** Programa de Pós-Graduação em Ciência da Computação (Universidade Estadual de Maringá);
- **Recursos disponíveis:**
  - processo de desenvolvimento de LP existente (Seção 2.3);
  - processo de gerenciamento de variabilidade para LP proposto;
  - ferramenta IBM/Rational Rose.
- **Cronograma:** no período de 01/09/2004 a 07/02/2005;
- **Importância do estudo de caso:** o estudo de caso aborda o gerenciamento de variabilidade seguindo um processo com atividades específicas para tal, com o objetivo de realizar uma comparação entre o processo de LP existente juntamente com o processo proposto (tratamento) e o processo de LP existente sem gerenciamento de variabilidade (controle) no que diz respeito ao gerenciamento de variabilidades. Pretende-se mostrar que o tratamento é capaz de identificar e representar uma quantidade maior de variabilidades do que o controle, tornando, também, a representação mais consistente e precisa.

4.2.1.2. Definição das Hipóteses

A definição das hipóteses é a maneira formal de especificar as metas do estudo de caso. Esta etapa é a mais difícil e a mais importante, pois o sucesso na realização do estudo de caso depende dela. Se as hipóteses forem definidas de forma errada, o estudo produzirá resultados inúteis.

Para se definir as hipóteses, é necessário saber:

- as metas do estudo de caso;
- o método/ferramenta que se deseja avaliar (tratamento);
- qual aspecto do método/ferramenta deve ser investigado;
- qual o efeito procurado na variável de resposta.

Para cada aspecto do método/ferramenta que deve ser investigado, são definidas duas hipóteses. Assim, os itens a seguir trazem a definição das hipóteses para este estudo de caso:

- ***Hipótese Nula (H0):*** não há diferença entre o tratamento e o controle na identificação e representação de variabilidades com relação à quantidade destas;
- ***Hipótese Alternativa (H1):*** o tratamento permite identificar e representar uma quantidade maior de variabilidades do que o controle;
- ***Efeito:*** mostrar que o tratamento permite identificar e representar uma quantidade maior de variabilidades em artefatos do que o controle para uma mesma LP;
- ***Variável de Resposta:*** número de variabilidades identificadas e representadas. Este número é representado pela quantidade de artefatos que possuem variação (pontos de variação e variantes) em diagramas de casos de uso, diagramas de classes e diagramas de componentes.

4.2.1.3. Seleção do Projeto Piloto

O tipo de projeto piloto a ser utilizado no estudo de caso deve ser representativo do tipo de projeto que uma organização normalmente desenvolve. Isto serve para garantir que os resultados da avaliação sejam aplicáveis a projetos reais.

Assim, para este estudo de caso, considera-se o domínio dos WfMS e como projeto piloto será realizado o desenvolvimento de uma LP para WfMS. Este domínio foi escolhido por ser utilizado como objeto de estudo em vários outros trabalhos realizados.

4.2.1.4. Identificação do Método de Comparação

Esta etapa tem o objetivo de identificar qual a melhor forma de se comparar os resultados de projetos em um estudo de caso.

Para isso, existem três formas de organizar o estudo de caso para facilitar a sua comparação:

1. comparar os resultados do novo método/ferramenta aos resultados de uma *baseline*;
2. se o método/ferramenta pode ser aplicado em componentes individuais de projeto, então aplica-se o método/ferramenta aleatoriamente a alguns componentes somente;
3. comparar os resultados de projetos similares quando não houver uma *baseline*.

Com base na forma de organização do estudo de caso com relação à comparação dos seus resultados, optou-se por utilizar o método de **Projetos Replicados,** que é uma especialização do método ***Sister Project*** (KITCHENHAM, 1996; ZELKOWITZ; WALLACE, 1998).

O método de Projetos Replicados se caracteriza por permitir o desenvolvimento de um projeto pela segunda vez usando um método/ferramenta diferente. A vantagem desse método

é que os projetos produzem o mesmo produto. No caso deste estudo de caso, o processo de LP existente em conjunto com o processo proposto especificará uma LP para WfMS. Esta LP já foi especificada, utilizando-se, para isso, o processo de LP existente sem o uso de um processo específico para o gerenciamento de variabilidade.

4.2.1.5. Redução dos Efeitos dos Fatores de Distorção

Esta etapa tem o objetivo de especificar os fatores de distorção de um estudo de caso.

Segundo Kitchenham (1996), existem três fatores de distorção significativos, que são:

1. **usar o estudo de caso como instrumento para aprender a utilizar o método/ferramenta:** este fator pode prejudicar o estudo se o método/ferramenta que está sendo avaliado estiver sendo utilizado por membros da equipe de avaliação, que não possuem um prévio conhecimento de tal método/ferramenta;
2. **a equipe de avaliação encontra-se demasiadamente entusiasmada ou cética:** este fator pode prejudicar a avaliação se a equipe tem como objetivo adotar uma nova versão de um método/ferramenta que já está sendo utilizada. Isto acontece quando a equipe deseja verificar novas funcionalidades ou atualizações do método/ferramenta existente;
3. **comparar diferentes tipos de projetos:** este fator pode prejudicar o estudo, pois projetos de domínios diferentes geram resultados significantemente diferentes, impossibilitando, assim, serem utilizados para a sua análise.

No estudo de caso realizado neste trabalho, nenhum destes fatores interferiu em seu andamento, uma vez que a equipe de desenvolvimento não o está utilizando para aprender sobre o processo que está sendo avaliado, não existe uma versão prévia do processo proposto

neste trabalho e o produto gerado nesse estudo pertence ao mesmo domínio do produto com o qual este é comparado.

4.2.1.6. Planejamento do Estudo de Caso

O planejamento do estudo de caso tomou como base o plano de avaliação proposto por Basili, Selby e Hutchens (1986). O plano é dividido em quatro partes, sendo elas:

1. **definição:** possui as seguintes informações sobre o estudo: a motivação do estudo, o objeto do estudo, o propósito do estudo, o domínio no qual o estudo está inserido e o seu contexto;
2. **planejamento:** possui informações sobre o tipo de projeto experimental e técnica estatística que será seguida, os critérios do estudo de caso e a forma de medição que será utilizada;
3. **operação:** possui informações sobre a preparação para a execução do estudo de caso, a execução propriamente dita e a forma de análise dos resultados obtidos;
4. **interpretação:** possui informações sobre o contexto da interpretação dos resultados obtidos, sua extrapolação e o impacto dos resultados sobre o estudo.

Os itens a seguir apresentam o plano de avaliação para o estudo de caso realizado neste trabalho.

1. **Definição:**
    a. **motivação:** o gerenciamento de variabilidade em uma LP é uma das atividades mais importantes do gerenciamento de LP. Nishimura (2004) propôs um processo para a geração de produtos específicos em LP. Este processo aborda, de forma superficial, o gerenciamento de variabilidades com base nas idéias do método KobrA (ATKINSON et

al., 2001). Para melhorar o gerenciamento de variabilidades, foi proposto um processo para tal. Assim, este estudo de caso permitirá avaliar o tratamento, comparando-o ao controle, no que diz respeito à identificação e representação de um número maior de variabilidades;

b. **objeto de estudo:** processo de LP existente com o processo de gerenciamento de variabilidade proposto;
c. **propósito:** caracterizar e avaliar as diferenças na identificação e representação de variabilidades do tratamento com relação ao controle, bem como justificar a adoção do tratamento ao processo de LP existente;
d. **perspectiva:** o estudo é realizado sob a perspectiva do gerente de LP;
e. **domínio:** o estudo é realizado pela equipe de desenvolvimento de LP;
f. **contexto:** é utilizado o método de comparação Projetos Replicados por existir um projeto realizado anteriormente.

2. **Planejamento:**
    a. **projeto experimental:** neste estudo de caso, não serão utilizadas técnicas estatísticas (BASILI; SELBY; HUTCHENS, 1986), visto que o conjunto de fatores, variáveis de resposta e métricas é pequeno e pelo estudo não ser realizado várias vezes;
    b. **critérios:** os critérios utilizados baseiam-se na variável de resposta definida juntamente com as hipóteses;
    c. **medição:** objetiva, através da variável de resposta; e subjetiva, através da interpretação dos valores desta;
3. **Operação:**

a. **preparação:** o projeto piloto é o projeto definido para o estudo de caso (Seção 4.2.1.3);
b. **execução:** segue o processo de desenvolvimento de LP junto com o processo proposto neste trabalho. A coleta de dados é realizada com base na variável de resposta do estudo de caso;
c. **análise:** realizada através de histogramas para os valores da variável de resposta obtidos pelo estudo de caso;

4. **Interpretação:**
   a. **contexto de interpretação:** a interpretação dos dados neste estudo de caso não seguirá um *framework* estatístico (BASILI; SELBY; HUTCHENS, 1986) devido ao reduzido número de variáveis de resposta e valores para estas, coletados durante a execução do estudo de caso;
   b. **extrapolação:** nenhuma estratégia de extrapolação (BASILI; SELBY; HUTCHENS, 1986) será utilizada por cauda da reduzida amostra de dados do estudo de caso;
   c. **impacto:** o impacto dos resultados é apresentado na forma de sugestões de melhorias para o processo avaliado, caso sejam necessárias.

Os itens a seguir apresentam a execução do estudo de caso e a análise dos resultados obtidos, na forma de histogramas e interpretações objetivas e subjetivas.

4.2.2 Execução do Estudo de Caso

A execução do estudo de caso seguiu o processo de desenvolvimento de LP existente, adicionando-se o processo de gerenciamento de variabilidade proposto. A interação entre os processos para a execução do estudo de caso é mostrada na Figura 26.

As seções a seguir apresentam as atividades do desenvolvimento de LP interagindo com as atividades do gerenciamento de variabilidade para a execução do estudo de caso.

4.2.2.1. Especificação de Requisitos

A especificação de requisitos é a primeira atividade do desenvolvimento da LP e é dividida em Elaboração do Modelo de Casos de Uso e Elaboração do Modelo de *Features*.

Durante a especificação de requisitos, foi elaborado um modelo representando os objetos e ações do domínio e identificando as semelhanças entre os produtos.

As funcionalidades básicas de produtos de *workflow* foram extraídas da arquitetura genérica e do modelo de referência da WfMC (WfMC, 1995).

A primeira atividade a ser realizada foi a **elaboração do modelo de casos de uso**. Esta atividade gerou como artefato de saída:

- o modelo de casos de uso do domínio (Figura 36);
- o refinamento do caso de uso "*Define Workflow Architecture*" para o ator "*Workflow Architecture Manager*" (Figura 37);
- o refinamento do caso de uso "*Execute Workflow*" para o ator "*Workflow User*" (Figura 38);
- o refinamento dos casos de uso "*Define Workflow*" e "*Manage Workflow*" para o ator "*Workflow Manager*" (Figura 39).

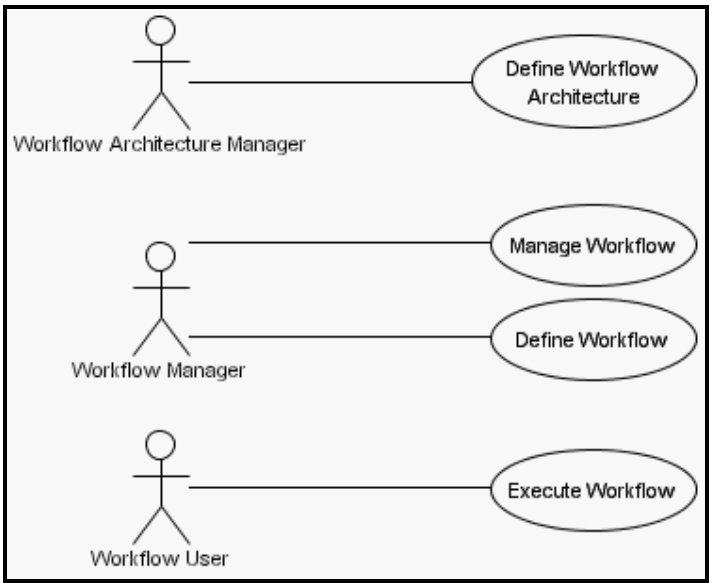

Figura 36 - Modelo de casos de uso do domínio para WfMS.

O modelo de casos de uso do domínio é formado por quatro casos de uso de negócio e três atores. Para cada um dos atores foi elaborado um diagrama de casos de uso que mostra o refinamento das ações que esses atores devem realizar para os seus casos de uso.

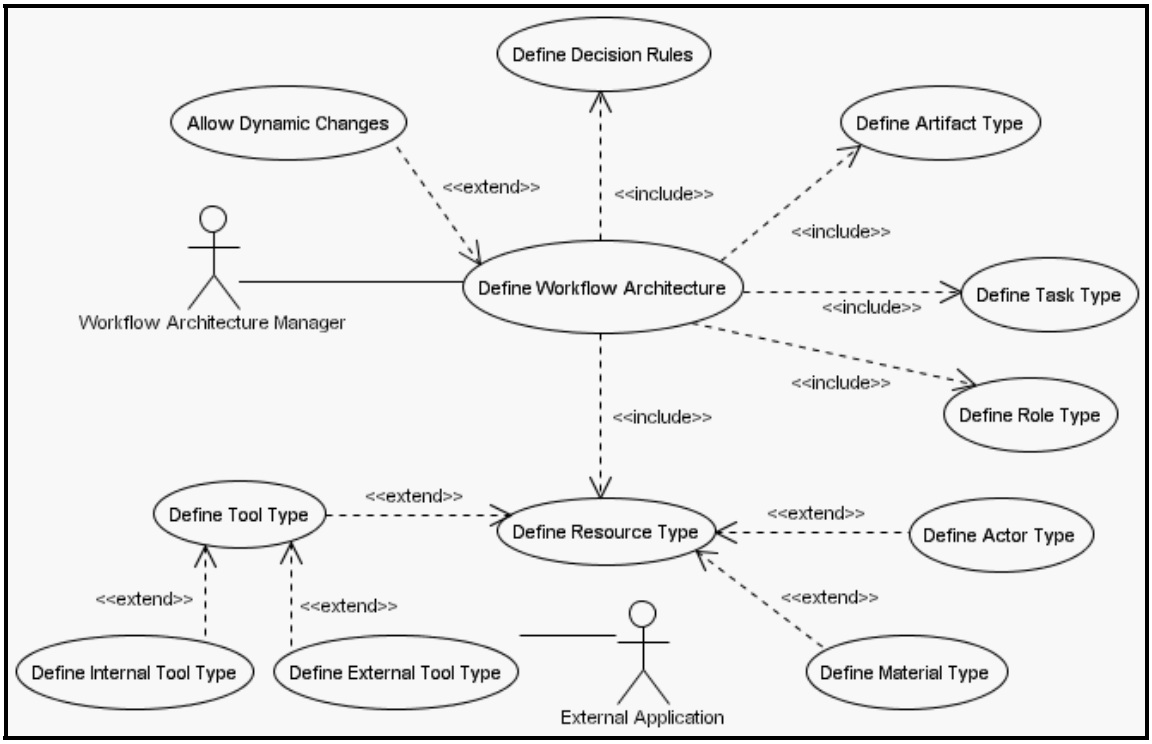

Figura 37 - Diagrama de casos de uso para o ator *Workflow Architecture Manager*.

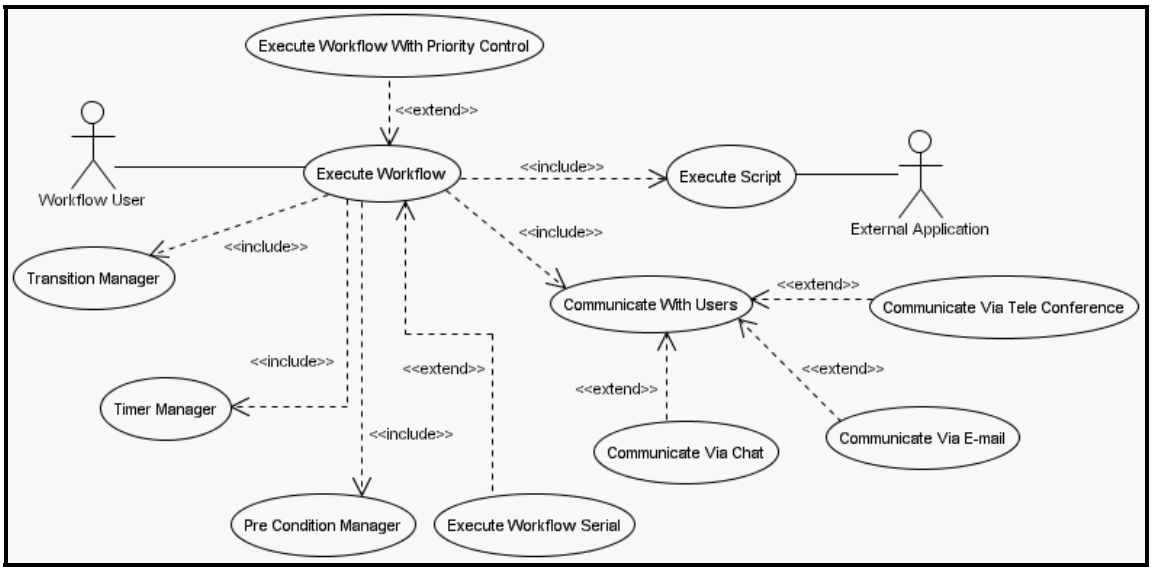

Figura 38 - Diagrama de casos de uso para o ator *Workflow User*.

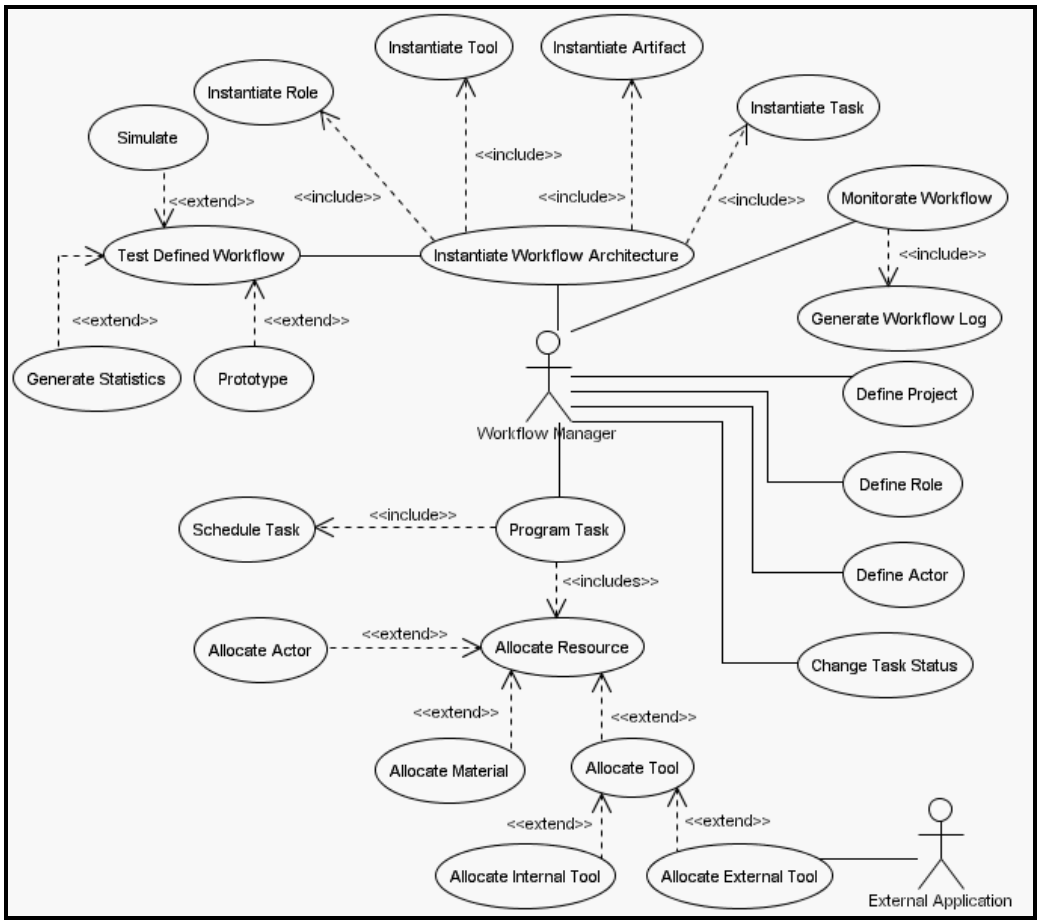

Figura 39 - Diagrama de casos de uso para o ator *Workflow Manager*.

Ao término da atividade de elaboração do modelo de casos de uso, foi realizada a primeira iteração do processo de gerenciamento de variabilidade. Nessa iteração ainda não é possível realizar a atividade de **elaboração do modelo de rastreamento de variabilidades**, pois esta exige como artefato de entrada o modelo de *features*, produzido na próxima atividade de desenvolvimento da LP.

Assim, a atividade de **identificação das variabilidades** tomou como artefatos de entrada os diagramas de casos de uso resultantes da atividade de elaboração do modelo de casos de uso. O artefato resultante da atividade de identificação das variabilidades é a evolução dos diagramas de casos de uso. Tais diagramas possuem agora as suas variabilidades identificadas.

A atividade de **delimitação das variabilidades** tomou como artefatos de entrada os diagramas de casos de uso resultantes da atividade anterior. O artefato resultante dessa atividade é a evolução dos diagramas de casos de uso com as variabilidades identificadas, porém estas, agora, apresentam-se delimitadas.

A Figura 40, a Figura 41 e a Figura 42 apresentam os diagramas de casos de uso com as variabilidades identificadas, representadas e delimitadas.

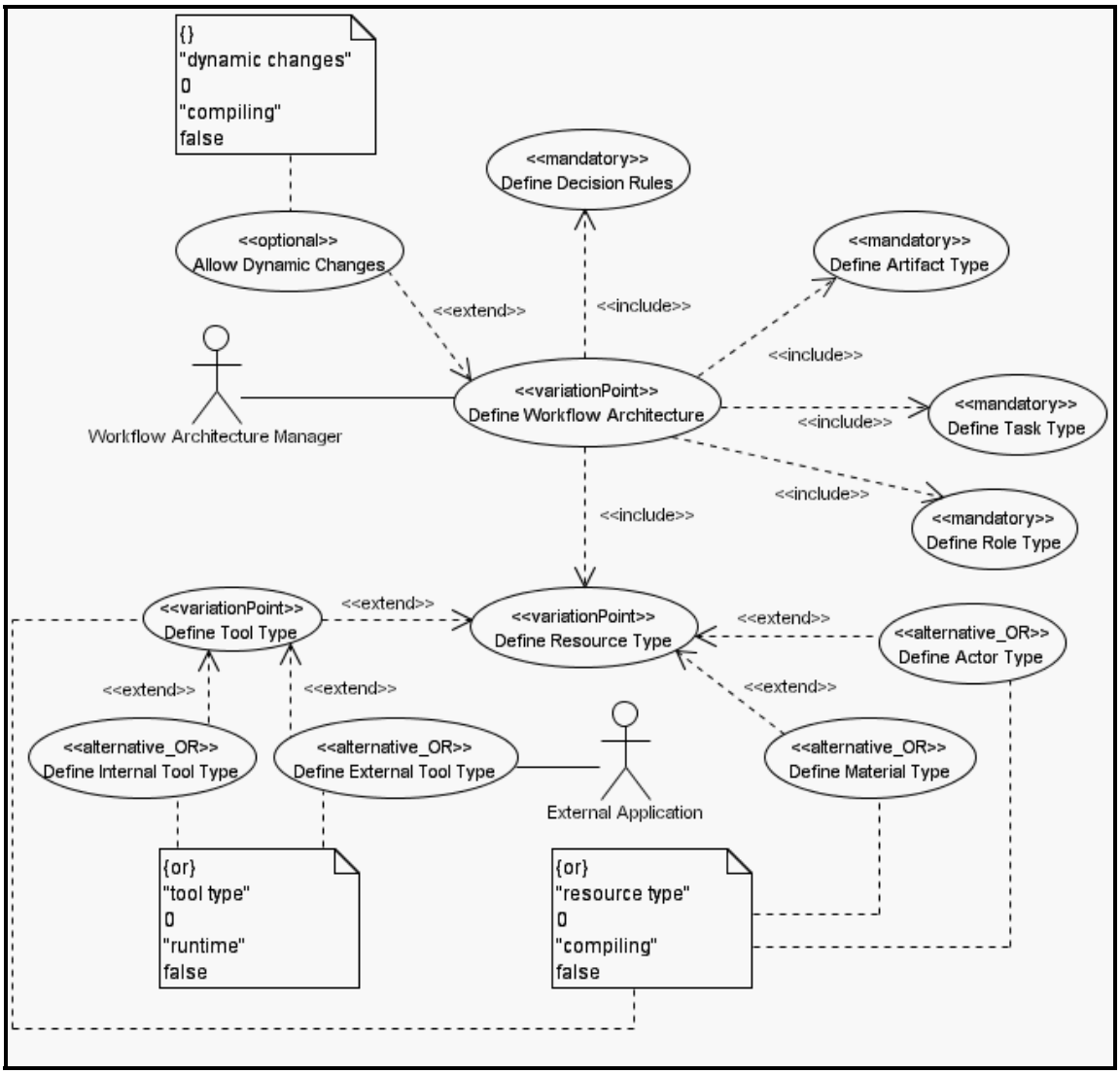

Figura 40 - Diagrama de casos de uso para o ator *Workflow Architecture Manager* com variabilidades identificadas e delimitadas.

Na Figura 40, percebe-se a existência de vários pontos de variação. Nesta figura, é interessante notar que o caso de uso "*Define Tool Type*" é, ao mesmo tempo, uma variante do tipo alternativa inclusiva do caso de uso "*Define Resource Type*" e também representa um ponto de variação com as variantes associadas "*Define Internal Tool Type*" e "*Define External Tool Type*". Assim, o caso de uso "*Define Tool Type*" possui o estereótipo <<variationPoint>>, ao invés do estereótipo <<alternative_OR>>, e está ligado à nota referente às informações sobre a variação "*resource type*".

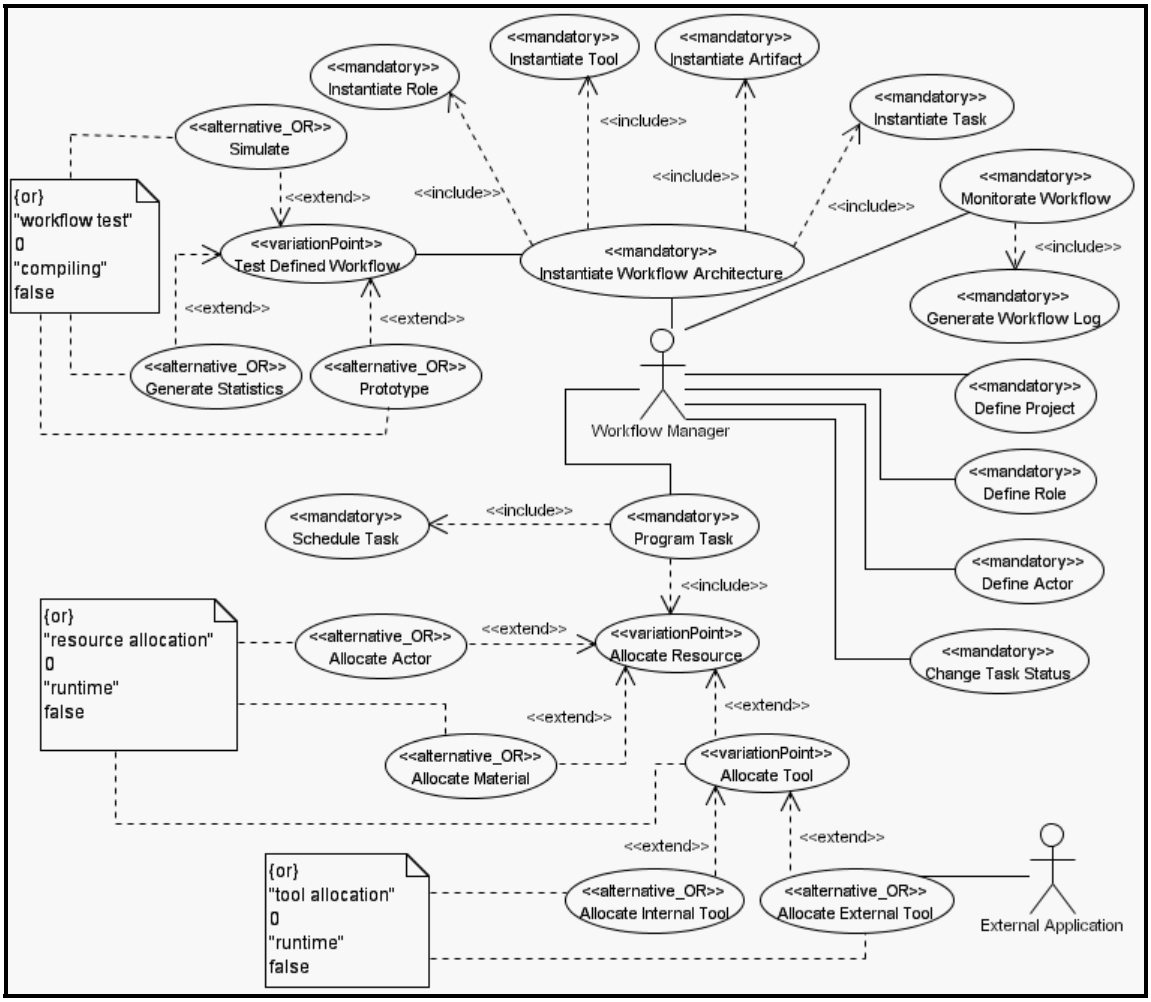

Figura 41 - Diagrama de casos de uso para o ator *Workflow Manager* com variabilidades identificadas e delimitadas.

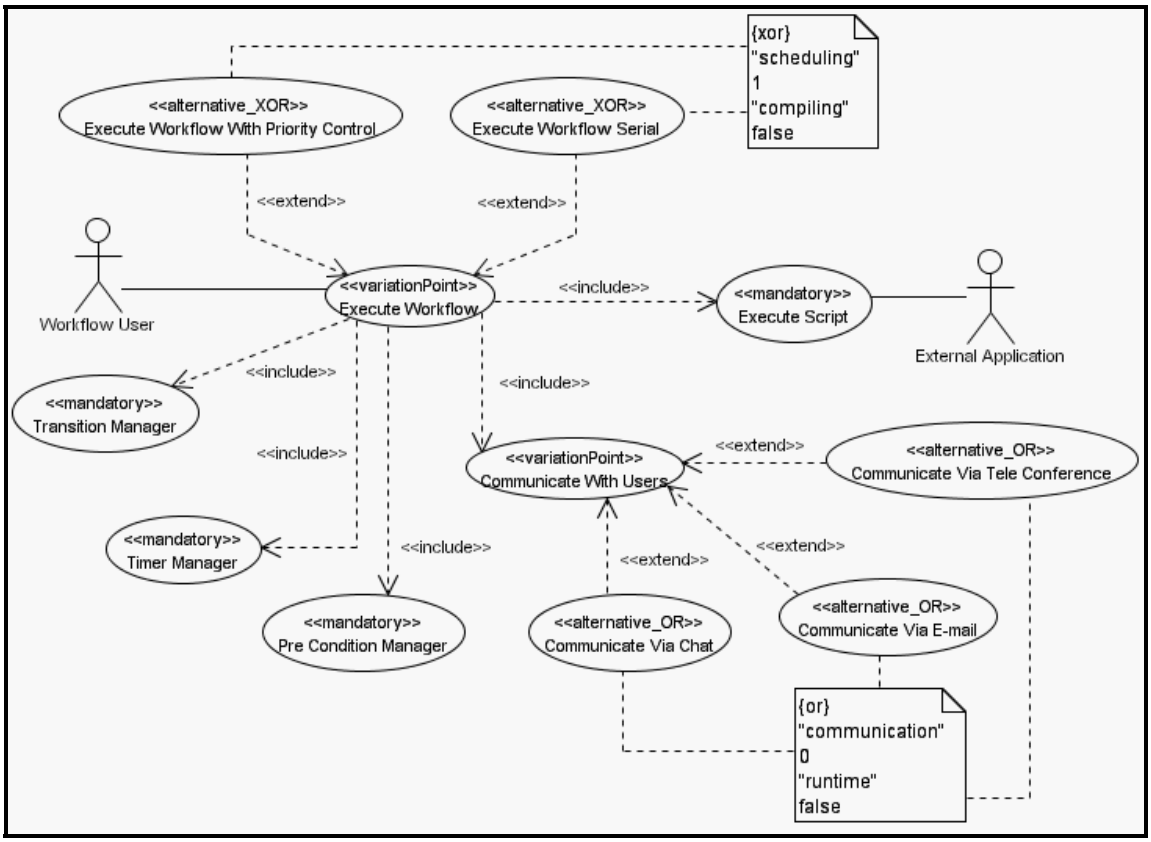

Figura 42 - Diagrama de casos de uso para o ator *WorkflowUser* com variabilidades identificadas e delimitadas.

Na Figura 42, observa-se que o caso de uso "*Execute Workflow*" possui duas variantes associadas à variação "*sheduling*", sendo elas os casos de uso "*Execute Workflow With Priority Control*" e "*Execute Workflow Serial*". Assim, segundo as relações entre pontos de variação e variantes, uma e somente uma destas variantes poderá ser escolhida para realizar o ponto de variação (relação alternativa exclusiva).

Um ponto importante a ressaltar na representação de variabilidades é que as variantes do tipo obrigatórias, com o estereótipo <<mandatory>>, não possuem notas da UML com informações sobre a variação. Isto acontece por que:

- os produtos devem obrigatoriamente conter tais variantes;
- o tempo de resolução das variantes obrigatórias é "*design*";

- a cardinalidade das variantes obrigatórias é sempre 1;
- não é possível adicionar mais variantes; não existe a possibilidade de escolher outras variantes em um conjunto formado por um único elemento.

Nessa iteração, ainda não é possível realizar a atividade de **escolha dos mecanismos de implementação de variabilidades**, pois tal atividade exige como entrada diagramas de classes e diagramas de componentes. Também, nessa iteração, não é possível realizar a atividade de **rastreamento e controle de variabilidades** e nem a atividade de **análise e configuração de produtos**, uma vez que o artefato de entrada para estas duas atividades é o modelo de rastreamento de variabilidades e este só é construído ao final da atividade de elaboração do modelo de *features*.

Para este estudo de caso, os diagramas de casos de uso são considerados aprovados. Caso não tivessem sido aprovados, estes deveriam ser modificados e uma nova iteração do processo de gerenciamento de variabilidade iniciaria.

Sendo assim, é realizada, então, a próxima atividade do desenvolvimento da LP, a elaboração do modelo de *features*.

A atividade de **elaboração do modelo de *features*** tem como artefato de entrada os diagramas de casos de uso elaborados na atividade anterior e gera como artefato de saída o modelo de *features* para WfMS.

Ao finalizar a elaboração do modelo de *features,* é iniciada uma nova iteração do processo de gerenciamento de variabilidade. Nessa iteração é realizada a atividade de **elaboração do modelo de rastreamento de variabilidades**. Essa atividade tem como artefato de entrada os diagramas de casos de uso e o modelo de *features* e gera como artefato de saída o modelo de rastreamento de variabilidades. Em seguida, são realizadas as demais atividades. Nesta iteração, ainda não é possível realizar a atividade de **escolha dos**

mecanismos de implementação de variabilidades por não existirem, ainda, os artefatos de entrada requeridos, diagramas de classes e diagrama de componentes. Nessa iteração, é possível realizar as atividades de **rastreamento e controle de variabilidades** e **análise de configurações de produtos**, pois os artefatos de entrada para tais atividades já foram elaborados.

A Figura 43 apresenta o modelo de *features* para WfMS com suas variabilidades identificadas e delimitadas. Nesta figura, nota-se que as *features* dividem-se em quatro grupos, sendo eles: "*Workflow Architecture Definition*", "*Workflow Definition*", "*Workflow Management*" e "*Workfow Execution*". Esses grupos correspondem às *features* identificadas a partir dos casos de uso da Figura 36 e dos seus respectivos refinamentos e evolução (Figura 40, Figura 41 e Figura 42).

Ainda na Figura 43, observa-se que existem várias relações de dependência entre *features* com o estereótipo <<requires>> como, por exemplo, a relação entre as *features* "*Workflow Definition*" e "*Workflow Architecture Definition*". Isto indica que um produto só poderá permitir a definição de um *workflow* se a definição da arquitetura de *workflow* também puder ser realizada no produto.

O Quadro 8 e o Quadro 9 apresentam o modelo de rastreamento de variabilidades para WfMS, sendo que o segundo é a continuação do primeiro. As *features* enumeradas no Quadro 8 e no Quadro 9 são:

1. *Workflow Architecture Definition*
2. *Decision rules*
3. *Dynamic Changes*
4. *MetaModel Definition*
5. *Resource Type*
6. *Tool Type*
7. *Internal Tool Type*
8. *External Tool Type*
9. *Material Type*
10. *Actor Type*
11. *Task Type*
12. *Role Type*
13. *Artefact Type*
14. *Workflow Definition*
15. *Workflow Architecture Instantiation*
16. *Role*
17. *Tool*
18. *Task*
19. *Artefact*
20. *Task Programming*

21. *Resource Allocation*
22. *Material*
23. *Tool*
24. *Internal tool*
25. *External Tool*
26. *Actor*
27. *Task Scheduling*
28. *Workflow Management*
29. *Workflow Testing*
30. *Simulation*
31. *Statistics*
32. *Prototyping*
33. *Project Definition*
34. *Role Definition*
35. *Actor Definition*
36. *Workflow Status Management*
37. *Workflow Monitoring*
38. *Workflow Execution*
39. *Execution Management*
40. *Transition Management*
41. *Timer Management*
42. *PreCondition Management*
43. *Execution Scheduling*
44. *Priority Control*
45. *Serial*
46. *User Communication*
47. *Chat*
48. *TeleConference*
49. *EMail*
50. *Script Execution*

Os casos de uso enumerados no Quadro 8 e no Quadro 9 são:

1. *Define Workflow Architecture*
2. *Allow Dynamic Changes*
3. *Define Decision Rules*
4. *Defie Artefact Type*
5. *Define Task Type*
6. *Defin eRole Type*
7. *Define Resource Type*
8. *Define Actor Type*
9. *Define Material Type*
10. *Define Tool Type*
11. *Define Internal Tool Type*
12. *Define External Tool Type*
13. *Instantiate Workflow Architecture*
14. *Instantiate Role*
15. *Instantiate Tool*
16. *Instantiate Artefact*
17. *Instantiate Task*
18. *Program Task*
19. *Schedule Task*
20. *Allocate Actor*
21. *Allocate Resource*
22. *Allocate Material*
23. *Allocate Tool*
24. *Allocate Internal Tool*
25. *Allocate External Tool*
26. *Test Defined Workflow*
27. *Simulate*
28. *Generate Statistics*
29. *Protoype*
30. *Monitorate Workflow*
31. *Generate Workflow Log*
32. *Define Project*
33. *Define Role*
34. *Define Actor*
35. *Change Task Status*
36. *Execute Workflow*
37. *Execute Workflow With Priority Control*
38. *Execute Workflow Serial*
39. *Execute Script*
40. *Transition Manager*
41. *Timer Manager*
42. *Pré Condition Manager*
43. *Communicate With Users*
44. *Communicate ViaTele Conference*
45. *Communicate Via EMail*
46. *Communicate Via Chat*

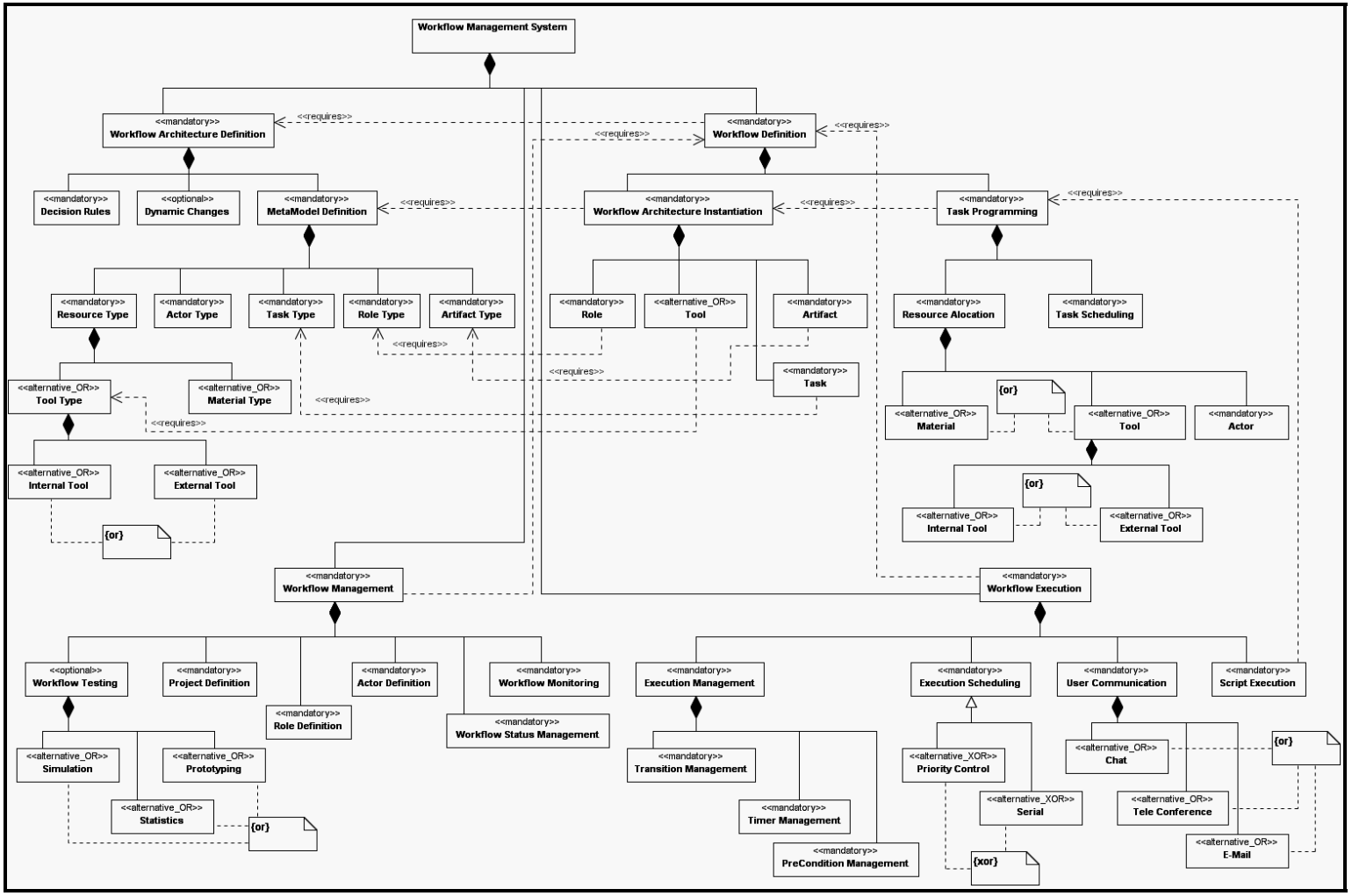

Figura 43 - Modelo de *features* para WfMS.

		CASOS DE USO																																		
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
FEATURES	1	•	•	•	•	•	•	•	•	•	•	•	•	•																						
	2			•																																
	3		•																																	
	4				•	•	•	•	•	•	•	•	•																							
	5							•		•	•	•	•																							
	6										•	•	•																							
	7											•																								
	8												•																							
	9									•																										
	10								•																											
	11					•																														
	12						•																													
	13				•																															
	14													•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	15													•	•	•	•	•																		
	16														•																					
	17															•																				
	18																	•																		
	19																•																			
	20																			•	•	•	•	•	•	•										
	21																					•	•	•	•	•										
	22																						•													
	23																							•	•	•										
	24																								•											
	25																									•										
	26																				•															
	27																			•																
	28																										•	•	•	•	•	•	•	•	•	•
	29																										•	•	•	•						
	30																											•								
	31																												•							
	32																													•						
	33																																•			
	34																																	•		
	35																																		•	
	36																																			•
	37																														•					

Quadro 8 - Modelo de rastreamento de variabilidades para WfMS.

		CASOS DE USO										
		36	37	38	39	40	41	42	43	44	45	46
FEATURES	38	●	●	●	●	●	●	●	●	●	●	●
	39					●	●					
	40					●						
	41						●					
	42							●				
	43		●	●								
	44		●									
	45			●								
	46								●	●	●	●
	47											●
	48									●		
	49										●	
	50				●							

Quadro 9 - Continuação do modelo de rastreamento de variabilidades para WfMS.

O modelo de rastreamento de variabilidades é o artefato mais importante para que se possa rastrear e controlar as variabilidades em LP e, com isso, saber exatamente quais são os elementos (casos de uso, classes e componentes) que sofrem alteração quando se adicionam ou removem *features* de um determinado produto. Isto permite a análise de configurações de produtos em atividades posteriores do processo.

4.2.2.2. Especificação do Sistema

Nesta atividade do desenvolvimento de LP, todos os elementos surgem como solução de software para a aplicação que está sendo especificada. Dentre os elementos identificados, estão os tipos que surgiram da análise das ações do sistema, representadas pelas Figura 40, Figura 41 e Figura 42. O principal artefato gerado neste estágio é o modelo estático de tipos para WfMS, apresentado na Figura 44.

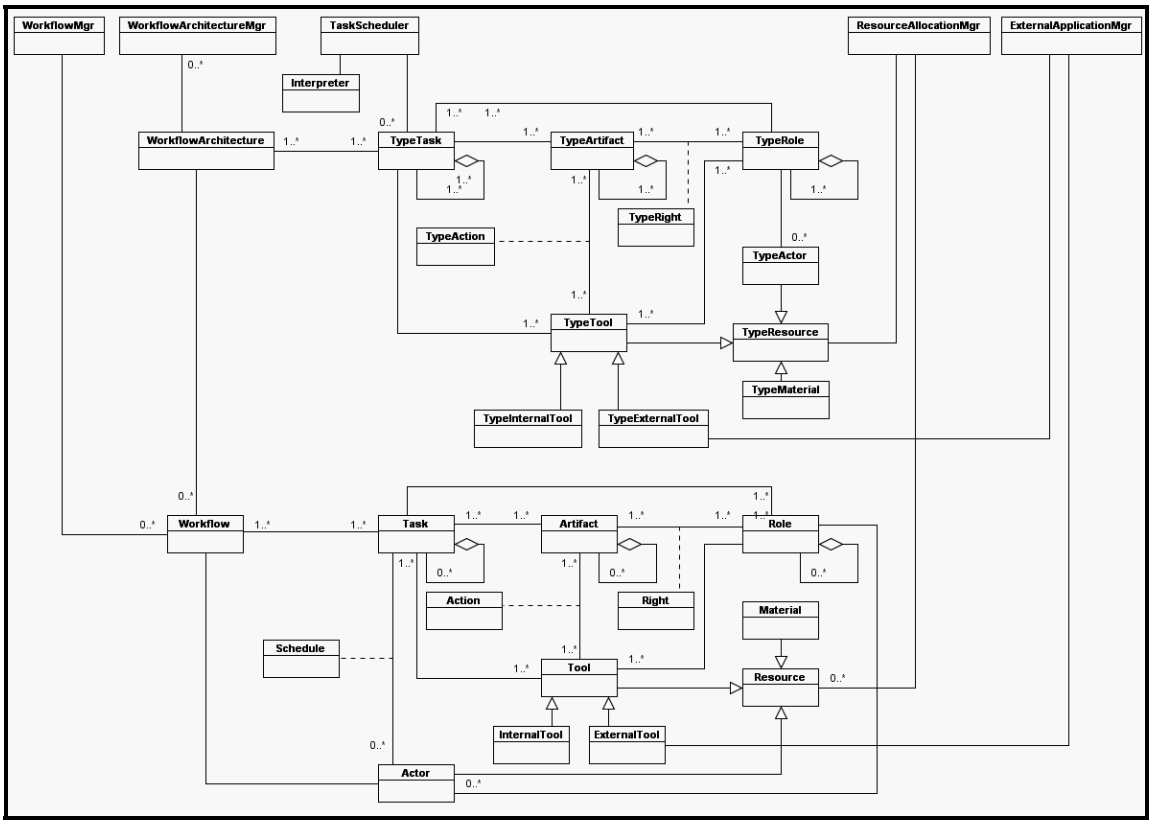

Figura 44 – Modelo estático de tipos para WfMS.

Ao término da atividade de especificação do sistema, inicia-se uma nova iteração do processo de gerenciamento de variabilidade. A partir dessa iteração não é necessário realizar a atividade de **elaboração do modelo de rastreamento de variabilidades**, pois o modelo de casos de uso e o modelo de *features* não são mais modificados.

Assim, a atividade de **identificação das variabilidades** é realizada e tem como entrada o modelo estático de tipos para WfMS (Figura 44). Como saída desta atividade, tem-se a evolução do modelo estático de tipos com suas variabilidades identificadas. Em seguida, é realizada a atividade de **delimitação das variabilidades** que tem como artefato de entrada o modelo estático de tipos com variabilidades identificadas, gerando, assim, como artefato de saída, o modelo estático de tipos da Figura 45.

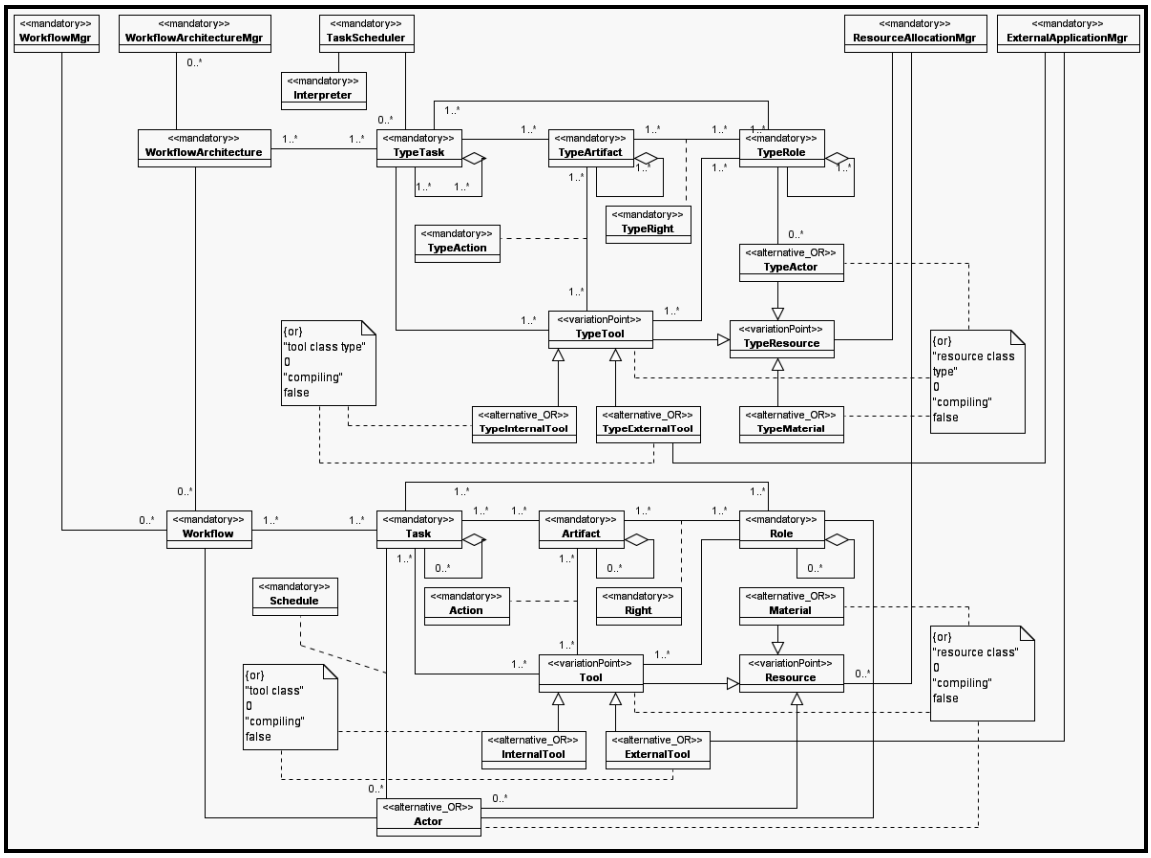

Figura 45 – Modelo estático de tipos para WfMS com variabilidades representadas e delimitadas.

Nessa iteração, já é possível realizar **a atividade de escolha dos mecanismos de implementação de variabilidades**, uma vez que foram identificadas variabilidades em nível de classe gerando como artefato de saída o modelo de implementação de variabilidades inicial para WfMS (Quadro 10).

Segundo o Quadro 4, como todas as variabilidades do modelo estático de tipos ocorrem em classes e os seus tempos de resolução são compilação, o mecanismo escolhido foi "Superposição de Fragmento de Código".

Para finalizar essa iteração, são realizadas as atividades de **rastreamento e controle de variabilidades** e **análise de configuração de produtos**.

A atividade seguinte do desenvolvimento da arquitetura de LP, Projeto Arquitetural,

não será apresentada, pois não produz nenhum tipo de artefato que permita a identificação de variabilidades para a LP para WfMS (Figura 26).

Variação	Nível	Tempo de Resolução	Mecanismo de Implementação	Estratégia de Implementação
tool class type	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.
resource class type	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.
tool class	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.
resource class	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.

Quadro 10 - Modelo de implementação de variabilidades inicial para WfMS.

4.2.2.3. Projeto Interno dos Componentes

Nessa atividade de desenvolvimento de LP, são identificados componentes que formam a arquitetura da aplicação, bem como as classes e as interfaces que pertencem a cada um dos componentes. Assim, o principal artefato gerado nesta atividade é o diagrama de componentes que representa a arquitetura de LP para WfMS (Figura 46).

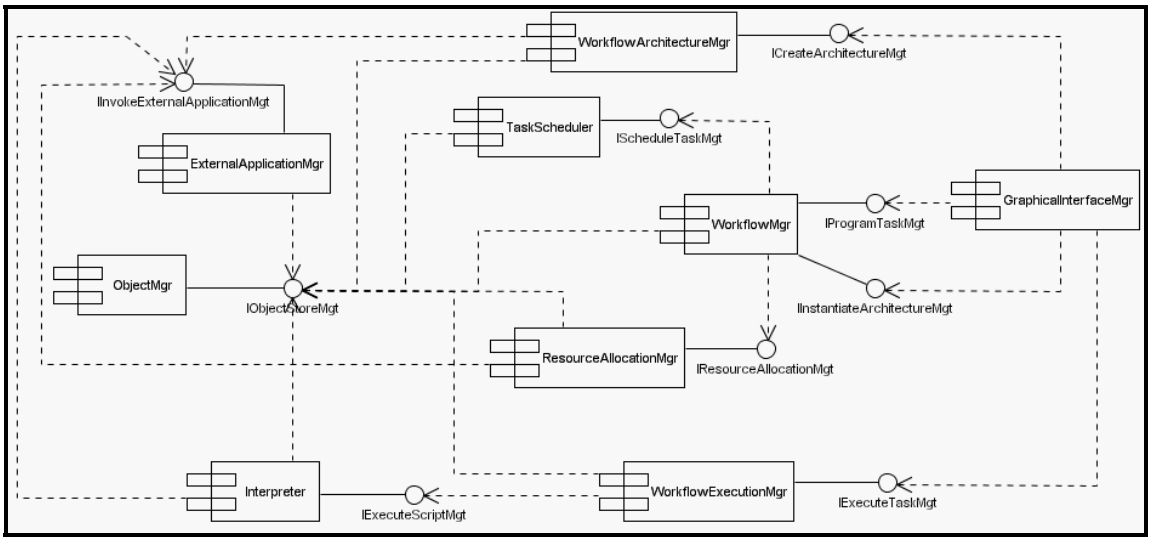

Figura 46 – Diagrama de components para WfMS.

Ao término do projeto interno dos componentes, é iniciada uma nova iteração do processo de gerenciamento de variabilidade. Assim, é realizada a atividade de **identificação das variabilidades** que têm como artefato de entrada o diagrama de componentes para WfMS (Figura 46). Como artefato de saída, é gerada a evolução do diagrama de componentes para WfMS, agora com as variabilidades identificadas. Este artefato é usado como entrada para a atividade de **delimitação das variabilidades** que gera como saída a evolução do diagrama de componentes da atividade anterior (Figura 47).

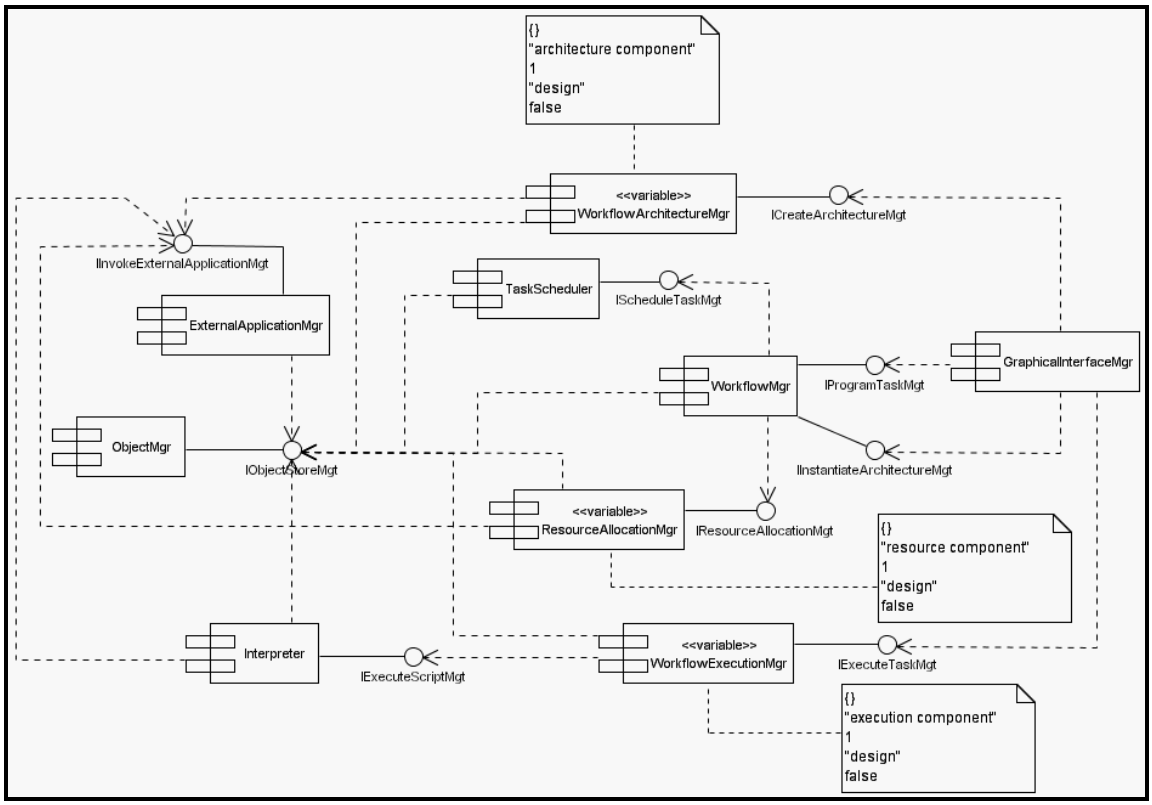

Figura 47 – Diagrama de componentes para WfMS com variabilidades representadas e delimitadas.

Na Figura 47, observa-se que somente os componentes "*WorkflowArchitectureMgr*", "*WorkflowExecutionMgr*" e "*ResourceAllocationMgr*" possuem variação, pois algumas classes pertencentes a estes componentes possuem variação. Assim, tais componentes possuem o estereótipo <<variable>> e notas da UML para as informações referentes às variabilidades de cada componente.

Com base na Figura 47, é realizada a atividade de **escolha dos mecanismos de implementação de variabilidades**, incrementando o modelo de implementação de variabilidades do Quadro 10. O Quadro 11 apresenta o modelo de implementação de variabilidades para WfMS completo.

Variação	Nível	Tempo de Resolução	Mecanismo de Implementação	Estratégia de Implementação
tool class type	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.
resource class type	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.
tool class	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.
resource class	Classe	Compilação	Superposição de Fragmento de Código	Desenvolver o software para funcionar de maneira geral e depois sobrepor o código com código específico ao produto. Pode-se utilizar programação orientada a aspectos, por exemplo.
architecture component	Componente	Projeto	Reorganização Arquitetural, Componente Variante de Arquitetura ou Componente Opcional de Arquitetura	Depende do mecanismo escolhido.
execution component	Componente	Projeto	Reorganização Arquitetural, Componente Variante de Arquitetura ou Componente Opcional de Arquitetura	Depende do mecanismo escolhido.
resource component	Componente	Projeto	Reorganização Arquitetural, Componente Variante de Arquitetura ou Componente Opcional de Arquitetura	Depende do mecanismo escolhido.

Quadro 11 - Modelo de implementação de variabilidades completo para WfMS.

Neste quadro, as três ultimas variações, referentes a componentes, permitem a escolha de um dos três mecanismos indicados para implementação de variabilidades, sendo eles: Reorganização Arquitetural, Componente Variante de Arquitetura e Componente Opcional de Arquitetura. Dependendo da escolha do mecanismo, deve-se utilizar uma estratégia diferente para a sua implementação.

Em Svahnberg, Van Gurp e Bosch (2002), os autores se referenciam a "Derivação da Arquitetura" como tempo de resolução. Neste trabalho, é utilizado o termo "Projeto" (Seção 3.3.3.2).

Para finalizar a iteração do processo de gerenciamento de variabilidade, são realizadas as atividades de **rastreamento e controle de variabilidades** e **análise de configurações de produtos**. Ao término da atividade de projeto interno dos componentes, deve ser realizada a atividade de **teste e avaliação da arquitetura de LP** para que a arquitetura de LP para WfMS resultante seja aprovada e esteja pronta para ser utilizada na geração de produtos da LP.

As seções a seguir apresentam a análise e a interpretação dos resultados obtidos com o estudo de caso.

4.2.3 Análise e Interpretação dos Resultados do Estudo de Caso

A análise dos dados de um estudo de caso depende da técnica utilizada para sua comparação (Seção 4.2.1.4) (KITCHENHAM, 1996).

Se o método utilizado for a comparação dos resultados com uma *baseline* da organização, deve-se, então, atualizar tal *baseline* com os valores do estudo de caso. Se o método utilizado for um projeto de desenvolvimento baseado em componentes, deve-se, então, utilizar técnicas estatísticas como a análise de variância para analisar os resultados da aplicação do estudo a alguns componentes aleatoriamente. Caso o método utilizado seja o de Projetos Replicados, é possível analisar subjetivamente os dados do estudo de caso com base

nas variáveis de resposta apresentando-os na forma de histogramas (KITCHENHAM, 1996; LINDVALL; TVEDT; COSTA, 2003).

Outra forma interessante de se apresentar os resultados de um estudo de caso, realizado através da técnica Projetos Replicados, é calcular a porcentagem diferencial obtida entre as variáveis de resposta do tratamento e do controle (KITCHENHAM, 1996).

As Figura 48 e 49 apresentam os histogramas dos valores obtidos no estudo de caso para a variável de resposta "número de variabilidades identificadas e representadas" (Seção 4.2.1.2). Neste estudo de caso, foi feito uma diferenciação dos valores obtidos com relação ao tipo de artefato variante.

Em um primeiro momento, os valores foram medidos sem levar em consideração se o artefato variante era obrigatório, opcional ou alternativo. Assim, o histograma da Figura 48 apresenta os valores obtidos. Pode-se observar, neste histograma, que os valores são altamente dispersivos, ou seja, variam muito em sua relação. Um exemplo são os valores obtidos para diagramas de casos de uso, em que o tratamento identificou 47 artefatos variantes enquanto o controle identificou apenas 22, uma diferença de 25 artefatos para os mesmos diagramas.

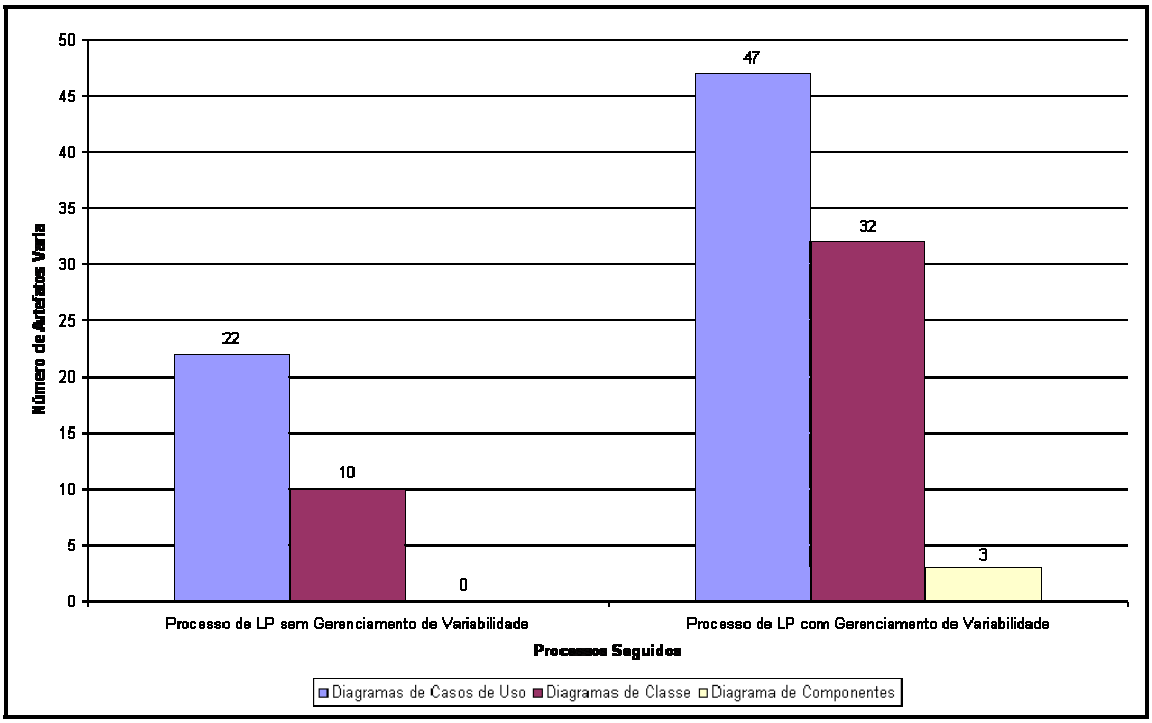

Figura 48 - Histograma de dados do estudo de caso considerando variantes obrigatórias.

Em um segundo momento, os valores foram medidos desprezando-se os artefatos variantes obrigatórios (com o estereótipo <<mandatory>>), pois este tipo de artefato se caracteriza por estar obrigatoriamente presente nos produtos de uma LP e, portanto, a sua representação não influencia no gerenciamento de variabilidades como um todo. A Figura 49 apresenta os valores obtidos sem considerar os artefatos variantes obrigatórios.

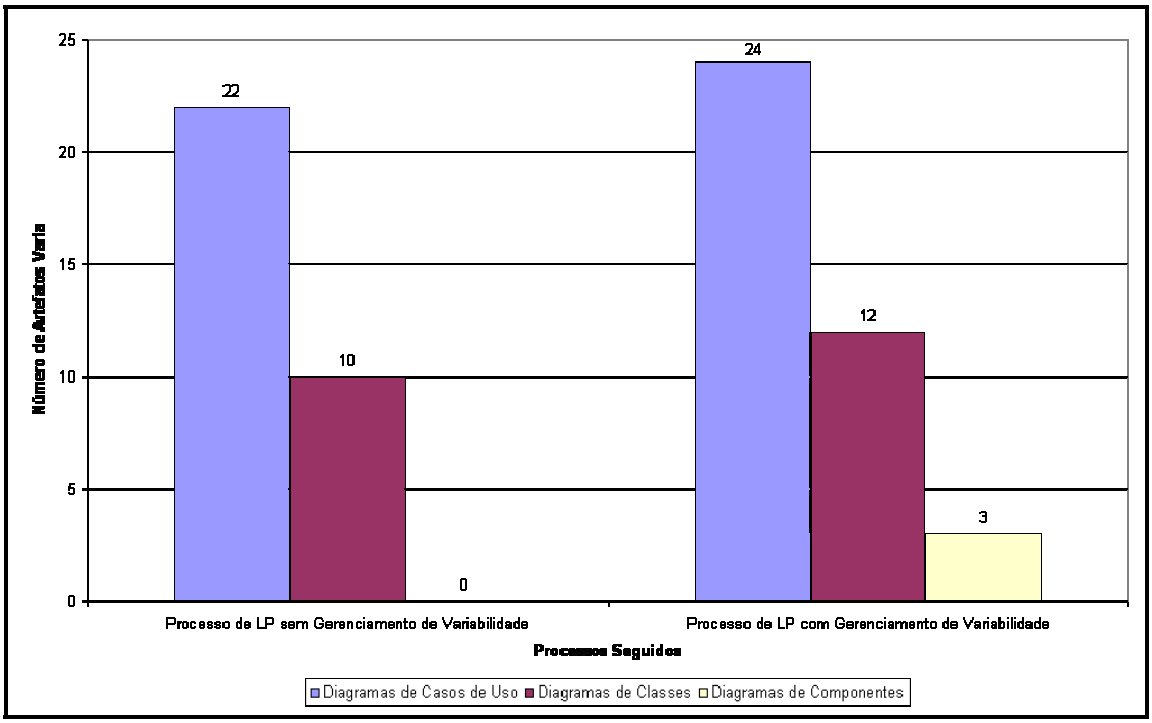

Figura 49 - Histograma de dados do estudo de caso sem considerar variantes obrigatórias.

Nesta figura, é possível observar que a diferença no número de artefatos variantes do tratamento e do controle é bastante reduzida quando comparada com a diferença do histograma da Figura 48. Por exemplo, considerando os diagramas de casos de uso da Figura 49, observa-se uma diferença no número de artefatos variantes identificados de apenas 2, reduzindo tal diferença de 25 artefatos (Figura 48) para 2 artefatos (Figura 49). O mesmo acontece para os demais tipos de artefatos, diagramas de classes e diagramas de componentes.

Apesar da redução na diferença do número de artefatos variantes sem considerar as variantes obrigatórias, observa-se, ainda, que o tratamento identificou um número maior de variabilidades do que o controle. Este fato pode ser melhor comprovado analisando-se o histograma da Figura 50.

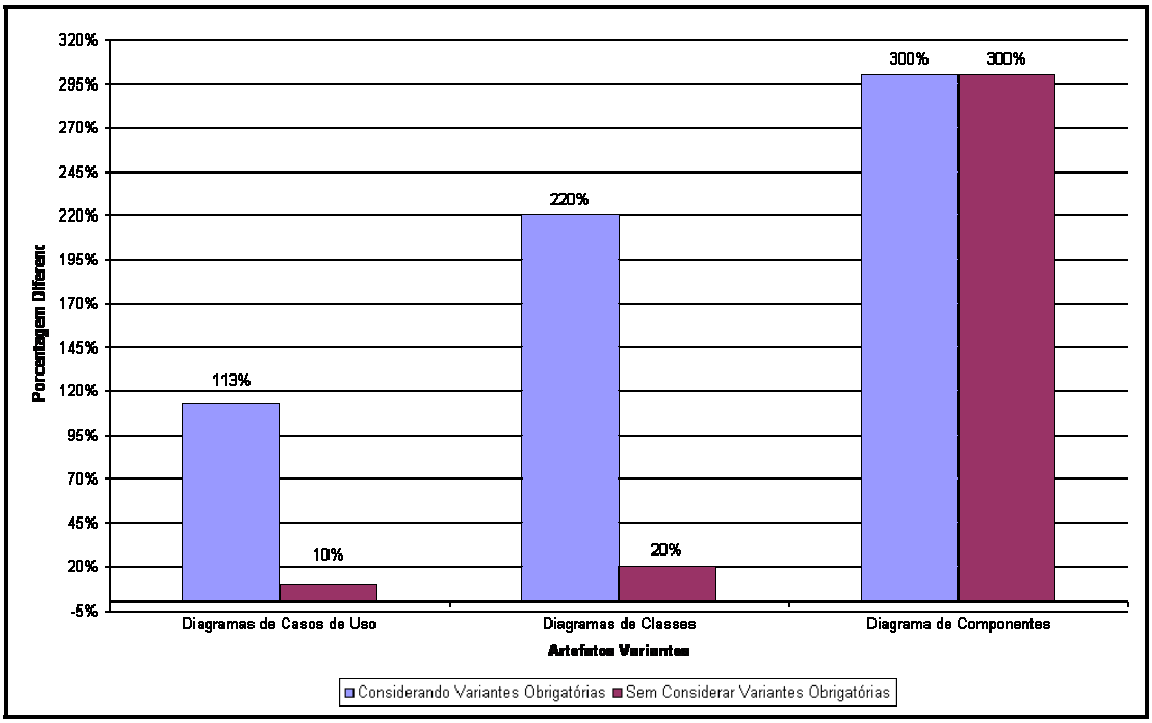

Figura 50 - Histograma de porcentagem diferencial dos valores obtidos no estudo de caso.

A Figura 50 apresenta a porcentagem de artefatos variantes (artefatos que representam pontos de variação ou variantes) que o tratamento conseguiu identificar e representar a mais do que o controle, classificados de acordo com o tipo de artefato variante. Por exemplo, considerando os diagramas de casos de uso e considerando os casos de uso variantes obrigatórios, o tratamento conseguiu identificar 113% a mais de variabilidade do que o controle. Já ao se desprezar os casos de uso variantes obrigatórios, essa porcentagem cai para 10%, porém, o tratamento ainda continua identificando mais variabilidades do que o controle. O mesmo acontece para os diagramas de classes, porém com porcentagens diferentes, 220% e 20%, respectivamente. Já para os diagramas de componentes, ambas as porcentagens são 300% pelo controle não identificar variabilidades nestes tipos de artefatos.

Pode-se concluir com isso que os resultados do estudo de caso serviram como um meio de ajudar a refutar a hipótese nula (H0) e a aceitar a hipótese alternativa (H1), em que o

tratamento permite identificar e representar um número maior de variabilidades do que o controle (Seção 4.2.1.2).

4.3 CONSIDERAÇÕES FINAIS

Os conceitos de engenharia de software experimental e avaliação empírica de métodos e ferramentas se mostram essenciais na organização e execução de estudos avaliativos. Tais estudos são apresentados de uma forma melhor estruturada, possibilitando ao usuário perceber os pontos onde o tratamento está sendo avaliado e onde se pretende chegar com o estudo.

Com base nestes conceitos, foi possível mostrar, por meio de um estudo de caso, que o processo de gerenciamento de variabilidade proposto neste trabalho de mestrado permite identificar e gerenciar um número maior de variabilidades e de tipos de variabilidades do que o processo de LP existente. Os resultados desta avaliação são comprovados com a apresentação de histogramas construídos a partir dos dados obtidos com o estudo.

Assim, a realização deste estudo de caso permite a construção de uma *baseline* para futuras avaliações do processo de LP existente.

CAPÍTULO 5

CONCLUSÕES E TRABALHOS FUTUROS

Existem evidências de que a adoção de uma abordagem de linha de produto de software é viável às organizações que pretendem estabelecer uma política bem definida de reutilização de artefatos dos seus processos de desenvolvimento. A abordagem se mostra altamente eficiente quando analisados os relatos de empresas que se beneficiam de sua adoção (BASS; CLEMENTS; KAZMAN, 2003).

Organizações que possuem um amplo conhecimento em um determinado domínio podem tirar proveito dos benefícios da abordagem de LP; deste modo, o chamado re-trabalho para o desenvolvimento dos artefatos comuns aos produtos de determinado domínio pode ser eliminado.

O gerenciamento de variabilidades é uma das atividades mais importantes do gerenciamento de uma LP, pois é por meio dele que se consegue identificar os pontos nos quais os produtos de uma LP diferenciam entre si. O gerenciamento de variabilidades também permite, com o auxílio do seu modelo de meta-dados, a análise de configuração dos produtos de uma LP, como visto na Seção 3.3.6.

Este trabalho propõe um processo de gerenciamento de variabilidade em LP que contribui para a solução de um problema em aberto na literatura existente. O processo é composto por atividades que permitem a identificação, a representação, a delimitação, a escolha de mecanismos de implementação, o monitoramento e o rastreamento de variabilidades em uma LP. O monitoramento e o rastreamento das variabilidades demandam suporte automatizado para permitir a navegação pelo modelo de meta-dados do processo em busca das variabilidades e dos artefatos variantes relacionados.

O estudo de caso realizado para avaliação do processo proposto tomou como base uma LP existente para WfMS. Foi utilizada a técnica de comparação de Projetos Replicados que permitiu refutar a hipótese nula e aceitar a hipótese alternativa de que o processo proposto permite identificar e representar um número maior de variabilidades do que o processo existente, que não possui gerenciamento de variabilidade. Concluiu-se, assim, que o processo proposto deve ser integrado ao processo de LP existente com o objetivo de melhorar o gerenciamento de variabilidades, aumentando, dessa forma, o nível de informação sobre tais variabilidades.

Também foi evidenciado que é importante construir uma *baseline* para a LP para WfMS existente a partir dos dados iniciais obtidos no estudo de caso realizado para que, futuramente, seja possível realizar experimentos formais com base na amostra de dados da *baseline* e a aplicação de técnicas estatísticas sobre esses dados.

A utilização dos conceitos de engenharia de software experimental para realização do estudo de caso mostrou-se relevante. A abordagem seguida permitiu comprovar os resultados esperados por meio de técnicas de avaliação de métodos e ferramentas de engenharia de software ao invés de simplesmente sugerir, de forma subjetiva, que os resultados estavam de acordo com as expectativas.

Como contribuições deste trabalho, têm-se a proposta do processo de gerenciamento de variabilidade e das atividades que o compõem, visto a carência de tal processo na literatura existente; a construção de um modelo de meta-dados para o processo proposto que permite o controle e o rastreamento das variabilidades de uma LP e que pode apoiar o desenvolvimento de ferramentas automatizadas para gerenciamento de variabilidade; a possibilidade de adoção do processo proposto por abordagens de LP existentes, uma vez que este utiliza como entrada artefatos comuns de tais abordagens; a realização de um estudo de caso para avaliar o

processo de variabilidade que possibilita a construção de uma *baseline* para futuras avaliações do processo de LP existente; e a análise do processo de LP existente e, conseqüentemente, a incorporação de novas atividades necessárias ao processo.

Como trabalhos futuros, podemos citar:

- a construção de uma ferramenta automatizada de apoio ao gerenciamento de variabilidade com base no modelo de meta-dados do processo proposto. Esta ferramenta permitirá a análise de configurações de produtos de uma LP com base na adição e remoção de *features* e análise dos artefatos afetados com estas ações;
- a construção de um ambiente experimental de LP para a realização de estudos com base em técnicas experimentais existentes na literatura com o objetivo de avaliar arquiteturas de LP e os seus componentes;
- um estudo sobre teste e avaliação de arquiteturas de LP, uma vez que esta atividade foi incorporada ao processo de desenvolvimento de LP existente. Para tanto, é necessário um estudo minucioso sobre as técnicas de teste e avaliação existentes na literatura como, por exemplo, os conceitos de engenharia de software experimental e avaliação empírica de métodos e ferramentas;
- a especificação e implementação de todos os componentes da LP para WfMS, visando identificar todas as possíveis variabilidades destes componentes;
- a adição de gerência de configuração e mudanças ao processo de LP existente.

REFERÊNCIAS

AMARAL, E. A. G. G. **Empacotamento de experimentos em engenharia de software**. 2003. 158 p. Dissertação (Mestrado em Ciências em Engenharia de Sistemas e Computação) – Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2003.

ANASTASOPOULOS, M. Implementing product line variabilities. **ACM SIGSOFT Software Engineering Notes**, New York, v. 26, n. 3, p. 109-117, May. 2001.

ATKINSON, C. BAYER, J., BUNSE, C., KAMSTIES, E., LAITENBERGER, O., LAQUA, R., MUTHING, D., PAECH, B., WÜST, J., ZETTEL, J. **Component-Based Product-Line Engineering with UML**. Boston: Addison-Wesley, 2001.

BACHMANN, F.; BASS, L. Managing variability in software architectures, In: SYMPOSIUM ON SOFTWARE REUSABILITY, 1., 2001, Toronto. **Proceedings…** New York: ACM, 2001. p. 126-132.

BASILI, V. R.; SELBY, R. W.; HUTCHENS, D. H. Experimentation in software engineering. **IEEE Transactions on Software Engineering**, Piscataway, v. 12, n. 7, p. 733-743, 1986.

BASS, L.; CLEMENTS, P.; KAZMAN, R.. **Software architecture in practice.** 2. ed. Boston: Addison-Wesley, 2003. 560 p.

BAYER, J.; FLEGE, O.; KNAUBER, P.; LAQUA, R.; SCHMID, K.; WIDEN, T.; DEBAUD, J. PuLSE: a methodology to develop software product lines. In: SYMPOSIUM ON SOFTWARE REUSABILITY, 5., 1999, Los Angeles. **Proceedings...**Los Angeles, 1999. p. 122-131.

BECKER, M. Towards a general model of variability in product families. In: SOFTWARE VARIABILITY MANAGEMENT WORKSHOP, 2003, Portland. **Proceedings…** Portland, 2003. p. 19-27.

BEUCHE, D.; PAPAJEWSKI, H.; SCHRÖDER-PREIKSCHAT, W. Variability management with feature models. In: SOFTWARE VARIABILITY MANAGEMENT WORKSHOP, 2003, Portland. **Proceedings…** Portland, 2003. p. 72-83.

BOSCH, J. **Design & use of software architectures**: adopting and evolving a product-line approach. Boston: Addison Wesley, 2000.

BÜHNE, S.; HALMANS, G.; POHL, K. Modelling Dependencies between Variation Points in Use Case Diagrams. In: 9th INTERNATIONAL WORKSHOP ON REQUIREMENTS ENGINEERING, 9., 2003, Austria. **Proceedings...** Austria, 2003. p. 59–69.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAND, P.; STAL, M. **Pattern-oriented software architecture**: a system of patterns. 1. ed. John Wiley & Sons, 1996.

CLAUß, M. Generic modeling using UML extensions for variability. In: OOPSLA 2001 WORKSHOP ON DOMAIN SPECIFIC VISUAL LANGUAGES, 1. 2001, Tampa Bay. **Proceedings**… Tampa Bay, 2001a. p. 11-18.

CLAUß, M. Modeling variability with UML. In: YOUNG RESEARCHES WORKSHOP., 2001, Erfurt. **Proceedings…** Erfurt. 2001b.

CLEMENTS, P.; NORTHROP, L. **Software product lines**: practices and patterns. 1. ed. Boston:

Addison-Wesley, 2001.

D'SOUZA, D.; WILLS, A. **Objects, Components and Frameworks with UML** – The Catalysis Approach. Boston: Addison-Wesley, 1999.

ECLIPSE PROJECT – Disponível em: <http://www.eclipse.org> - Acessado em: novembro de 2004.

FRITSCH, C.; LEHN, A.; STROHM, T. Evaluating variability implementation mechanisms. In: INTERNATIONAL WORKSHOP ON PRODUCT LINE ENGINEERING, 2., 2002, Seattle. *Proceedings*… Seattle, 2002. p. 59-64.

GARG, A.; CRITCHLOW, M.; CHEN, P.; VAN DER WESTHUIZEN, C.; VAN DER HOEK, A. An environment for managing product line architectures. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2003, Amsterdam. **Proceedings…** Amsterdam, 2003. p. 358-367.

GIMENES, I. M. S., ExPSEE - An Experimental Process Centred Software Engineering Environment. Maringá: 2002. UEM/CTC/DIN, Relatório Técnico.

GIMENES, I. M. S.; TRAVASSOS, G. H. O enfoque de linha de produto para desenvolvimento de software. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA DA SBC, 22., 2002, Florianópolis. **Anais...** Florianópolis, 2002. p. 34

GOMAA, H. **Designing software product lines with UML**: from use cases to pattern-based software architectures. Boston: Addison-Wesley, 2005.

GOMAA, H.; WEBBER, D. Modeling adaptive and evolvable software product lines using the variation point model. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 37., 2004, Hawaii. **Proceedings…** Hawaii, 2004. p. 01-10.

GRISS, M. L.; FAVARO, J.; D´ALESSANDRO, M. Integrating feature modeling with the RSEB. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE, 5., 1998, Washington. **Proceedings…** Washington, 1998. p. 76-85.

HALMANS, G.; POHL, K. Communicating the variability of a software-product family to customers. **Journal on Software and Systems Modeling**, New York, v. 2, n. 1, p. 15-36, mar. 2003.

HALMEMAN, R. J. **Projeto do componente gerenciador de execução de workflow segundo a abordagem de linha de produto de software**. 2003. 85 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal do Paraná, Curitiba, 2003.

HEYMANS, P.; TRIGAUX, J. C. Software product line: state of the art. Technical report for PLENTY project, Institut d'Informatique FUNDP, Namur, 2003.

KITCHENHAM, B.; PICKARD, L.; PFLEEGER, S. L. Case studies for method and tool evaluation. **IEEE Software,** v.11, p. 52-62, 1995.

KITCHENHAM, B. DESMET: a method for evaluating software engineering methods and tools. Technical Report TR96-09, Keele, United Kingdom, 1996. 49 p.

IBM RATIONAL SOFTWARE - Disponível em: <http://www.ibm.com/software/rational> - Acessado em: novembro de 2004.

JACOBSON, I.; GRISS, M.; JONSSON, P. **Software reuse** - architecture process and organization for business success. 1. ed. Boston: Addison-Wesley, 1997. 528 p.

KANG, K. Feature-oriented domain analysis (FODA) - feasibility study. Technical Report CMU/SEI-90-TR-21, SEI/CMU, Pittsburgh, 1990.

KANG, K.; KIM, S.; KIM, K.; KIM, G.; SHIN, E. FORM: A Feature-Oriented Reuse Method with Domain-Specific Refrence Architecture, SEI Technical Report, 1998.

KRUCHTEN, P. **The Rational Unified Process** – An Introduction, Second Edition. Boston: Addison-Wesley, 2000.

LAZILHA, F. R. **Uma Proposta de Arquitetura de Linha de Produto para Sistemas de Gerenciamento de Workflow**. 2002. 119 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

LINDVALL, M.; TVEDT, R. T; COSTA, P. Na empirically-based process for software architecture evaluation. **Empirical Software Engineering**, Netherlands, v. 8, p. 83-108, 2003.

MORISIO, M.; TRAVASSOS, G.; STARK, M. E. Extending UML to support domain analysis. In: THE INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 5., 2000, Grenoble. *Proceedings*... Grenoble, 2000, p. 321-324.

NISHIMURA, R. T. **Geração de Produto em uma Abordagem de Linha de Produto para Sistemas Gerenciadores de Workflow**. 2004. 125 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática, Universidade Estadual de Maringá, Maringá, 2004.

NORTHROP, L. M. SEI's Software product line tenets. **IEEE Software**, v. 19, n. 14, p. 32-41, 2002.

OLIVEIRA JUNIOR, E. A. **Especificação do ambiente ExPSEE de acordo com a abordagem de desenvolvimento baseado em componentes**. 2002. 137 f. Trabalho de Conclusão de Curso (Graduação) – Departamento de Informática, Universidade Estadual de Maringá, Maringá, 2002.

OMG - Object Management Group - Disponível em: <http://www.omg.org/technology/documents/modeling_spec_catalog.htm> - Acessado em: janeiro de 2005.

PARSONS, D.; RASHID, A.; SPECK, A. T. A framework for object oriented frameworks design. **IEEE Society**, v. 1, n. 1, p. 141-151, 1999.

PFLEEGER, S. L. Experimental design and analysis in software engineering – how to set up an experiment. **ACM SIGSOFT – software Engineering Notes.** v. 20, n. 1, p. 22-26, 1995.

PREE, W. **Design patterns for object-oriented software development**. Boston: Addison-Wesley, 1995.

PURE-SYSTEMS - pure-variants: Variant Management – Disponível em: <http://web.pure-systems.com/Variant_Management.49.0.html> - Acessado em: novembro de 2004.

SEI - Software Engineering Institute. A framework for software product line practice 4.2. Pittsburgh. Disponível em <http://www.sei.cmu.edu/plp/framework.html>. Acesso em: 22 de jun. 2004.

SIMONS, M.; CREPS, D.; KLINGLER, C.; LEVINE, L.; ALLEMANG, D. Organization domain modeling (ODM) guidebook, version 2.0. Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defence Systems, 1996.

SOCHOS, P.; PHILIPPOW, I.; RIEBISCH, M. Feature-oriented development of software product lines: mapping feature models to the architecture. Springer, LNCS 3263, 2004, p. 138-152.

SPC - SOFTWARE PRODUCTIVITY CONSORTIUM. Reuse-Driven Software Processes Guidebook. SPC-92019-CMC version 02.00.03 November 1993.

SPLC – 9th International Software Product Line Conference – Disponível em <http://splc-europe.irisa.fr>. Acessado em 10 de jan. 2005.

SUCCI, G.; YIP, J.; PEDRYCZ, W. Holmes: an intelligent system to support software product line Development. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 23., 2001, Toronto. **Proceedings…**Toronto, 2001, p.829-832.

SVAHNBERG, M.; BOSCH, J. Issues concerning variability in software product lines. In: INTERNATIONAL WORKSHOP ON SOFTWARE ARCHITECTURES FOR PRODUCT FAMILIES, 3., 2000, Las Palmas. **Proceedings…** Las Palmas, 2000. p. 146-157.

SVAHNBERG, M.; VAN GURP, J.; BOSCH, J. A taxonomy of variability realization techniques. Technical report, Blekinge Institute of Technology, Sweden, 2002.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G. G. Introdução à engenharia de software experimental. Relatório Técnico RT-ES-590/02. Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2002. 52 p.

TRIGAUX, J. C.; HEYMANS, P. Modelling variability requirements in software product lines: a comparative survey. Technical report PLENTY project, Institut d'Informatique FUNDP, Namur, 2003.

VAN DER HOEK, A. Capturing product line architectures. In: INTERNATIONAL SOFTWARE ARCHITECTURE WORKSHOP, 4., 2000, Limerick. **Proceedings…** Limerick, 2000. p. 95-99.

VAN GURP, J. **Variability in software systems** – the key to software reuse. 2000. 194 f. Trabalho de Conclusão de Curso (Graduação) - Department of Software Engineering and Computer Science, University of Groningen, Groningen, 2000.

VAN GURP, J.; BOSCH, J. Managing variability in software product lines. In: THE WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2000, Amsterdam. **Proceedings…** Amsterdam, 2000.

VAN GURP, J.; BOSCH, J. On the notion of variability in software product lines. In: THE WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, 2001, Amsterdam. *Proceedings*… Amsterdam, 2001.

WEISS, D.; CHI TAU, R. L. **Software product-line engineering**: a family-based software development process. Boston: Addison-Wesley, 1999.

WfMC – Workflow Management Coalition. Workflow Reference Model. Bruxelas, Jan. 1995. 55p. (Document number TC00-1003).

WfMC – Workflow Management Coalition. Terminology & Glossary. Bruxelas, Jun. 1999. 52p.

WfMC – Workflow Management Coalition. Disponível em <http://www.wfmc.org>. Acesso em 10 de jun. 2004.

ZELKOWITZ, M. V.; WALLACE, D. R. Experimental models for validating technology. **IEEE Computer**, v. 31, Issue 5, p. 23-31, 1998.

APÊNDICE A - CONCEITOS BÁSICOS SOBRE SISTEMAS DE GERENCIAMENTO DE *WORKFLOW*

Este apêndice apresenta os fundamentos relativos à tecnologia de *workflow* visando destacar os principais conceitos. Serão definidos os conceitos de *workflow* baseados nos padrões propostos pela WfMC (1995). A WfMC é uma organização internacional, formada em 1993, cujo objetivo é promover a área de *workflow* através da divulgação da tecnologia e do desenvolvimento de padrões para a interoperabilidade de sistemas de *workflow*. Serão definidos também os principais conceitos relacionados a sistemas de gerenciamento de *workflow* (WfMS), bem como sua arquitetura genérica e o modelo de referência propostos pela WfMC.

A.1 CARACTERIZAÇÃO DE *WORKFLOW*

As tecnologias de *workflow* e gerenciamento de *workflow* estão em contínuo crescimento e vêm sendo exploradas cada vez mais em diversos domínios de negócios. Muitos produtos de gerenciamento de *workflow* estão disponíveis no mercado, cada um deles dando ênfase em suas características particulares.

Com base nas definições da WfMC (1995) podemos conceituar *workflow* como uma coleção de atividades organizadas para realizar um processo de negócio. Um *workflow* estabelece a ordem de execução das atividades e as condições em que cada atividade pode ser iniciada, assim como a sincronização das atividades, o fluxo de informações e os participantes do grupo.

Um sistema de gerenciamento de *workflow* fornece os mecanismos para a automação dos processos de negócios através do gerenciamento das atividades de trabalho, além da

invocação de recursos humanos ou de máquinas associados aos vários passos de cada atividade. Assim, um WfMS é um sistema computacional que permite a definição, a gerência e a execução completa de *workflows* (WfMC, 1995). Esses sistemas executam em uma ou mais máquinas de *workflow* que são aptas a interpretar a definição dos processos, interagir com os participantes do *workflow* e, quando necessário, invocar ferramentas ou aplicações.

Em um nível mais alto de abstração pode-se observar que um WfMS fornece apoio em três áreas funcionais que são: funções em tempo de construção, funções de controle em tempo de execução e funções de interação em tempo de execução.

As **funções em tempo de construção** tratam a definição do processo de *workflow* e suas atividades constituintes. As **funções de controle em tempo de execução** tratam do gerenciamento do *workflow* em um ambiente operacional e do sequenciamento das várias atividades a serem tratadas como parte de cada processo. Essas funções controlam instâncias de processo, agendamento de atividades, invocação de aplicações externas, entre outras. Por fim, as **funções de interação em tempo de execução** lidam com a interação com usuários humanos e aplicações de tecnologia da informação para o processamento das várias etapas de uma atividade.

Para padronizar as especificações de produtos de *workflow*, a WfMC propôs um modelo de referência para WfMS (WfMC, 1995). Este modelo contém: a identificação das principais características, a terminologia, os componentes principais e as interfaces dos WfMS. O modelo de referência procura identificar o fluxo de informação entre os diversos componentes, além de apresentar diversos cenários de interoperabilidade. A tecnologia de *workflow* envolve muitos conceitos inter-relacionados, mostrados na Figura 51.

As seções a seguir apresentam, brevemente, a arquitetura genérica e o modelo de referência da WfMC.

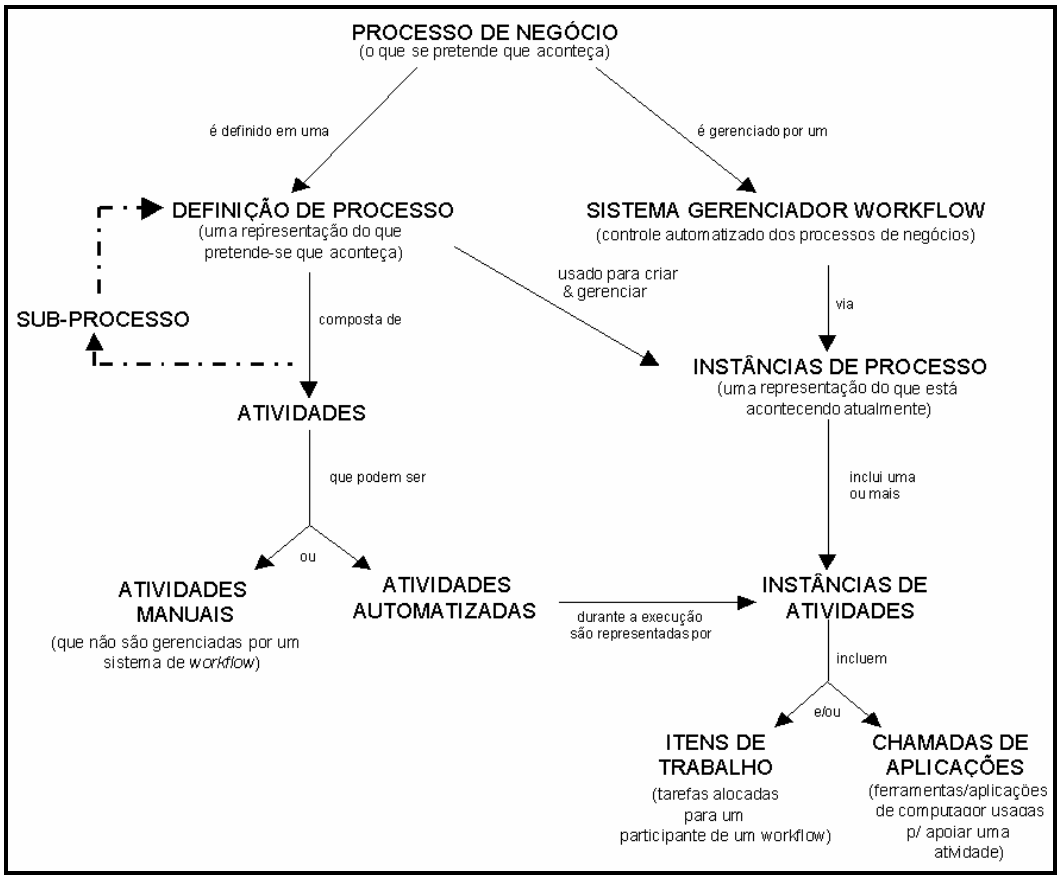

Figura 51 - Relação da terminologia básica associada com *workflow*
Fonte: WfMC (1999, p. 7).

A.2 ARQUITETURA GENÉRICA DE UM SISTEMA DE GERENCIAMENTO DE *WORKFLOW*

Apesar da variedade de produtos de *workflow* existentes, a WfMC, após uma série de análises e comparações entre produtos disponíveis, identificou a viabilidade da construção de um modelo de implementação genérico de um sistema de *workflow* que pode ser ajustado à maioria dos produtos do mercado. Esse modelo identifica os principais componentes funcionais e as suas respectivas interfaces dentro de um WfMS. A partir deste modelo, cada implementação de um WfMS pode optar por adaptar componentes ou interfaces de acordo com as necessidades da aplicação (LAZILHA, 2002). A arquitetura genérica de um WfMS é

apresentada na Figura 52.

A arquitetura é composta por três tipos de componentes: **componentes de software**: são os componentes que oferecem suporte às funcionalidades do WfMS, tais como a ferramenta de definição, a máquina de *workflow*, o gerenciador da lista de trabalho e a interface com o usuário; **dados de controle do sistema**: são os dados utilizados pelos componentes de software, tais como os dados de controle de *workflow*, os dados relevantes do *workflow* e o modelo organizacional; e **aplicativos e suas bases de dados**: são os aplicativos que não fazem parte dos WfMS, mas podem ser invocados por ele como parte do sistema de *workflow*.

Cada um desses componentes que integram a arquitetura genérica para WfMS é descrito na próxima seção, que trata do modelo de referência para WfMS derivado desta arquitetura genérica.

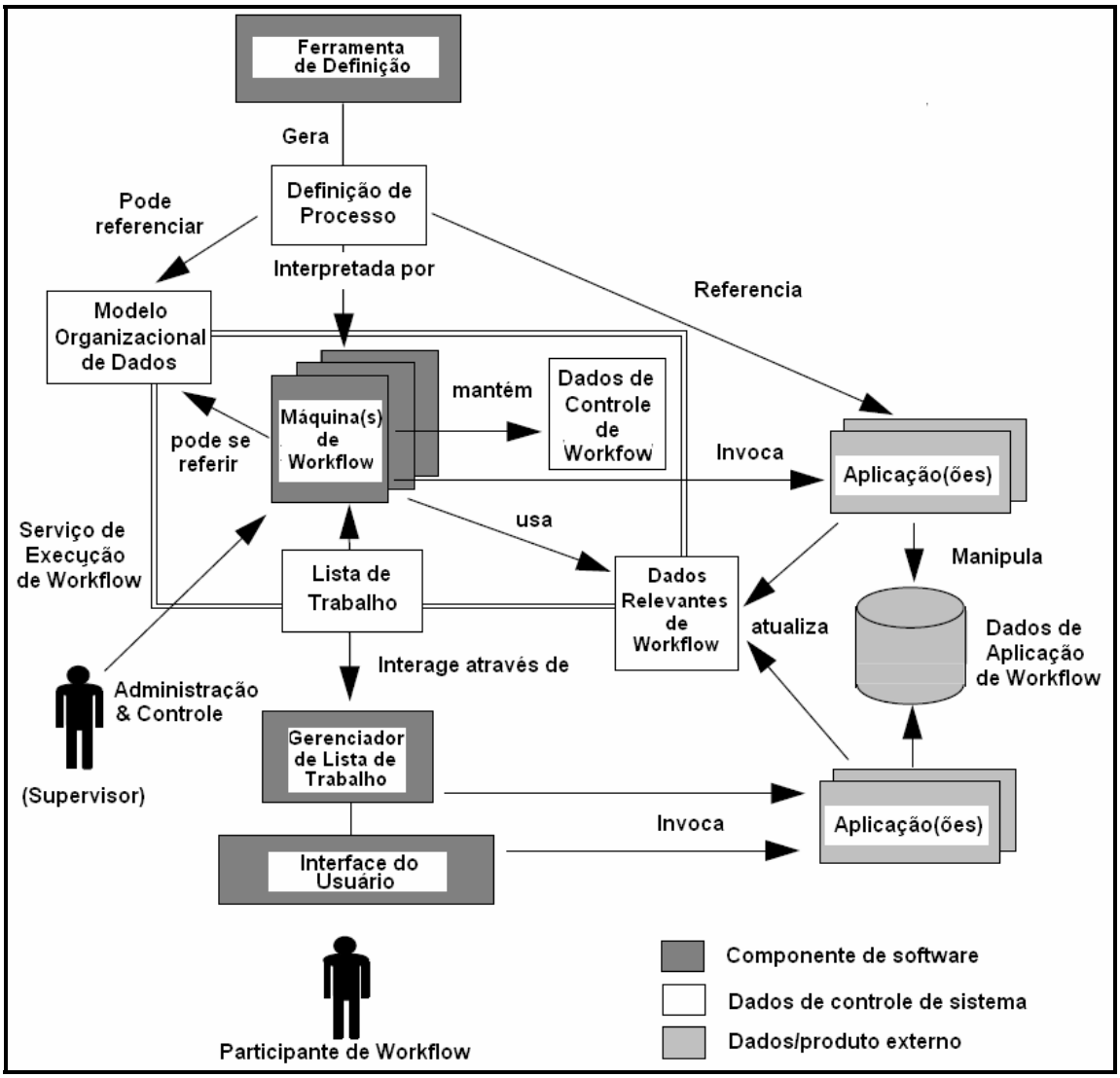

Figura 52 - Arquitetura genérica de um sistema de gerenciamento de *workflow*
Fonte: WfMC (1995, p.13).

A.3 MODELO DE REFERÊNCIA PARA SISTEMAS DE GERENCIAMENTO DE *WORKFLOW*

A arquitetura genérica de aplicações de *workflow* serviu de suporte para o desenvolvimento e construção do modelo de referência para WfMS (WfMC, 1995). Isso foi feito através da análise dos elementos que compõem a estrutura genérica dos WfMS e da

análise das interfaces existentes entre eles. Além da identificação de cada interface, identificou-se também o comportamento de cada uma delas.

Assim, um conjunto de interfaces padronizadas e formatos de intercâmbio de dados surgiu como resultado deste trabalho. A partir desse modelo, diferentes produtos de *workflow* podem alcançar variados níveis de interoperabilidade. Portanto, o modelo de referência inclui cinco padrões de interoperabilidade e comunicação que permitem a coexistência e interação de múltiplos produtos de *workflow* com o ambiente do usuário, como mostra a Figura 53.

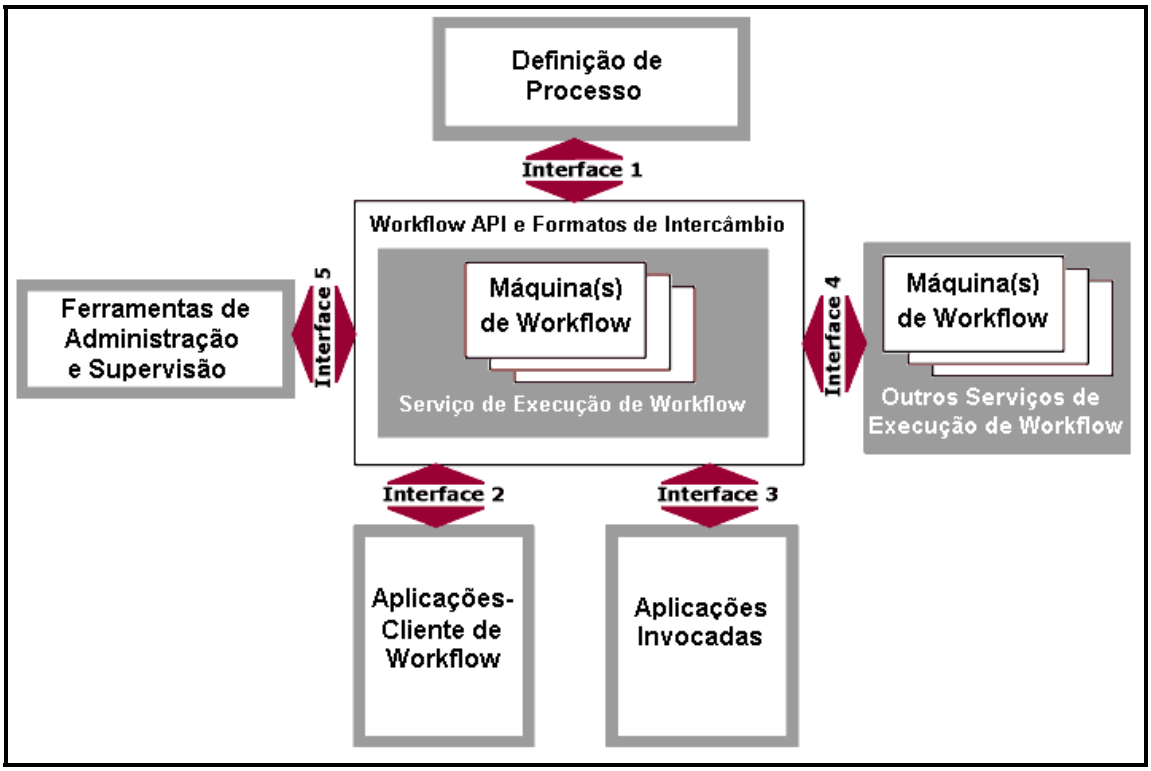

Figura 53 - Modelo de referência para sistemas de gerenciamento de *workflow*
Fonte: WfMC (1999, p. 20).

As interfaces ao redor do serviço de execução de *workflow* são chamadas de *Workflow Application Programming Interface* (*Workflow API*) e os formatos de intercâmbio de dados regulam e definem como o serviço de execução do *workflow* vai se comunicar com os outros componentes do sistema. Os itens a seguir apresentam uma discussão mais completa sobre cada uma das interfaces e componentes do modelo de referência:

- **Serviço de Execução de *Workflow*:** o serviço de execução de *workflow* é quem fornece o ambiente e os mecanismos de execução nos quais instanciações e ativações de processo ocorrem, usando para isso uma ou mais máquinas de *workflow*, as quais são responsáveis pela interpretação e ativação de parte, ou de toda a definição de processo, e também pela interação com recursos externos necessários para a execução das atividades;
- **Definição de Processo:** uma variedade de diferentes ferramentas pode ser usada para analisar e descrever um processo de negócio. Essas ferramentas podem ser manuais como, por exemplo, papel e caneta ou podem ser dispositivos mais sofisticados e formais. Da mesma forma, elas podem ou não fazer parte do produto de *workflow*. Elas devem fornecer, como resultado final, a descrição do modelo de processo de negócio de uma determinada organização;
- **Aplicações-Cliente de *Workflow*:** o gerenciador de lista de trabalhos é uma entidade de software que interage com o usuário final e com aquelas atividades que envolvem recursos humanos em um WfMS. Igualmente à ferramenta de definição de processo, o gerenciador de lista de trabalho pode ser fornecido como parte do WfMS ou pode aparecer como um produto de terceiros. O conceito de lista de trabalho serve como base para a interação entre o gerenciador de lista de trabalho e o serviço de execução de *workflow*. Uma lista de trabalhos é uma fila de itens de trabalho atribuídos por uma determinada máquina de *workflow* a um determinado usuário ou grupo de usuários. A interação mais simples ocorre quando uma máquina de *workflow* atribui um item de trabalho na lista de trabalhos para que o gerenciador de lista de trabalho posteriormente recupere este item e encaminhe para o seu destinatário;

- **Aplicações Invocadas:** um WfMS deve ter a capacidade de invocar certas aplicações para automatizar uma atividade (ou parte dela) que faz parte de um item de trabalho de um determinado participante (LAZILHA, 2002). Os tipos de invocação que podem ser feitas por um WfMS para uma aplicação podem variar de uma simples chamada local de função até a execução de procedimento remoto. Algum tipo de mecanismo de integração pode ser utilizado na implementação de um WfMS para garantir a comunicação entre a máquina de *workflow* e a aplicação invocada;
- **Interoperabilidade entre Serviços de Execução de *Workflow*:** através da interface de interoperabilidade, é possível que várias implementações diferentes de WfMS troquem itens de trabalho entre si, possibilitando, dessa forma, que duas ou mais máquinas de *workflow* se comuniquem e trabalhem juntas;
- **Ferramentas de Administração e Supervisão:** por fim, é necessária também a existência de ferramentas e interfaces que possibilitem atividades de administração e monitoramento dentro de um WfMS. Dessa maneira, tornam-se possível a avaliação do estado geral e a extração de métricas do sistema, elementos que são indispensáveis em algumas organizações.

APÊNDICE B – ARTEFATOS DE ENTRADA E SAÍDA DAS ATIVIDADES DO PROCESSO DE GERENCIAMENTO DE VARIABILIDADE

Atividades	Artefatos de Entrada	Artefatos de Saída
Elaboração do Modelo de Rastreamento de Variabilidades	Modelo de Casos de Uso Modelo de *Features*	Modelo de Rastreamento de Variabilidades
Identificação das Variabilidades	Modelo de Casos de Uso Modelo de *Features* Modelo Estático de Tipos Diagramas de Componentes	Modelo de Casos de Uso Modelo de *Features* Modelo Estático de Tipos Diagramas de Componentes
Delimitação das Variabilidades	Modelo de Casos de Uso Modelo de *Features* Modelo Estático de Tipos Diagramas de Componentes	Modelo de Casos de Uso Modelo de *Features* Modelo Estático de Tipos Diagramas de Componentes
Escolha dos Mecanismos de Variabilidade	Modelo Estático de Tipos Diagramas de Componentes	Modelo de Implementação de Variabilidades
Rastreamento e Controle das Variabilidades	Modelo de Meta-Dados do Processo Modelo de Rastreamento de Variabilidades	----------
Análise de Configurações de Produtos Específicos	Modelo de Meta-Dados do Processo Modelo de Rastreamento de Variabilidades Modelo de Implementação de Variabilidades	----------