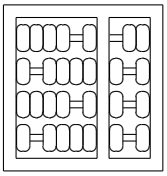

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Um Estudo para Implantação de Linha de Produto de Software baseada em Componentes

Ana Elisa de Campos Lobo
Cecília Mary Fischer Rubira

Technical Report - IC-09-17 - Relatório Técnico

May - 2009 - Maio

Um Estudo para Implantação de Linha de Produto de Software baseada em Componentes

Ana Elisa de Campos Lobo* Cecília Mary Fischer Rubira†

Resumo

A atividade de desenvolvimento de software vem enfrentando crescentes desafios em termos de diminuição de custos, esforço e tempo de chegada ao mercado, acompanhados do aumento da complexidade e do tamanho dos produtos de software. Para atender a estas demandas, novos enfoques que favoreçam o reuso de artefatos de software têm sido propostos, tais como o de Engenharia de Linha de Produto. A engenharia de linha de produto consiste em projetar, implementar e evoluir um conjunto de produtos com alto grau de similaridade entre si, de forma prescritiva, a partir de um conjunto de artefatos básicos. Empresas que desenvolvem produtos de software em um determinado domínio de aplicação podem obter ganhos significativos em termos de redução de esforço e custos utilizando o enfoque de linha de produto, desenvolvendo vários produtos similares ao mesmo tempo, ao invés de focar no desenvolvimento de um único produto por vez.

A Engenharia de Domínio é um dos fundamentos básicos para a engenharia de linha de produto. A Análise de Domínio, que é uma das atividades da engenharia de domínio, consiste na identificação dos objetos e operações de um conjunto de sistemas similares em um domínio específico. Na literatura podem ser encontrados vários métodos de engenharia de domínio e de linha de produto; entretanto, a maioria deles tratam do estabelecimento de uma linha de produtos desde o seu início, e não a partir de componentes e sistemas baseados em componentes já existentes. Hoje já existem várias empresas que usam desenvolvimento baseado em componentes, o que facilita o estabelecimento de uma linha de produtos a partir de artefatos já existentes.

Este relatório apresenta um estudo para a implantação de linha de produto em uma empresa de desenvolvimento de software que desenvolve produtos com alto grau de similaridade entre si, e que já utiliza o paradigma de desenvolvimento baseado em componentes. Como passo inicial para a criação de uma linha de produto foi feita uma análise de domínio, visando identificar e modelar as similaridades existentes entre os produtos desenvolvidos pela empresa. O método escolhido para a análise de domínio foi o FODA, e a ferramenta utilizada para suporte à modelagem do domínio foi o Odyssey. O objetivo deste estudo é obter familiaridade com métodos e ferramentas de análise de domínio, buscando identificar possíveis problemas e apontar soluções para a construção de uma linha de produto de software baseada em componentes pré-existentes.

O estudo de caso apresentado foi realizado em uma empresa da área de aplicações financeiras, em Janeiro de 2006.

*alobo@ic.unicamp.br

†cmrubira@ic.unicamp.br

1 Introdução

Nos dias atuais, a atividade de desenvolvimento de software vem enfrentando vários desafios. Por um lado, existe a necessidade crítica de redução de custo, esforço e tempo de chegada ao mercado dos produtos de software. Por outro lado, a complexidade e o tamanho dos produtos de software vem crescendo muito, assim como as necessidades dos clientes por produtos de qualidade que atendam suas necessidades individuais. Para atender a estas necessidades às vezes contraditórias, novos enfoques que favoreçam o reuso de artefatos de software têm sido propostos, como o de `Engenharia de Linha de Produto` ([27]). Uma `Linha de Produto de Software` é um conjunto de produtos de software com alto grau de similaridade entre si, que atendem às necessidades específicas de um segmento de mercado ou missão, e que são desenvolvidos de forma prescritiva a partir de um conjunto de artefatos básicos ([10]).

Hoje em dia, muitas empresas desenvolvem e mantêm vários produtos de software similares pertencentes a um mesmo domínio de aplicação (por exemplo, telecomunicações ou financeiro). Para o desenvolvimento de tais produtos com alto grau de similaridade entre si, empregar enfoque de linha de produto de software traz benefícios consideráveis, tanto em termos de redução de custo e esforço, quanto em melhoria da qualidade e produtividade ([9, 34, 38, 40, 6]).

A `Engenharia de Domínio` e mais especificamente a `Análise de Domínio` são fundamentos básicos para a engenharia de linha de produto. Na literatura podem ser encontrados vários métodos de engenharia de domínio e de linha de produto, como o FODA, ODM, FAST, DSSA, PuLSE, KobrA, dentre outros. De um modo geral, estes métodos tratam da criação de uma linha de produtos inteiramente nova, e não a partir de sistemas legados, ou seja, produtos já existentes.

O `Desenvolvimento Baseado em Componentes` (DBC) pode suportar o enfoque de linha de produto através da separação entre especificação de componente e implementação de componente, favorecendo o reuso de software, e facilitando a construção de sistemas de software a partir de componentes reutilizáveis ([8]). Atualmente, já existem várias empresas que utilizam o paradigma de DBC. Entretanto, poucas empresas exploram conceitos importantes de engenharia de domínio, como por exemplo, análise de domínio ([28]), modelagem de características ([22]) e gerência de variabilidades de produtos ([20]).

Este relatório apresenta um estudo para a implantação de linha de produto em uma empresa de desenvolvimento de software que implementa aplicações com alto grau de similaridade entre si, e que já utiliza o paradigma de desenvolvimento baseado em componentes. O objetivo deste estudo de é o de levantar os requisitos necessários para implantar o conceito de linha de produto de software no processo de desenvolvimento de software da empresa. Desta forma, o primeiro passo foi realizar uma análise de domínio sobre os produtos desenvolvidos pela empresa, visando identificar possíveis problemas ou facilitadores na implantação de uma linha de produtos de software. O objetivo da análise de domínio é modelar o domínio dos produtos, identificando características comuns e variáveis entre eles, visando identificar pontos de melhorias no processo de desenvolvimento de uma família de produtos em um determinado domínio, sugerindo novos enfoques, metodologias e ferramentas que possam ser utilizados para obter melhores resultados em termos de custo, prazo e qualidade. Para a

realização da análise de domínio foi escolhido o método FODA ([22]) e a ferramenta Odyssey ([39]) foi utilizada como suporte à modelagem de domínio.

Este relatório técnico está organizado da seguinte maneira. Na Seção 2 são apresentados fundamentos básicos de linhas de produto, DBC, variabilidade e gerência de configuração. Na Seção 3 o método FODA é descrito de forma resumida. Na Seção 4 é apresentado o Odyssey, que é um ambiente de desenvolvimento de aplicações baseadas em engenharia de domínio. Na Seção 5 é detalhado como foi realizado o estudo de caso em análise de domínio, e seus resultados são apresentados e analisados. Na Seção 6 são apresentados trabalhos relacionados à implantação de linha de produto ou de desenvolvimento baseado em componentes. Finalmente, na Seção 7 são apresentadas as conclusões e considerações finais, e são apresentados os trabalhos futuros a serem realizados.

2 Fundamentos

2.1 Engenharia de Domínio e Engenharia de Aplicação

`Engenharia de Domínio` é a atividade de coletar, organizar e armazenar experiências anteriores na construção de aplicações em um domínio específico na forma de artefatos reutilizáveis, e a utilização sistemática destes artefatos na construção de novas aplicações ([32]). A `Análise de Domínio,` uma das fases da engenharia de domínio, é a atividade de identificação dos objetos e operações de uma classe de aplicações similares em um domínio específico ([28]). Os termos engenharia de domínio e análise de domínio são usados de forma inconsistente, mas embora a definição de análise de domínio seja mais antiga, a engenharia de domínio é mais abrangente. Estes conceitos fornecem o embasamento para o enfoque de engenharia de linha de produto ([32]).

A engenharia de domínio consiste nas fases de análise de domínio (definição de escopo e modelagem do domínio), projeto de domínio (especificação da arquitetura) e implementação de domínio (implementação da arquitetura e dos componentes do domínio) para uma família de aplicações em um domínio ([12, 32]).

`Engenharia de Aplicação` é o processo de construção de sistemas a partir dos resultados obtidos na engenharia de domínio ([12]). Na engenharia de aplicação, aplicações são instanciadas a partir da seleção de um subconjunto de artefatos reutilizáveis que foram desenvolvidos na engenharia de domínio ([29]). Os processos de engenharia de domínio e engenharia de aplicação são paralelos, e existem pontos em que os resultados da engenharia de domínio alimentam a engenharia de aplicação (Figura 1).

2.2 Engenharia de Linha de Produto

A utilização de métodos de engenharia de domínio deveria melhorar a eficiência do desenvolvimento de software devido à economia de escopo, por tratar similaridades entre sistemas em um determinado domínio. Desta forma, o domínio de aplicações seria utilizado para definir o escopo da infra-estrutura de reutilização. Entretanto, a utilização destes métodos em empresas mostrou-se não tão efetiva quanto o esperado. Um dos fatores é a dificuldade de utilização do domínio para definir o escopo dos produtos desenvolvidos pela empresa,

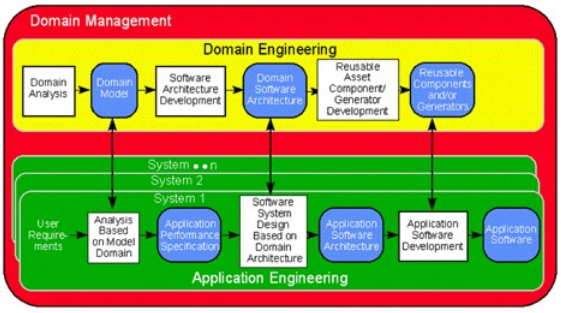

Figura 1: Engenharia de Domínio e Engenharia de Aplicação

devido ao fato que um escopo mapeia vários elementos externos que não são de interesse da empresa. Então, a visão de domínio não provê uma boa base econômica para se tomar decisões sobre o escopo. Por outro lado, empresas tomam decisões focadas em produtos. A diferença de foco é que produtos encobrem vários domínios de aplicação, e normalmente cobrem somente uma parte do domínio completo. Isto motivou uma iniciativa diferente, a engenharia de linha de produto, que é uma derivação da engenharia de domínio ([5]).

Uma `Família de Produtos` é um conjunto de produtos construídos a partir de um conjunto de artefatos comuns, tais como arquitetura e um conjunto de componentes. `Engenharia de Linha de Produto` é o enfoque que trata os produtos de software desenvolvidos por uma empresa como membros de uma mesma família de produtos que compartilham várias funcionalidades em comum. O gerenciamento das funcionalidades comuns e variáveis da família de produtos não é uma tarefa simples, e para isso a engenharia de linha de produto divide o ciclo de vida de desenvolvimento em duas fases concorrentes: engenharia de família (*family engineering*) e engenharia de aplicação (*application engineering)* (Figura 2). A engenharia de família analisa os produtos (existentes e planejados) de uma empresa em relação às suas funcionalidades comuns e variáveis, que são então utilizadas para construir uma infra-estrutura para a linha de produto, como um repositório de artefatos reutilizáveis. A engenharia de família é semelhante à engenharia de domínio, com a diferença que neste caso o foco é nos produtos, e não no domínio. A engenharia de aplicação usa esta infra-estrutura para derivar produtos específicos ([27]).

Uma outra diferença essencial entre engenharia de domínio e engenharia de linha de produto é em relação à integração entre os processos de engenharia de família e engenharia de aplicação. A engenharia de família cobre o desenvolvimento de artefatos comuns reutilizáveis, enquanto que a engenharia de aplicação deriva produtos a partir destes artefatos. Entretanto, para o estabelecimento bem sucedido destes processos em uma empresa é necessário tratar uma série de questões que ultrapassam o escopo da engenharia de domínio. Em particular, existem questões de gerenciamento e organizacionais relativas ao estabelecimento destes processos, bem como a coordenação entre eles ([12]). Tais questões estão divididas em gerenciamento técnico (gerência de configuração, análise de compra ou subcontratação, planejamento técnico, planejamento de escopo, etc) e gerenciamento organizacional (institucionalização gerenciamento de riscos, planejamento organizacional, análise de mercado, etc) de uma linha de produto. Existe uma iniciativa do *Software Engineerig*

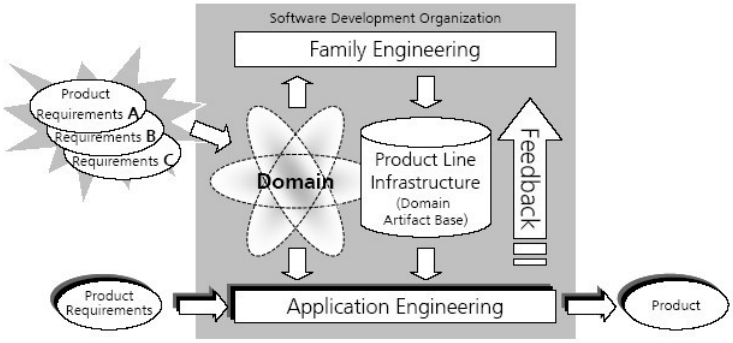

Figura 2: Ciclo de Vida de Linha de Produto

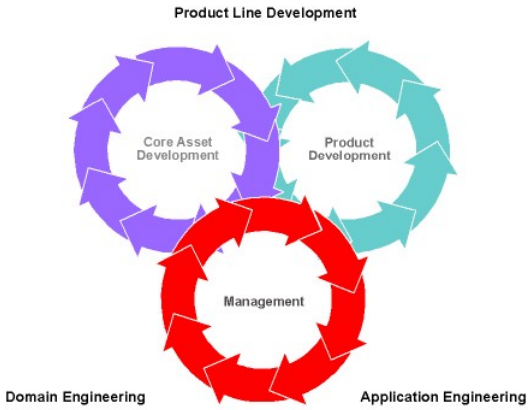

Figura 3: Representação da Estrutura para Linhas de Produto do SEI

Institute (SEI) , denominada *Framework for Product Line Practice* (guia *on-line*, disponível em `http://www.sei.cmu.edu/productlines/framework.html`), cuja idéia é identificar as diferentes questões e práticas relevantes para o estabelecimento bem-sucedido de linha de produto em uma organização. A Figura 3 representa as três atividades essenciais no desenvolvimento de uma linha de produto, segundo o SEI. Em resumo, o desenvolvimento de uma linha de produto envolve o desenvolvimento de bens comuns (*Core Asset Development*), o desenvolvimento de produtos utilizando estes bens comuns (*Product Development*) , através de um gerenciamento técnico e organizacional (*Management*). O desenvolvimento de bens comuns é equivalente à engenharia de domínio, e o desenvolvimento de produtos é equivalente à engenharia de aplicação. As três atividades são representadas como círculos em rotação, indicando que são essenciais, estão intrinsecamente ligadas, a saída de uma realimenta a entrada da outra, e são iterativas.

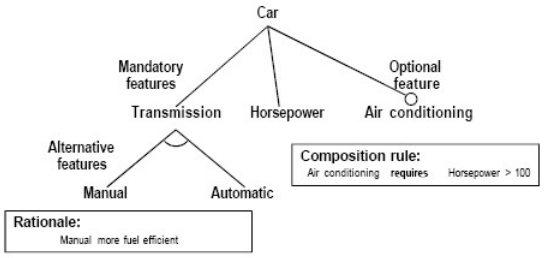

Figura 4: Modelo de Características de um Carro

2.3 Características e Modelo de Características

`Característica` (ou *feature*) *é* uma propriedade de sistema que é relevante para alguma parte interessada e é usada para capturar funcionalidades comuns ou variáveis entre sistemas de uma mesma família de produtos ([13]). Na definição inicialmente proposta por Kang ([22]), são atributos de sistema que afetam diretamente o usuário final. A primeira definição é preferível por ser mais abrangente. As características de um sistema de software são documentadas em um modelo de características. Um `Modelo de Características` consiste dos seguintes elementos principais: (i) diagrama de características, representando a decomposição hierárquica das características; (ii) regras de composição para as características; e (iii) análise racional das características.

O diagrama de características representa as características padrão de uma família de sistemas em um domínio e o relacionamento entre elas. Um dos relacionamentos possíveis é o agrupamento de características. Características podem ser `mandatórias`, `opcionais` ou `alternativas`, sendo que características alternativas só têm sentido dentro de um agrupamento de características. Outro tipo de relacionamento é a especialização de uma característica. A Figura 4 representa o modelo de caraterísticas de um carro. As características *Transmission* e *Horsepower* são mandatórias, enquanto que a característica *Air Conditioning* é opcional. A transmissão de um carro pode ser automática ou manual, representadas pelas características alternativas *Manual* e *Automatic*. As características alternativas *Automatic* e *Manual* são especializações da característica *Transmission*. `Regras de composição` definem a semântica existente entre características que não pode ser expressa no diagrama de características, indicando quais combinações de características são válidas. No exemplo, uma regra de composição indica qual a potência necessária para que o carro tenha ar condicionado (*Air Conditioning requires Horsepower* $>$ *100*). Uma `análise racional` é uma recomendação especificando quando uma determinada característica deve ou não ser selecionada. No exemplo do carro, a recomendação é que a transmissão manual é mais econômica que a automática.

2.4 Métodos de Engenharia de Domínio ou Linha de Produto

Existem vários métodos de engenharia de domínio ou linha de produto. O primeiro deles foi o `Draco` ([28]), que define um enfoque em engenharia de domínio e um ambiente ba-

seado na tecnologia de transformações. As principais idéias introduzidas por Draco foram Análise de Domínio, linguagens de domínio específico e componentes como um conjunto de transformações.

`FODA` (*Feature-Oriented Domain Analysis*) é um método de análise de domínio, criado pelo SEI. Este método foi responsável pela introdução do modelo de características (Seção 2.3) para tratar as variabilidades entre sistemas, e é um dos métodos mais maduros e bem documentados que existem. Detalhes deste método serão apresentados na Seção 3.

`ODM` (*Organization Domain Modeling*) é um método de engenharia de domínio que assimilou vários outros enfoques de engenharia de domínio e foi empregado em vários projetos, notadamente no projeto STARS (*Software Technology for Adaptable, Reliable Systems*). Este método introduziu vários conceitos novos, como: (i) tipos de domínios (horizontais ou verticais); (ii) definição mais abrangente de característica; (iii) análise de combinação de características; (iv) modelagem conceitual; (v) refinamento do escopo da base de bens; (v) arquitetura flexível, ou arquitetura de artefatos básicos, para representar as variabilidades; e (vi) processo customizado ([12, 31]).

`DSSA` (*Domain-Specific Software Architecture*) é um enfoque que enfatiza o papel da arquitetura de software na engenharia de domínio. Os principais produtos de trabalho do processo DSSA incluem os seguintes: (i) modelo de domínio ou modelo de conceitos; (ii) requisitos de referência (equivalente ao modelo de características do FODA); e (iii) arquitetura de referência para uma família de sistemas. A necessidade de representação formal dos componentes de uma arquitetura e seus relacionamentos levaram ao desenvolvimento de linguagens de descrição de arquitetura ou ADLs[1]. O método DSSA foi aplicado a vários projetos, e este esforço resultou em um conjunto de ferramentas e outros produtos para apoiar o processo, notadamente ADLs específicas ([37]).

`FeatuRSEB` é o resultado da integração de FODAcom, uma adaptação de FODA orientado a objetos para o domínio de telecomunicações, com RSEB (*Reuse-driven Software Engineering Business*), um método de engenharia de software orientado a reuso e a objetos, baseado na notação UML. O método FeatuRSEB é centrado no modelo de características, e faz a distinção entre características funcionais, arquiteturais e de implementação ([19]).

`FAST` (*Family-Oriented Abstraction, Specification and Translation*) é um processo de desenvolvimento para produzir software para uma família de produtos, desenvolvido pela Lucent e aplicado a vários domínios e produtos específicos. Este processo pode ser dividido em três sub-processos: (i) qualificação do domínio para identificar famílias de produto economicamente viáveis, (ii) engenharia de domínio para investir em facilidades para produzir membros da família de produtos, e (iii) engenharia de aplicação para usar as facilidades para produzir membros da família com eficiência. Cada sub-processo consiste em diferentes atividades, produz diferentes artefatos e são executados por pessoas atuando em diferentes papéis. O processo FAST pode ser aplicado de maneira consistente e disciplinada, utilizando o modelo PASTA (*Process and Artifact State Transition Abstraction*), que fornece instruções precisas a serem seguidas durante o processo ([21]).

`PuLSE` (*Product Line Software Engineering*) é uma metodologia de linha de produto que tem por objetivo permitir a implantação e evolução de linhas de produto de software em

[1] *Architecture Description Languages*

uma variedade de contextos industriais. O método PuLSE é composto por três elementos principais: fases de estabelecimento, componentes técnicos e componentes de suporte. As fases de estabelecimento da linha de produto são: inicialização, construção da infra-estrutura, utilização da infra-estrutura, evolução e gerenciamento da linha de produto. Os componentes técnicos provêm o conhecimento necessário para operacionalizar o desenvolvimento da linha de produto. Os componentes de suporte são pacotes de apoio utilizados pelos outros componentes, que atuam como guias para permitir uma melhor adaptação, evolução e implantação da linha de produto. O método PuLSE é fortemente focado nos produtos, através das fases da linha de produto, da customização de seus componentes, e da sua introdução incremental por níveis de maturidade, em diferentes ambientes industriais ([3]).

O método `KobrA` é uma síntese de diversas abordagens de engenharia de software, incluindo linha de produtos, desenvolvimento baseado em componentes, *frameworks* e modelagem de processos. A engenharia de *framework* é responsável por criar e manter um *framework* genérico para toda a família de produtos, incluindo funcionalidades comuns e variáveis entre eles. Na engenharia de aplicação, o *framework* é instanciado e estendido, e uma aplicação é construída através da resolução dos pontos de variabilidade do *framework*. A modelagem de componentes, dividida nas fases de especificação de componentes (o que fazer) e realização de componentes (como fazer), é feita através de diagramas UML estendido para tratamento de variabilidades ([1]).

2.5 Engenharia de Software Baseada em Componentes

Um `componente de software` pode ser definido como uma unidade independente de software que encapsula seu projeto e implementação, e oferece serviços para o meio externo através de interfaces bem definidas ([2]). Componente é uma unidade de composição com interfaces especificadas por contrato e dependência explícita de contexto. Um componente de software pode ser distribuído de maneira independente e pode ser composto por partes desenvolvidas por terceiros ([35]). Um componente pode ser atômico ou composto. Um componente composto é constituído por outros componentes, que são seus subcomponentes.

Um componente de software tem interfaces providas e requeridas. Uma interface provida representa um ponto de acesso a serviços ou capacidades que o componente oferece ao meio externo. Em geral, um componente tem múltiplas interfaces correspondendo às diferentes visões que os clientes têm dos serviços providos pelo componente. Por outro lado, uma interface requerida especifica as dependências do componente, isto é, os serviços que o componente requer e que são providos por outros componentes ([35]).

`Desenvolvimento Baseado em Componentes` (DBC) ([16]) é uma abordagem para o desenvolvimento de software em que todos os artefatos, do código executável até as especificações das interfaces, arquiteturas e modelos de negócios, podem ser construídos montando-se, adaptando-se e interconectando-se componentes existentes em diversas configurações.

`Engenharia de software baseada em componentes` é um paradigma emergente de desenvolvimento que promete acelerar o desenvolvimento de software e reduzir custos através da construção de sistemas a partir de componentes de software previamente fabricados.

2.6 Variabilidade e Gerência de Configuração de Software

O conceito de variabilidade é fundamental no desenvolvimento de linhas de produto. `Variabilidade de software` é a capacidade de um sistema de software ou artefato ser modificado, customizado ou configurado, para ser utilizado em um contexto específico ([20]). Um alto grau de variabilidade permite a utilização do artefato de software em um contexto mais amplo, o que torna o artefato mais reutilizável. É possível antecipar alguns tipos de variabilidade e construir sistemas que facilitem este tipo de variabilidade. Em linhas de produto ([10]), os artefatos de software devem ser flexíveis o suficiente de modo a permitir que detalhes de implementação de um produto específico possam ser postergados para fases posteriores do desenvolvimento. Estas decisões de projeto postergadas são denominadas de pontos de variação. Um `ponto de variação` é o local do artefato de software em que uma decisão de projeto pode ser tomada, e `variantes` são alternativas de projeto associadas a este ponto ([20]).

Em DBC, a separação entre a especificação e implementação de um componente provê um tipo de variabilidade. Uma outra espécie de variabilidade está relacionada à composição de componentes. Um componente pode ser composto por diferentes estruturas e combinações de elementos arquiteturais, fornecendo pontos de variação.

Variabilidade pode ser dividida em duas dimensões: espaço e tempo. A dimensão de espaço é relativa ao uso do software em múltiplos contextos, por exemplo, os vários produtos de uma linha de produtos de software. A dimensão de tempo é relativa à capacidade do software apoiar a evolução e mudança de requisitos ao longo do tempo. O conjunto de atividades envolvidas no gerenciamento de variabilidades em artefatos de software na dimensão do tempo é tradicionalmente conhecida como gerência de configuração de software ([20]).

`Gerência de Configuração de Software` é a disciplina de gerenciar a evolução de sistemas de software grandes e complexos. O objetivo principal da gerência de configuração é garantir que as mudanças nos artefatos de desenvolvimento de software sejam feitas através de uma maneira sistemática e rastreável, de modo que o sistema de software esteja sempre em um estado bem definido e consistente ([30]).

3 Método FODA

O método FODA é um método de análise de domínio que visa a construção de artefatos genéricos que sejam largamente reutilizáveis dentro do domínio ([22]). O objetivo principal da análise de domínio é obter e representar informações sobre sistemas de software que compartilham características comuns. Este método foi desenvolvido pelo SEI em 1990, e consiste nas atividades de `Análise de Contexto` e `Modelagem de Domínio`[2]. A atividade de modelagem do domínio é subdividida nas atividades: `Análise de Características`, `Modelagem Entidade-Relacionamento` e `Análise Funcional`. A Figura 5 representa o diagrama de atividades do método FODA. As subseções seguintes descrevem cada uma destas atividades.

[2] Originalmente, o método FODA continha uma terceira atividade denominada Modelagem da Arquitetura. Posteriormente, em 1997, esta atividade foi transferida para a fase de Projeto de Domínio do MBSE (*Model-Based Software Engineering*)[33]), também do SEI.

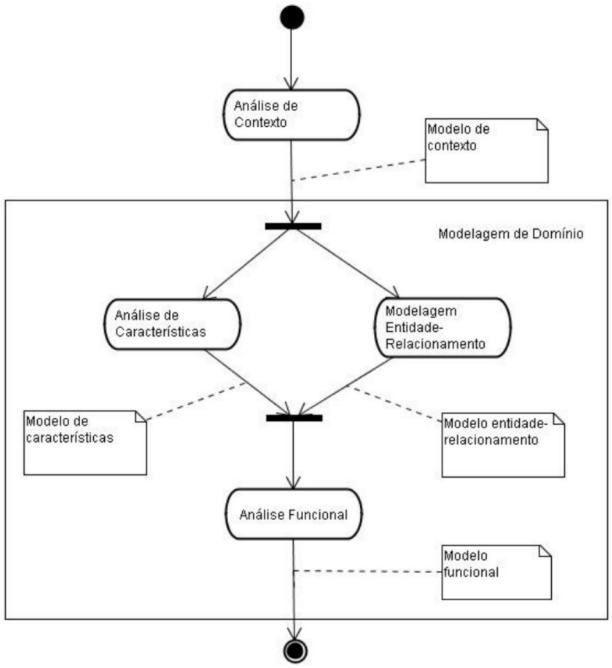

Figura 5: Atividades do Método FODA

O método FODA foi escolhido para realizar o estudo de caso em análise de domínio, por ser o método mais maduro, documentado e conhecido, dentre os métodos de análise de domínio. Apesar disso, ele tem suas deficiências, como a falta de orientação quanto à modelagem da arquitetura do domínio.

3.1 Análise de Contexto

O propósito da análise de contexto é definir o escopo do domínio que será explorado para obter artefatos do domínio. Nesta fase, as relações existentes entre os domínios candidatos e os elementos externos ao domínio são analisadas, e a variabilidade destas relações são analisadas. O resultado final da análise de contexto deve ser documentado em um modelo de contexto. O modelo de contexto define os limites do domínio, ou seja, o escopo do domínio que será modelado na próxima fase.

3.2 Modelagem de Domínio

Na fase de modelagem de domínio, as características comuns e variáveis entre as aplicações são analisadas, e vários modelos cobrindo diferentes aspectos do domínio são produzidos. A fase de modelagem de domínio consiste nas atividades: (i) `Análise de Características`, (ii) `Modelagem Entidade-Relacionamento` e (iii) `Análise Funcional`.

3.2.1 Análise de Características

O propósito da análise de características é representar em um modelo as características comuns e variáveis do domínio, seus relacionamentos e a semântica entre elas. Um modelo de características consiste em um diagrama de características classificadas em mandatórias, opcionais ou alternativas, e algumas informações adicionais das características, tais como descrições, restrições, dependências, lógica de uso, etc. A análise de características é a atividade principal do método FODA.

O processo de análise de características consiste nas fases de: (i) identificação das características, (ii) abstração e classificação das características como um modelo, e definição das características (modelagem de características), e (iii) validação do modelo. O resultado final da análise de características é um modelo de características validado.

3.2.2 Modelagem Entidade-Relacionamento

A modelagem entidade-relacionamento ([7]) captura e define o conhecimento do domínio que é essencial para a implementação de aplicações naquele domínio. O propósito da modelagem entidade-relacionamento é representar o conhecimento do domínio explicitamente, em termos de entidades do domínio e seus relacionamentos. O resultado final desta atividade é o modelo entidade-relacionamento.

3.2.3 Análise Funcional

A análise funcional identifica funcionalidades comuns e variáveis das aplicações dentro do domínio. As funcionalidades comuns são abstraídas em um modelo funcional, a partir do qual modelos de aplicações específicas podem ser instanciados. O resultado final desta atividade é o modelo funcional. As especificações de um modelo funcional podem ser classificadas em duas categorias principais: especificações de funções e especificações de comportamento. As especificações de funções são normalmente representadas por diagramas de atividades, e especificações de comportamento são representadas por diagramas de estados. O diagrama de casos de uso também pode ser utilizado para representar um modelo funcional.

4 Sistema Odyssey

O sistema `Odyssey`, desenvolvido na COPPE/UFRJ[3], é um ambiente de desenvolvimento de software cujo objetivo é construir uma infra-estrutura para reutilização baseada em modelagem de domínio, linhas de produto e desenvolvimento baseado em componentes ([11]). A versão atual do Odyssey denomina-se `OdysseyLight`, que é um ambiente de carga dinâmica de ferramentas, que contém apenas as funcionalidades de apoio às atividades de Engenharia de Domínio e Engenharia de Aplicação.

As ferramentas do núcleo do OdysseyLight são:

[3] Coordenação dos Programas de Pós-graduação em Engenharia da Universidade Federal do Rio de Janeiro

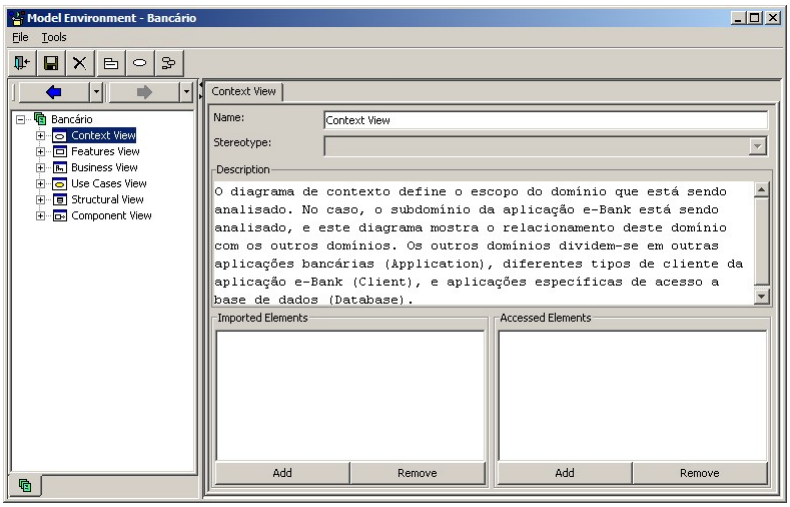

Figura 6: Modelos de Engenharia de Domínio no Odyssey

- Editor de Diagramas UML[4]
- Editor de Diagramas de Características
- Navegador Inteligente
- Gerador de Aplicações
- Máquina de Processos
- Suporte a Desenvolvimento Baseado em Componentes
- Persistência Objeto-Relacional
- Documentação de Artefatos de Software

As atividades de criação de um novo domínio ou aplicação podem ser feitas através da instanciação de um processo previamente carregado no sistema (`CBD-Arch-DE`) ou diretamente, através da criação de diferentes modelos. Os tipos de modelo de domínio são (Figura 6): modelo de contextos (*Context View*), modelo de características (*Features View*), modelo de casos de uso (*Use Cases View*), modelo de classes (*Structural View*), modelo de tipos de negócio (*Business View*), modelos comportamentais (diagramas de sequência e de estado na *Use Cases View*) e modelo de componentes (*Component View*). O processo completo de engenharia de domínio e de aplicação envolve a criação de um domínio, a modelagem do domínio através da criação dos diferentes modelos, a criação de uma ou mais aplicações instanciadas a partir deste domínio, e a modelagem da aplicação.

[4] *Unified Modeling Language*

A ferramenta Odyssey foi escolhida para ser utilizada no estudo de caso em análise de domínio, por ser uma ferramenta que incorpora a modelagem de características à modelagem UML dentro de seu processo de engenharia de domínio, permitindo apoiar as várias atividades do método FODA (Seção 3).

5 Estudo de Caso em Análise de Domínio

O propósito deste estudo de caso é adquirir conhecimento e experiência em análise de domínio, aplicados num caso real. Em particular, o estudo de caso foi realizado em uma empresa que já pratica o desenvolvimento baseado em componentes. O objetivo principal deste estudo é identificar problemas no desenvolvimento de aplicações diferentes em um mesmo domínio, e apresentar soluções em que o uso de enfoque de linha de produto possam beneficiar este tipo de desenvolvimento.

5.1 Ambiente do Estudo de Caso

O ambiente escolhido para este estudo de caso foi o de uma empresa de médio porte que atua no desenvolvimento de aplicações bancárias. Esta empresa desenvolve uma família de produtos de software que trabalham de forma integrada para atender a necessidades específicas de seus clientes, que são bancos e outras instituições financeiras. Todas estas aplicações pertencem a um mesmo domínio, ou seja, o domínio de aplicações bancárias. A empresa já utiliza o paradigma de desenvolvimento baseado em componentes, mas não utiliza o enfoque de linha de produtos. Por ser o domínio de aplicações bancárias muito amplo, e utilizando o enfoque de linha de produto descrito na Seção 2.2, o escopo para a realização do estudo de caso foi baseado em produtos já existentes, e não no domínio.

Assim, o escopo escolhido para o estudo de caso foi o do produto e-Bank. O `e-Bank` é um sistema que oferece serviços de negócio para uma agência bancária, tais como serviços de conta corrente, aplicações financeiras, transferência para outros bancos e outros.

O sistema e-Bank se integra com vários outros sistemas através de interfaces bem-definidas, como por exemplo, o sistema de Renda Fixa, de Conta Corrente, etc. O desenvolvimento é feito no sistema operacional Windows, implementando o modelo de componentes `Cosmos` ([14]) através da linguagem Java, e as ferramentas de desenvolvimento e apoio utilizadas são o ambiente Eclipse ([17]), a ferramenta de controle de versões CVS[5] e a ferramenta de modelagem GME[6] ([24]).

5.2 Método e Ferramentas Utilizados

O método de análise de domínio que foi utilizado neste estudo de caso foi o método FODA. Este método foi selecionado por ser, dentre os métodos existentes, o mais maduro e bem documentado. O método FODA inclui várias atividades que devem ser executadas, que têm como resultado um conjunto de modelos, como apresentado na Seção 3.

[5] *Concurrent Versions System*
[6] *Generic Modeling Environment*

Como ferramenta de suporte à modelagem foi utilizado o Odyssey (Seção 4), por ser uma ferramenta acadêmica que possui integração entre as atividades de engenharia de domínio e engenharia de aplicação, com alta disponibilidade e acesso à equipe de desenvolvimento e suporte. Esta ferramenta encontra-se em desenvolvimento, sendo que a versão utilizada para este estudo de caso foi a Odyssey Light 1.3.0. Esta não é a última versão da ferramenta, mas foi selecionada devido ao fato de a versão mais recente (1.4.0) ter apresentado problemas quanto à serialização dos modelos gerados.

5.3 Descrição do Estudo de Caso

O estudo de caso consistiu na análise do domínio do produto e-Bank, utilizando o método FODA e a ferramenta Odyssey para a geração dos modelos em cada uma das atividades do FODA. A análise de domínio foi integralmente realizada nas dependências da empresa selecionada, através de entrevistas com especialistas nas aplicações e especialistas no negócio. Foram também feitas consultas à documentação existente e ao próprio sistema e-Bank. A modelagem foi feita iterativamente, à medida que o conhecimento nas aplicações ia sendo aprofundado. Durante o estudo de caso, as seguintes atividades foram realizadas, segundo o método FODA (Figura 5):

- Análise de Contexto
- Análise de Características
- Modelagem Entidade-Relacionamento
- Análise Funcional

Cada uma destas atividades apresentou como resultado um modelo, gerado pela ferramenta Odyssey. As próximas seções descrevem cada uma das atividades realizadas no estudo de caso, e apresenta os modelos gerados.

5.3.1 Análise de Contexto

A primeira atividade do método FODA é a análise de contexto (Seção 3.1), que consiste na definição do escopo que será trabalhado na análise do domínio. O resultado principal desta atividade é a geração do modelo de contexto, que define os limites do domínio, ou seja, o escopo do domínio que será analisado. Neste estudo de caso, foi utilizado o enfoque de linha de produto, selecionando-se o escopo de um produto, no caso o do sistema e-Bank. O sistema `e-Bank` envolve várias funcionalidades operacionais ligadas a caixa e serviços de retaguarda de agência bancária. Esse sistema se relaciona com vários outros sistemas desenvolvidos pela empresa. O modelo de contexto (Figura 7) apresenta o recorte do que estará sendo analisado, delimitando os limites entre o escopo do sistema e-Bank e os demais sistemas que a ele estão associados. No caso, o sistema e-Bank está sendo analisado, e este modelo mostra o relacionamento deste sistema com entidades externas a ele, que não são o foco do estudo. As entidades externas dividem-se em: outros sistemas bancários (*System*), diferentes tipos de cliente do sistema e-Bank (*Client*), e aplicações somente de acesso a banco de dados (*Database).*

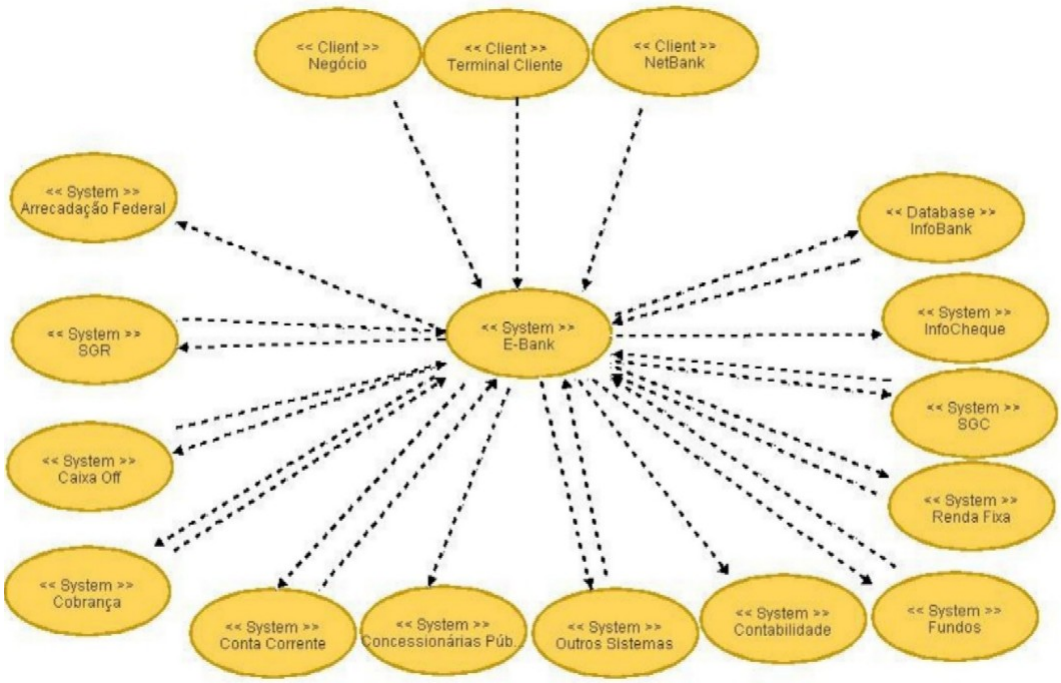

Figura 7: Modelo de Contexto do e-Bank

5.3.2 Análise de Características

Após ser definido o contexto da análise de domínio, foi realizada a primeira sub-atividade da modelagem do domínio, a análise de características (Seção 3.2.1), na qual foram identificadas as características do sistema e-Bank. Em nosso estudo, utilizamos a definição de Kang ([22]) para característica, ou seja, as características são atributos do sistema que afetam diretamente seus usuários finais. Para este estudo, somente características funcionais foram consideradas. Mediante consulta à documentação e entrevista com especialistas no negócio, foram levantadas as características principais do sistema e-Bank, classificadas em mandatórias e opcionais. Atualmente, os sistemas desenvolvidos na empresa não têm configurações diferentes para atender a diferentes tipos de cliente. Por este motivo, o levantamento de características opcionais, que podem ou não fazer parte do produto dependendo das necessidades específicas do cliente, foi feito através de consulta a especialistas no negócio. O modelo de características (Figura 8) apresenta as características principais do sistema e-Bank, sob o ponto-de-vista do usuário final. As características mandatórias são representadas por linha contínua e as características opcionais são representadas por linha tracejada. Assim, por exemplo, as características de `Fundo` e `Renda Fixa` e todas as suas subcaracterísticas foram classificadas como opcionais. Por outro lado, todas as características associadas a `Movimento` e `SPB` são obrigatórias. Não foram identificadas características alternativas.

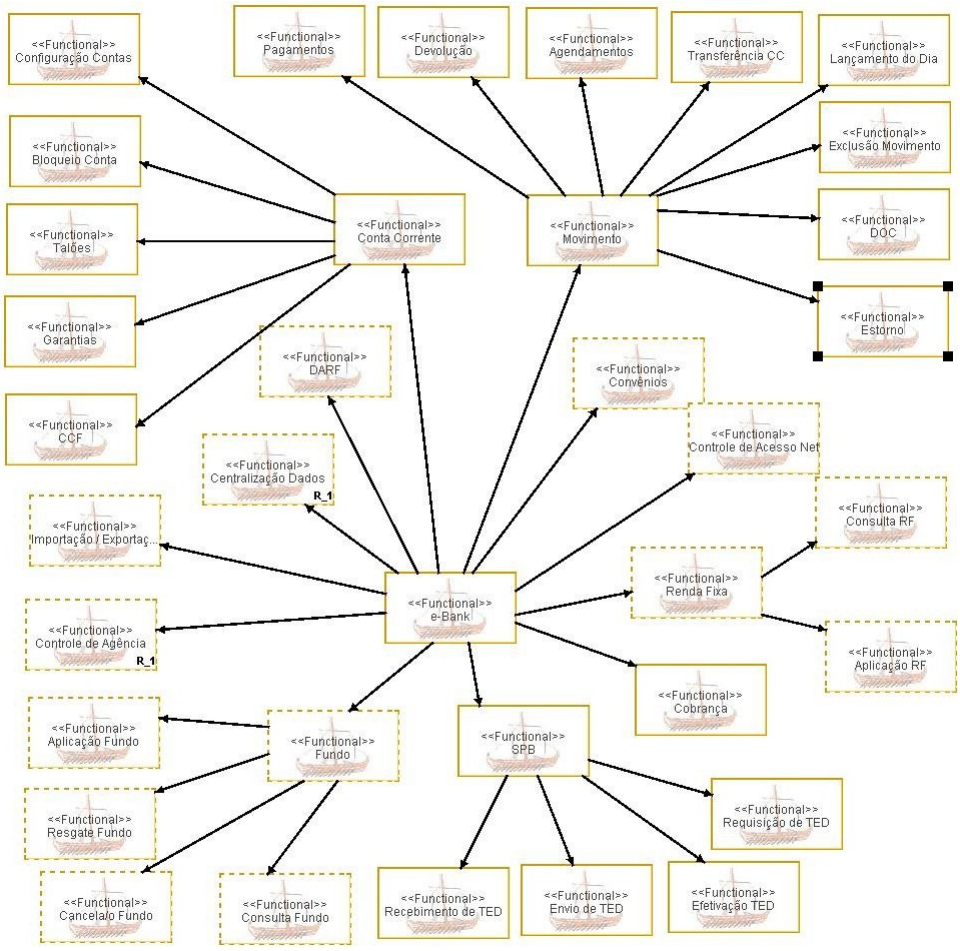

Figura 8: Modelo de Características do e-Bank

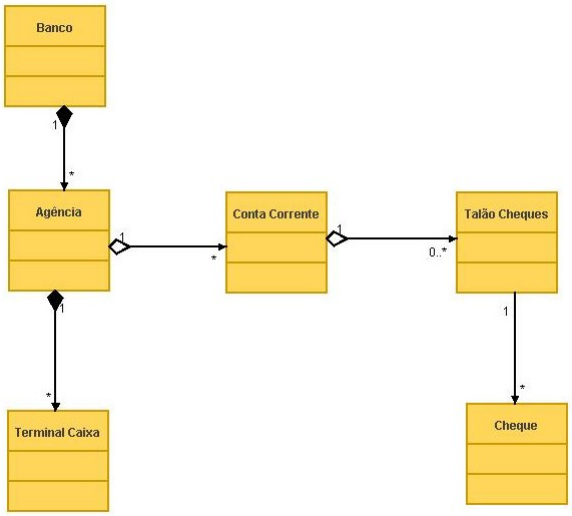

Figura 9: Entidade-Relacionamento de Agência

5.3.3 Modelagem Entidade-Relacionamento

Outra sub-atividade da modelagem do domínio é a modelagem entidade-relacionamento. O propósito da modelagem entidade-relacionamento (Seção 3.2.2) é representar o conhecimento do domínio por meio de entidades do domínio e seus relacionamentos.

Para o estudo de caso, foi feita consulta a documentos tais como modelo de dados e modelo estrutural, e mediante entrevista com especialistas na aplicação foram levantados as principais entidades do sistema e-Bank e o relacionameno entre elas. O modelo entidade-relacionamento do sistema e-Bank foi representado através de diagramas de classes, na visão estrutural (*Structural View*) do Odyssey. Foram utilizados diagramas de classes devido ao fato de diagramas de entidade-relacionamento não serem representados no Odyssey. O modelo foi dividido em três diagramas, para melhor compreensão, de acordo com o tipo de entidade-relacionamento que se quis representar em cada um deles: de `Agência` (Figura 9), `de Cliente` (Figura 10) e de `Movimento` (Figura 11). O `Movimento` é um transação de `Crédito` ou `Débito`. Existem movimentos que são somente de `Débito` (`Pagto Tributo`, `Pago Título`) e movimentos que envolvem `Crédito` e `Débito` (`Transferência`, `DOC`, `TED`) envolvendo uma ou mais `Agências` ou `Bancos`.

5.3.4 Análise Funcional

A sub-atividade final da modelagem de domínio é a análise funcional (Seção 3.2.3). Os modelos de características e entidade-relacionamento, gerados nas sub-atividades anteriores, foram utilizados como base para a construção do modelo funcional. A especificação de um modelo funcional pode ser classificada em duas categorias principais: especificação das funcionalidades e especificação do comportamento do sistema. Normalmente, a especificação funcional é representada por diagramas de atividade, e a especificação do comportamento,

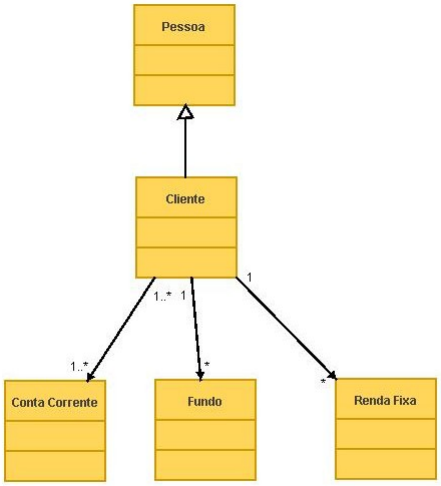

Figura 10: Entidade-Relacionamento de Cliente

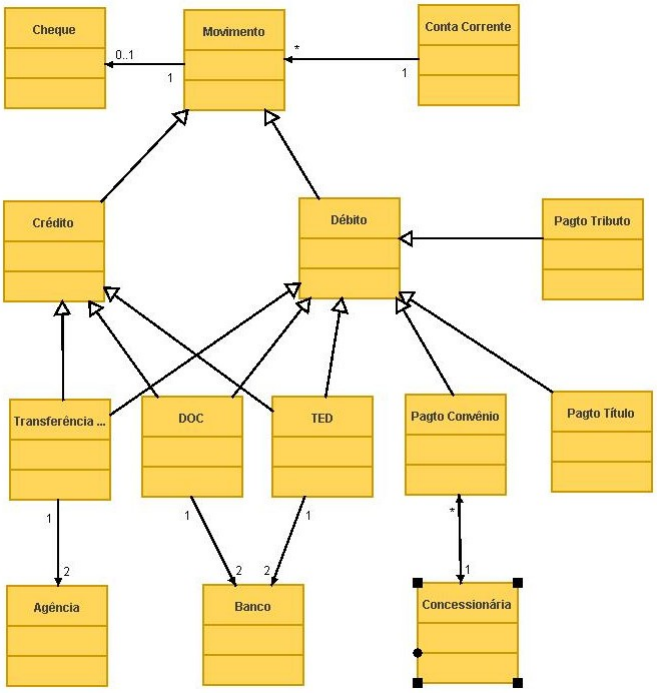

Figura 11: Entidade-Relacionamento de Movimento

por diagrama de estado. Devido a limitações do ambiente Odyssey, que não incorpora estes diagramas, e também a limitações de tempo para fazer uma análise funcional mais detalhada, o modelo funcional do sistema e-Bank foi representado através de diagramas de casos de uso, na visão de caso de uso (*Use Cases View*) do Odyssey. Os casos de uso representam funcionalidades que podem ser executadas por determinados tipos de usuário do sistema e-Bank. Os casos de uso foram classificados em: (i) `Casos de Uso de Negócio` (Figura 12), representando as funcionalidades disponíveis para funcionários da retaguarda da agência bancária; (ii) `Casos de Uso de Caixa` (Figura 13), representando as funcionalidades disponíveis para funcionários que trabalham como caixa da agência bancária; (iii) `Casos de Uso de Cliente` (Figura 14), englobando as funcionalidades disponíveis para clientes da agência bancária, seja através de Internet Banking, ou através de caixa automática; (iv) `Casos de Uso de Apoio`, referente às funcionalidades de entrada e saída de movimentos na agência bancária (Figura 15); (v) `Casos de Uso de Cadastro`, referente ao cadastro de informações no sistema (Figura 16); (vi) `Casos de Uso de Segurança`, relativos a funções de acesso seguro ao sistema (Figura 17); (vii) `Casos de Uso de Jurídico`, referente a funcionalidades envolvendo aspectos judiciais do sistema (Figura 18); e (viii) `Casos de Uso de Compensação`, referente a funcionalidades de compensação de cheques entre bancos (Figura 19).

5.4 Análise dos Resultados

O resultado principal deste trabalho são os modelos gerados pelo Odyssey, como apresentado na Seção 5.3. A seguir são feitas algumas considerações sobre a análise do sistema e-Bank atual, e como a utilização do enfoque de linha de produto pode favorecer o desenvolvimento deste e outros produtos de software da mesma empresa.

A empresa selecionada desenvolve e mantém vários produtos de software dentro do domínio de aplicações bancárias, atendendo a diferentes clientes. O sistema e-Bank, embora atenda a diferentes requisitos de diferentes clientes, é desenvolvido e mantido como um sistema único. Optou-se por esta forma de tratamento do produto devido à dificuldade de configuração e manutenção de diferentes versões do mesmo produto, tanto pela ausência de um processo sistemático quanto de ferramentas. Deste modo, quando uma mudança de requisito é solicitada por um cliente, a funcionalidade alterada é incorporada ao e-Bank e todos os clientes recebem uma nova versão do produto, mesmo que o requisito seja necessário para somente um cliente. Com isso, muitos clientes têm funcionalidades que não necessitam, e todos os clientes pagam o mesmo preço pelo produto.

Uma outra constatação que foi feita durante o estudo foi a do baixo grau de reutilização dos componentes. Apesar da empresa utilizar o paradigma de desenvolvimento baseado em componentes, atualmente não há reutilização de componentes de negócio entre os diferentes produtos de software. Existe somente o reuso de componentes de suporte e infra-estrutura, tais como acesso a sistema de arquivos, componentes utilitários, etc. Como os produtos de software da empresa têm alto grau de similaridade entre si, um mesmo componente de negócio poderia ser utilizado em mais de um produto de software (por exemplo, no sistema `e-Bank` e no sistema `Conta Corrente`). Para que tal grau de reuso seja atingido, é necessário que os componentes de negócio apresentem pontos de variação, permitindo a configuração de

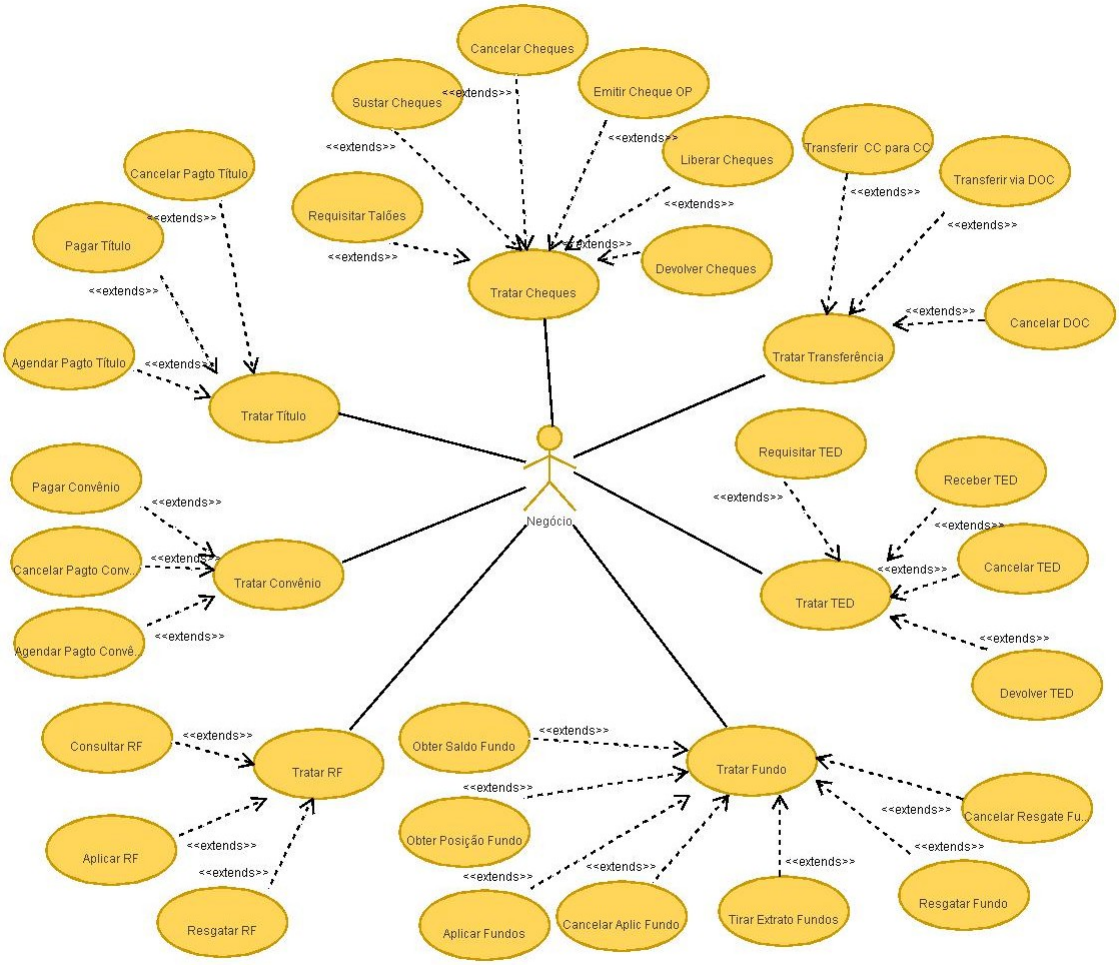

Figura 12: Casos de Uso de Negócio

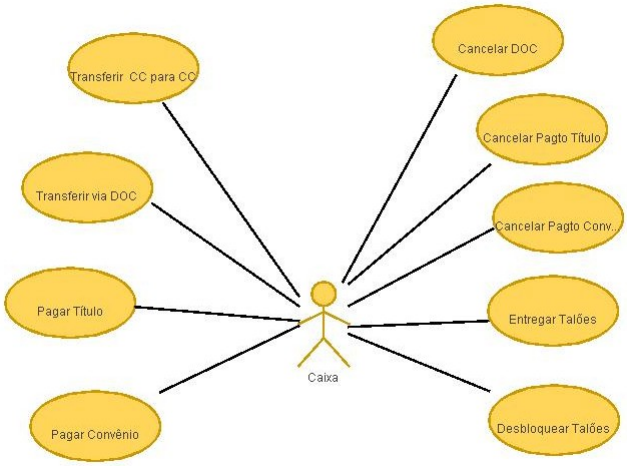

Figura 13: Casos de Uso de Caixa

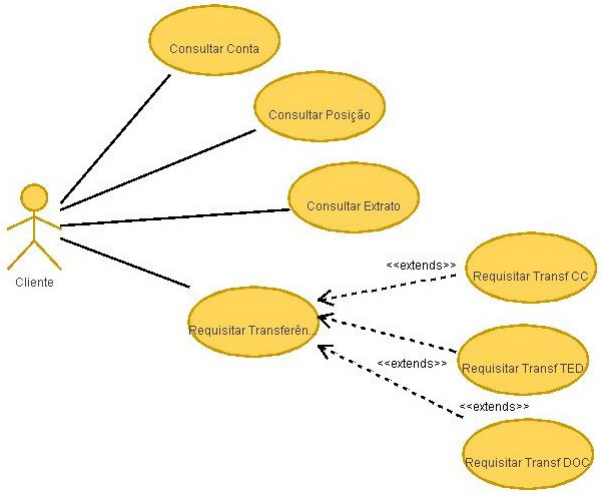

Figura 14: Casos de Uso de Cliente

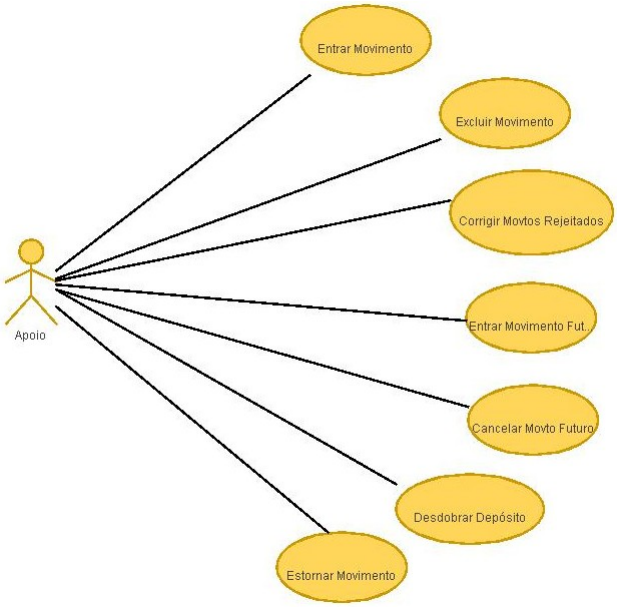

Figura 15: Casos de Uso de Apoio

Figura 16: Casos de Uso de Cadastro

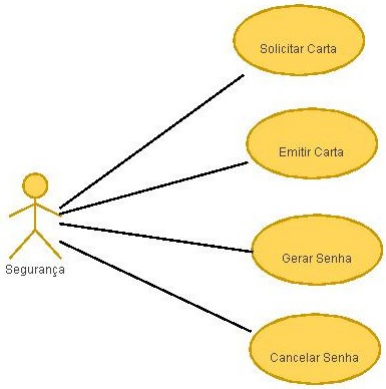

Figura 17: Casos de Uso de Segurança

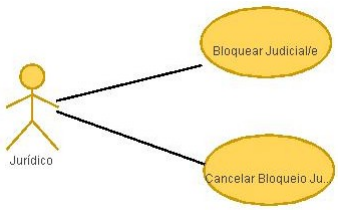

Figura 18: Casos de Uso de Jurídico

Figura 19: Casos de Uso de Compensação

diferentes variantes (Seção 2.6). Como as equipes que desenvolvem cada um dos produtos de software são distintas e não são utilizados métodos de análise de domínio e linha de produto, as similaridades e variabilidades entre os produtos não são exploradas, impedindo um maior grau de reutilização dos artefatos de software em todas as fases do desenvolvimento.

Um problema levantado pelos especialistas foi o de que não existe ferramenta adequada para a gerência de configuração de sistemas baseados em componentes. Embora estejam sendo utilizados o modelo de evolução e processo sistemático apresentados em [25], estes ainda apresentam várias limitações e restrições, principalmente por não existir ainda ferramenta que dê suporte à evolução de sistemas baseados em componentes. Atualmente, para gerar uma nova versão de componente são necessárias várias etapas, sendo que nem todas estão integradas de forma automatizada.

A utilização do enfoque de engenharia de linha de produto neste ambiente, com a utilização de processos e ferramentas adequados, traz benefícios em termos de produtividade, configurabilidade e negócios. Em produtividade, porque o esforço de desenvolvimento e manutenção de componentes de negócio do domínio pode ser amortizado entre os vários produtos. Em termos de configurabilidade, com a utilização de uma ferramenta de gerência de configuração voltada ao desenvolvimento baseado em componentes, é possível criar e manter diferentes configurações de diferentes produtos construídos a partir dos mesmos componentes de negócio do domínio. A utilização de ferramenta de gerência de configuração de componentes traz grandes benefícios também em termos de negócios, porque necessidades específicas dos clientes poderiam ser atendidas, estabelecendo preços diferentes de acordo com estas necessidades, ampliando as oportunidades de negócios. Uma outra vantagem de negócios obtida em decorrência é a escalabilidade. Deste modo, um cliente pode inicialmente comprar uma versão básica de um produto, e incorporar novas funcionalidades de forma incremental, podendo desta forma planejar a médio e longo prazo os seus dispêndios com aquisições de software.

Dependendo do modelo de negócios da empresa, até mesmo componentes individuais

podem ser comercializados como COTS[7], atendendo a necessidades específicas de clientes, permitindo a integração com sistemas já existentes.

6 Trabalhos Relacionados

`RE-PLACE` (*Reengineering-Enabled Produt Line Architecture Creation and Evolution*) é um método para ajudar organizações a transicionar para o enfoque de linha de produto de software através da reengenharia da arquitetura, visando o suporte à evolução de aplicações existentes e possibilitando incorporar novas aplicações ([4]). A idéia principal deste método é o conceito de quadro negro (*blackboard*), que é um espaço de trabalho compartilhado que permite que as atividades de reengenharia e linha de produto troquem informações, formando uma base de dados de conhecimento acumulado, utilizada durante o processo.

Kang apresenta um processo de reengenharia de sistemas legados visando a transição para linha de produto de software utilizando conceitos de análise de domínio presentes no método FODA ([23]). Primeiramente, através de engenharia reversa, a arquitetura e os componentes das aplicações legadas são recuperados. Em seguida, utilizando-se as informações recuperadas e o conhecimento do domínio, é reconstruído o modelo de características para as aplicações legadas. O modelo de características é refinado baseado em antecipação de mudanças em oportunidades de negócio ou em tecnologia. Finalmente, utilizando o modelo de características refinado e princípios de desenvolvimento de software, a nova arquitetura e os componentes das aplicações são reprojetados.

Prado apresenta uma estratégia de `Reengenharia Baseada em Componentes` (RBC), utilizando como base o código legado de um sistema escrito em linguagem procedural, e tendo como resultado sua reimplementação utilizando uma linguagem orientada a componentes ([15]). As principais atividades deste processo são: organizar código legado, recuperar projeto do sistema atual, reespecificar, reprojetar e reimplementar o sistema. As atividades de reengenharia deste processo são apoiadas por um sistema de transformação de software (Draco-PUC) e uma ferramenta CASE[8] orientada a componentes (MVCASE).

7 Conclusões e Trabalhos Futuros

Este estudo foi realizado com a finalidade de obter familiaridade com métodos e ferramentas de análise de domínio e ferramenta, utilizando-os em uma empresa que desenvolve produtos de software com alto grau de similaridade, utilizando o DBC. O objetivo principal é o levantamento dos requisitos necessários para implantação do enfoque de linha de produto de software no processo de desenvolvimento de software da empresa.

Com o estudo de caso em análise de domínio e seus resultados, levantaram-se alguns problemas ou limitações existentes no processo de desenvolvimento atual, tanto nas áreas de engenharia de software quanto em gerenciamento de negócios, conforme apresentado na Seção 5.4. Na mesma seção, foram apresentadas formas com que o enfoque de linha de produto poderia beneficiar o desenvolvimento de produtos de software similares.

[7] *Commercial Off-The-Shelf*
[8] *Computer-Aided Software Engineering*

Os resultados apresentados neste estudo irão guiar pesquisa futura em linha de produto, arquitetura de software e engenharia de software baseada em componentes. construção de linha de produtos a partir de sistemas baseados em componentes. A idéia central desta pesquisa é conceber um método prescritivo para a construção e evolução de linhas de produto a partir de produtos de software já existentes, baseados em componentes. O método proposto também deverá propor soluções para a questão de evolução e gerência de configuração de linhas de produtos, que tornam-se mais complexas devido à existência de um conjunto de produtos fortemente dependentes entre si através do reuso de componentes comuns cujas evoluções podem ocorrer de forma independente dos produtos gerados pela linha.

Para atingir os objetivos globais deste projeto de pesquisa, alguns trabalhos já estão sendo realizados.

Um destes trabalhos em andamento é o desenvolvimento de um metamodelo agregando conceitos de características (Seção 2.3) e variabilidade (Seção 2.6) ao desenvolvimento de software baseado em componentes e centrado na arquitetura de software. Através deste metamodelo, definiu-se conceitos tais como arquitetura de linha de produto e componente variante, necessários ao desenvolvimento baseado em linhas de produto. Definiu-se também uma forma de mapeamento entre as características e os pontos de variação associados aos componentes. No mesmo trabalho, definiu-se um método prescritivo de tratamento destes conceitos de variabilidade dentro de um processo de desenvolvimento baseado em componentes ([26]).

Como continuidade deste trabalho, será elaborado um modelo de arquitetura de linha de produto para os produtos da empresa pesquisada, utilizando o metamodelo e o método desenvolvidos. Talvez seja necessário, nesta etapa, realizar um reprojeto da arquitetura dos sistemas, visando separar as partes comuns das partes variáveis entre os produtos, pois hoje a empresa utiliza o enfoque de produto único. O trabalho de pesquisa incluirá métodos e heurísticas para mapeamento destas partes comuns, que possam auxiliar na rearquitetura.

Com o modelo de arquitetura de linha de produto e o modelo de características, utilizando-se o método de mapeamento descrito em , é possível instanciar cada um dos produtos da linha de produto através da seleção de características. Um outro trabalho futuro é a extensão dos conceitos de variabilidade na ferramenta `Bellatrix` ([36]), que implementa o modelo `Cosmos` ([14]), para permitir a geração automática do código relativo à arquitetura de componentes do produto resultante da seleção de características.

Um outro trabalho que já está sendo realizado, na área de evolução de sistemas baseados em componentes, é o `CosmosLoader` ([18]). O `CosmosLoader` é uma ferramenta que permite a carga dinâmica de versões específicas de uma configuração de sistema baseado em componentes utilizando o modelo Cosmos. A configuração do sistema é representada por um arquivo XML[9], e os componentes a serem carregados são obtidos de um repositório de componentes. Tal ferramenta pode ser integrada futuramente com outras ferramentas e métodos sendo desenvolvidos, permitindo a carga dinâmica de uma determinada instância de produto da linha de produtos, através da sua configuração de componentes.

Tais trabalhos integram a linha de pesquisa de vários alunos de pós-graduação da Unicamp, e juntos visam apresentar métodos, processos e ferramentas inovadores na área de

[9] *eXtensible Markup Language*

engenharia de software baseada em componentes.

Referências

[1] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel. *Component-based Product Line Engineering with UML*. Addison-Wesley, 2002.

[2] Leonor Barroca, Itana M. S. Gimenes, and Elisa H. M. Huzita. *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*, chapter 1, pages 1–26. Itana M. S. Gimenes and Elisa H. M. Huzita, 2005.

[3] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schimid, Tanya Widen, and Jean-Marc DeBaud. PuLSE: A Methodology to Develop Software Product Lines. *Symposium on Software Reusability´99 (SSR´99)*, 1999.

[4] Joachim Bayer, Jean-François Girard, Martin Würthner, Jean-Marc DeBaud, and Martin Apel. Transitioning legacy assets to a product line architecture. In *7th European Software Engineering Conference in 7th ACM SIGSOFT Foundations of Software Engineering (ESEC/FSE-7)*, pages 446–463, 1999.

[5] Joachim Bayer, Dirk Muthig, and Brigitte Göpfert. The Library Systems Product Line - A KobrA Case Study. *IESE-Report No. 024.01/E, Version 1.0*, 2001.

[6] Jan Bosch. Software Product Families in Nokia. In *Software Product Lines: 9th International Conference (SPLC 2005), Rennes, France*, pages 2–6, 2005.

[7] Peter Pin-Shan Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[8] Adam Childs, Jesse Greenwald, Georg Jung, Matthew Hoosier, and John Hatcliff. CALM and Cadena: Metamodeling for Component-Based Product-Line Development. *IEEE Computer*, 39(2):42–50, 2006.

[9] Paul Clements and Linda Northrop. Salion, Inc.: A Software Product Line Case Study. *CMU/SEI-2002-TR-038 ESC-TR-2002-038*, 2002.

[10] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[11] COPPE/UFRJ. The Odyssey Project. `http://reuse.cos.ufrj.br/odyssey/index_en.html/`.

[12] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[13] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing Cardinality-based Feature Models and their Specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[14] Moacir Caetano da Silva Jr. Cosmos - Um Modelo de Estruturação de Componentes para Sistemas Orientados a Objetos. Master's thesis, Unicamp, 2003.

[15] Antonio Francisco do Prado. *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*, chapter 5, pages 105–166. Itana M. S. Gimenes and Elisa H. M. Huzita, 2005.

[16] Desmond F. D'Souza and Alan C. Wills. *Objects, Components, and Frameworks with UML : The Catalysis Approach.* Addison-Wesley Publishing Company, 1998.

[17] Eclipse Consortium. Eclipse.org main page. `http://www.eclipse.org/`, 2003.

[18] Leonel Aguilar Gayard, Paulo Astério de Castro Guerra, Ana Elisa de Campos Lobo, and Cecília Mary Fischer Rubira. Automated Deployment of Component Architectures with Versioned Components. In *Eleventh International Workshop on Component-Oriented Programming (WCOP), in 20th European Conference on Object-Oriented Programming (ECOOP)*, 2006.

[19] Martin L. Griss, John Favaro, and Massimo d'Alessandro. Integrating Feature Modeling with the RSEB. *The Fifty International Conference on Sofware Reuse (ICSR) IEEE Computer Society*, pages 76–85, 1998.

[20] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the Notion of Variability in Software Product Lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, 2001.

[21] Maarit Harsu. FAST product-line architecture process. Technical report, Report 29, Institute of Software Systems, Tampere University of Technology, 2002.

[22] Kyo Kang, Sholom Cohen, James Hess, William Novak, and Spencer Peterson. Feature-Oriented Domain Analysis (FODA) feasibility study. *CMU/SEI-90-TR-21 ESD-90-TR-222*, 1990.

[23] Kyo Chul Kang, Moonzoo Kim, Jaejoon Lee, and Byungkil Kim. Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets - a Case Study. In Klaus Pohl Henk Obbink, editor, *Software Product Lines: 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005. Proceedings*, volume 3714 / 2005, pages 45–56, 2005.

[24] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The Generic Modeling Environment. In *Proceedings of WISP'2001, Budapest, Hungary.*, 2001.

[25] Ana Elisa Lobo, Paulo Astério Guerra, Cecília Rubira, and Fernando Castor. A Systematic Approach for the Evolution of Reusable Software Components. In *Workshop on Architecture-Centric Evolution*, 2005.

[26] Ana Elisa C. Lobo, Patrick H. da S. Brito, and Cecília M. F. Rubira. Managing Variabilities in Component-Based Software Product Lines. In *Sent to WDBC 2006*, 2006.

[27] Dirk Muthig, Isabel John, Michalis Anastasopoulos, Thomas Forster, Jörg Dörr, and Klaus Schmids. GoPhone - A Software Product Line in the Mobile Phone Domain. *IESE-Report No. 025.04/E, Version 1.0, March 5*, S 2004.

[28] James M. Neighbors. *Software Construction Using Components*. PhD thesis, Department of Information and Computer Science University of California, Irvine, 1980.

[29] Regiane Oliveira, Ana Paula Blois, Aline Vasconcelos, and Cláudia Werner. Representação de Variabilidades em Componentes de Negócio no Contexto da Engenharia de Domínio. In *Workshop de Desenvolvimento Baseado em Componentes*, 2005.

[30] Bernhard Westfechtel Reidar Conradi. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, 1998.

[31] Mark A. Simos. Organization domain modeling (ODM): formalizing the core domain modeling life cycle. *SIGSOFT Softw. Eng. Notes*, 20(SI):196–205, 1995.

[32] Software Engineering Institute. Domain Engineering and Domain Analysis. `http://www.sei.cmu.edu/str/descriptions/deda_body.html/`, 1997.

[33] Software Engineering Institute. Model-Based Software Engineering. `http://www.sei.cmu.edu/mbse/`, 1997.

[34] Mirjam Steger, Christian Tischer, Birgit Boss, Andreas Müller, Oliver Pertler, Wolfgang Stolz, and Stefan Ferber. Introducing PLA at Bosch Gasoline Systems: Experiences and Practices. In *Software Product Line Conference (SPLC)*, 2004.

[35] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[36] Rodrigo Teruo Tomita. Bellatrix: Um Ambiente para Suporte Arquitetural ao Desenvolvimento Baseado em Componentes. Master's thesis, Universidade Estadual de Campinas, 2006.

[37] Will Tracz. DSSA (Domain-Specific Software Architecture): pedagogical example. *SIGSOFT Softw. Eng. Notes*, 20(3):49–62, 1995.

[38] Martin Verlage and Thomas Kiesgen. Five years of product line engineering in a small company. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 534–543, New York, NY, USA, 2005. ACM Press.

[39] Cláudia Werner, Marco Mangan, Leonardo Murta, Robson Pinheiro, Marta Mattoso, Regina Braga, and Marcos Borges. OdysseyShare: an Environment for Collaborative Component-Based Development. In *IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, Nevada, USA*, 2003.

[40] Weishan Zhang and Stan Jarzabek. Reuse without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices. In *Software Product Lines: 9th International Conference (SPLC 2005)*, volume 3714 / 2005, pages 57–69, 2005.