Mineração de Componentes para a Revitalização de Softwares Embutidos

Marcelo A. Ramos[1,2], Rosângela A. D. Penteado[2]

[1]ADC/LAC - Centro de Desenvolvimento de Aplicações para a América Latina
VeriFone do Brasil
São Paulo, SP, Brasil

[2]DC/UFSCar - Departamento de Computação
Universidade Federal de São Carlos
Rod. Washington Luís, Km 235
Caixa Postal 676, CEP 13565-905, São Carlos, SP, Brasil
marcelo_r1@verifone.com, {marcelo_ramos, rosangel}@dc.ufscar.br

Abstract. *In order to reduce costs, to minimize risks, to anticipate deadlines and to optimize resources of projects of new products it must, whenever possible, to reuse artifacts of existing similar products. These artifacts must be of easy adaptation to meet the requirements of the new products. This paper proposes the use of mining of generic software components from embedded legacy systems to collect and to document the knowledge contained in such systems, to achieve their revitalization and to produce a core of reusable assets to support the fast development of similar products. Software Product Line techniques are used for domain modeling and components design. A case study for POS (Point of Sale) terminals domain is presented.*

Resumo. *Para reduzir custos, minimizar riscos, antecipar prazos e otimizar recursos de projetos de novos produtos deve-se, sempre que possível, reutilizar artefatos de produtos similares existentes. Esses artefatos devem ser de fácil adaptação para satisfazerem aos requisitos dos novos produtos. Este trabalho propõe o uso de mineração de componentes genéricos de software a partir de sistemas embutidos legados, para reunir e documentar os conhecimentos contidos nesses sistemas, efetuar suas revitalizações e produzir um núcleo de ativos reutilizáveis para apoiar o rápido desenvolvimento de produtos similares. Técnicas de Linha de Produtos de Software são utilizadas para a modelagem de domínio e projeto de componentes. Um estudo de caso para o domínio de terminais POS (Point Of Sale) é apresentado.*

1. Introdução

No domínio de sistemas embutidos, variabilidades do hardware ocorrem muitas vezes de maneira não planejada. Componentes antigos são freqüentemente substituídos por outros mais modernos, baratos e eficientes, produzindo versões melhoradas do produto original. Possivelmente, alguns desses componentes nem existiam quando a primeira versão do produto foi entregue. Muitas empresas desse domínio desejam que a evolução de seus softwares ocorra de maneira semelhante, com a reutilização parcial ou integral dos artefatos de suas aplicações bem sucedidas, para a criação de novas aplicações com pouca ou nenhuma manutenção [Graaf, Lormans e Toetenel 2003]. Nesse contexto, a mineração de componentes a partir do código legado e a sua posterior reconexão ao sistema original é uma técnica que pode não só modernizar o código existente, mas também estender as funções do sistema, garantindo-lhe uma sobrevida.

Este trabalho propõe o uso de mineração de componentes genéricos de software a partir de sistemas embutidos legados, para reunir e documentar os conhecimentos neles contidos, efetuar suas revitalizações e produzir um núcleo de ativos[1] reutilizáveis para apoiar o rápido desenvolvimento de uma Linha de Produtos de Software (LPS). Além dessa introdução sua organização é a seguinte: A Seção 2 descreve fundamentos teóricos, presentes na literatura, relacionados ao contexto deste trabalho e relevantes para a sua realização; A Seção 3 reúne trabalhos relacionados à presente proposta; A Seção 4 apresenta um modelo de processo para a mineração de ativos para LPS, a partir de sistemas embutidos legados, que pode ser usado para apoiar a revitalização de seus códigos, como mostra a Seção 5; A Seção 6 ilustra um estudo de caso para o domínio de terminais POS (*Point Of Sale*), para exemplificar a presente proposta e, finalmente, na Seção 7 são colocadas as considerações finais e os trabalhos futuros.

2. Fundamentos Teóricos

Sistemas embutidos são sistemas computacionais especializados, hardware e software, que integram sistemas com funcionalidade mais ampla, realizando tarefas específicas e pré-definidas. Softwares escritos para sistemas embutidos são freqüentemente chamados de softwares embutidos ou *firmwares*. Apesar da variedade de projetos, esses softwares geralmente atuam em hardwares com conjunto reduzido de recursos e capacidade limitada de processamento e armazenamento de informações [Vahid e Givargis 2002].

Reuso de software se refere à construção de sistemas a partir de artefatos existentes, ao invés de desenvolvê-los a partir do zero [Krueger 1992]. Diferentes técnicas têm sido pesquisadas com a finalidade de elevar o reuso para níveis de abstração mais altos, como por exemplo: Componentes [Szyperski 2002] e Linha de Produtos de Software [Clements e Northrop 2001].

Componentes de software são elementos de composição, independentes do contexto, implementados para uma determinada especificação[2], distribuídos de maneira autônoma e que, por meio suas interfaces, agregam funções aos sistemas que integram [Szyperski 2002]. Porém, no domínio de sistemas embutidos, componentes geralmente

[1] Tradução do termo original em inglês *core assets*.
[2] Por exemplo: COM (***C**omponent **O**bject **M**odel*), EJB (***E**nterprise **J**ava **B**ean*), etc.

não são elementos autônomos (*run-time component*s), mas sim códigos escritos em linguagens de alto nível, que podem ser conectados ao código de uma aplicação durante a criação de versões do sistema (*build-time components*) [Crnkovic 2005]. Componentes com alto grau de generalidade e fácil adaptação permitem a criação de soluções reutilizáveis para requisitos similares em diferentes domínios. Para isso, devem agregar propriedades intrínsecas dos domínios aos quais se aplicam, para apoiar a implementação das variabilidades neles existentes [Bergey, O'Brien e Smith 2000].

Linha de Produtos de Software consiste em um conjunto, ou família, de produtos pertencentes a um determinado domínio, que tem como base uma arquitetura comum. Seu objetivo é ampliar a eficiência dos processos de desenvolvimento, explorando a identificação e o reuso das similaridades e controlando as variabilidades dos membros de uma família [Atkinson et al 2002]. Um processo genérico de engenharia para LPS é constituído de duas atividades principais: Engenharia de Domínio, que cria o núcleo de ativos reutilizáveis e a infra-estrutura de desenvolvimento da LPS, e Engenharia de Aplicação, que desenvolve produtos membros da família a partir desses recursos [Ziadi, Jézéquel e Fondement, 2003]. O núcleo de ativos reutilizáveis de uma LPS é composto por seus requisitos, modelos de domínio, arquiteturas, componentes de software etc.

Mineração de ativos frequentemente se refere ao processo de extração não trivial de artefatos potencialmente úteis e reutilizáveis, embutidos nos sistemas legados de uma empresa [Bergey, O'Brien e Smith 2000].

Features são propriedades de um domínio, visíveis ao usuário, que permitem identificar semelhanças e diferenças entre os sistemas de software desse domínio. Com o objetivo de auxiliar a identificação de propriedades importantes ou especiais de um domínio durante a fase de análise, Kang et al (1990) introduziram o modelo de *features* em seu método FODA (***F**eature **O**riented **D**omain **A**nalysis)*. Nesse modelo, as *features* são organizadas hierarquicamente em forma de árvore e são unidas por relacionamentos estruturais formando agrupamentos. Cada *feature* possui um especificador próprio que a define como obrigatória, opcional ou alternativa. A Figura 1 mostra um modelo de *features* que exemplifica as definições propostas por Kang et al (1990).

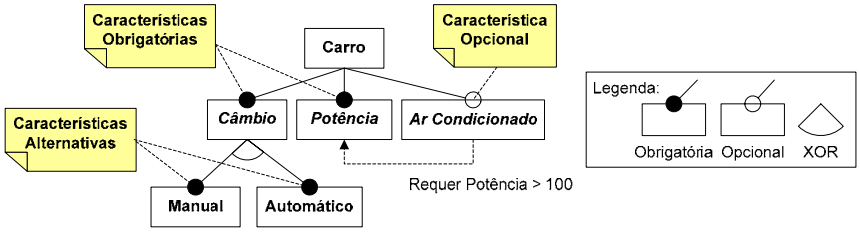

Figura 1. Modelo de *features* {Adaptado de [Kang et al 1990]}

Diferentes notações têm sido propostas para ampliar a representatividade desse modelo em relação a diferentes tipos de relacionamentos estruturais. Por exemplo, Czarnecki e Eisenecker (2000) distinguem relacionamentos XOR, entre *features* alternativas, mutuamente exclusivas, de OR, entre *features* opcionais, que podem ser combinadas entre si. Riebisch et al (2002) expandem os relacionamentos OR com a notação de multiplicidade da UML [Booch, Rumbaugh e Jacobson 1999], ampliando o número de combinações possíveis entre suas *features*. Gomaa (2004) propõe uma forma alternativa de representar as *features* de um domínio, baseada na UML.

3. Trabalhos relacionados

Muitos dos métodos para a criação de LPS propõem o planejamento antecipado de todos os produtos que podem ser derivados da infra-estrutura comum de desenvolvimento, a ser criada para uma determinada família. Assim, não mencionam claramente como prover o apoio a variabilidades não planejadas. Adicionalmente, muitos deles não citam a possibilidade e a maneira de criar LPS a partir dos artefatos legados de um domínio. Por esse motivo, não foram incluídos nessa seção.

Métodos para a criação de LPS no domínio de sistemas embutidos incluem, geralmente, uma atividade inicial de reengenharia para criar ativos reutilizáveis a partir de sistemas legados, para apoiar o desenvolvimento futuro de uma família de produtos. Um exemplo é o método FOOM (***F**eature-based **O**bject **O**riented **M**odeling*) [Ajila e Tierney 2002]. No entanto, tarefas potencialmente complexas, como a recuperação e a transformação da arquitetura, ainda são realizadas integralmente antes que uma única variabilidade, mesmo que simples, seja disponibilizada na forma de um novo produto. Um método incremental, que permita agregar gradativamente ao código legado apoio às variabilidades do domínio, com base nas necessidades imediatas e particulares de cada empresa, pode ser uma alternativa para reduzir prazos, riscos e custos associados à criação de famílias de produtos. Porém, nenhum dos métodos consultados adota essa abordagem. Alguns trabalhos, realizados em outros domínios de aplicação, podem apoiar evoluções de softwares embutidos para LPS, segundo a abordagem descrita. A seguir são mencionados alguns deles.

Mehta e Heineman (2002) propõem uma metodologia para a modernização de sistemas de software, que agrega *features*, testes de regressão e engenharia de software baseada em componentes. A proposta prevê a modernização de um conjunto de *features* identificadas durante testes de regressão. O código associado a cada *feature* é identificado, refatorado, convertido em componente de software e inserido novamente na aplicação. Os componentes criados podem ser reutilizados em outras aplicações. O'Brien (2005) observa que se os paradigmas de desenvolvimento do sistema legado e dos componentes recém criados forem distintos, *gateways* devem ser desenvolvidos para permitir a reconexão de seus códigos.

Bergey, O'Brien e Smith (2000) propõem a realização da mineração de ativos para LPS em quatro passos: 1) Reunir as informações preliminares, 2) Decidir pela mineração e definir sua estratégia, 3) Obter entendimento técnico detalhado dos ativos existentes e 4) Recuperar os ativos. Bosch e Ran (2000) consideram essencial a criação de um modelo de *features* para apoiar a realização dessa tarefa.

4. Um Modelo de Processo para a Mineração de Ativos para LPS

Comumente tecnologias tradicionais de desenvolvimento de software não consideram necessidades específicas associadas à criação de softwares embutidos e restrições usuais desse domínio, como a limitação de memória e as variações do hardware [Graaf, Lormans e Toetenel 2003].

Com base na metodologia de Mehta e Heineman (2002), o modelo de processo proposto prevê a criação de componentes de software a partir de *features* potencialmente úteis, contidas em sistemas embutidos legados. No entanto, o processo de extração de *features* utilizado é resultante de adaptações feitas aos quatro passos

propostos por Bergey, O'Brien e Smith (2000) para a mineração de ativos para LPS e considera, adicionalmente, o uso de modelos de *features* para apoiar a sua realização, como proposto por Bosch e Ran (2000). Dessa forma, não se tem como requisito obrigatório a existência de artefatos provenientes de tarefas supostamente realizadas em desenvolvimentos anteriores, como modelos de análise, relatórios de testes etc. O modelo de processo adaptado é descrito a seguir e considera inicialmente a existência do código legado e da documentação do produto, *i.e.*, hardware, sistema operacional etc.

A reunião de programadores, usuários e outras pessoas ligadas direta ou indiretamente aos sistemas legados inicia o processo e tem como objetivo elicitar as deficiências atuais e as necessidades imediatas e futuras desses sistemas, passo (1). Com essas informações e com a ajuda do cliente é possível efetuar análises preliminares de viabilidade técnica e econômica do projeto de revitalização dos sistemas, com base nas necessidades obtidas anteriormente. Quando aprovada a continuidade da revitalização essas necessidades devem guiar a modelagem de uma estratégia apropriada para a realização do projeto, assim como estabelecer seu escopo, seu objetivo e suas prioridades, passo (2). O passo (3) inicia com a formação de um grupo técnico de pessoas, que contém um ou mais especialistas do domínio, para auxiliar o entendimento da documentação e do código legado existentes, além de esclarecer, da melhor maneira possível, conceitos específicos do domínio e suas particularidades. As prioridades do projeto ajudam a definir o conjunto de conhecimentos a ser adquirido inicialmente, que pode abranger todo um sistema ou apenas uma de suas partes, caso a mineração deva ser realizada de forma gradativa. Técnicas de LPS para modelagem de domínios são utilizadas para documentar os conhecimentos obtidos nessa etapa e apoiar o próximo passo da mineração. De posse das informações coletadas até esse ponto, são iniciadas as atividades de mineração, passo (4), que dependem do objetivo do projeto e da estratégia estabelecida previamente.

Um modelo de *features* é utilizado para a modelagem do domínio e o apoio ao desenvolvimento futuro de componentes genéricos de software. A criação desse modelo deve ser feita de modo que cada *feature* tenha um único componente associado. Para isso, podem ser necessários refinamentos sucessivos do modelo para a remoção de divisões e junções de *features*, Figura 2, descritas por Sochos, Riebisch e Philippow (2006) no método FArM (***F**eature-**Ar**chitecture **M**apping*).

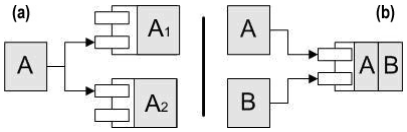

Figura 2. Divisão da *feature* A em dois componentes A1 e A2 (a); Junção das *features* A e B em um único componente AB (b) {Adaptado de [Sochos, Riebisch e Philippow 2006]}

Em uma mineração gradativa, o modelo de *features* do domínio pode ser inicialmente simples, detalhando apenas as *features* que representam as necessidades imediatas e deficiências atuais dos sistemas legados. À medida que os componentes são desenvolvidos, podem ser feitos refinamentos ou adições de novas *features* ao modelo existente. Esse processo é realizado iterativamente.

O código legado das aplicações é utilizado para guiar o projeto das interfaces dos componentes para preservar a funcionalidade existente. Técnicas como herança e parametrização podem ser usadas para estender essa funcionalidade, permitindo que variabilidades do domínio sejam inseridas nesses sistemas. Para cada *feature*, deve-se efetuar uma busca no código legado procurando identificar todas as funções diretamente relacionadas a ela. A partir dessas funções, um Mapa de Conexão (MCo) é construído e usado como base para o projeto das interfaces do componente associado à *feature*. Quando houver um conjunto de sistemas similares, o MCo pode retratar de forma unificada a conexão de seus códigos com uma determinada *feature* comum. Isso aumenta a generalidade da interface do componente em desenvolvimento, ampliando as possibilidades de seu reuso no projeto de novos produtos de uma mesma família. A Figura 3 mostra de forma simplificada as atividades do modelo de processo apresentado. A seção seguinte apresenta a proposta de revitalização de códigos legados a partir desse processo.

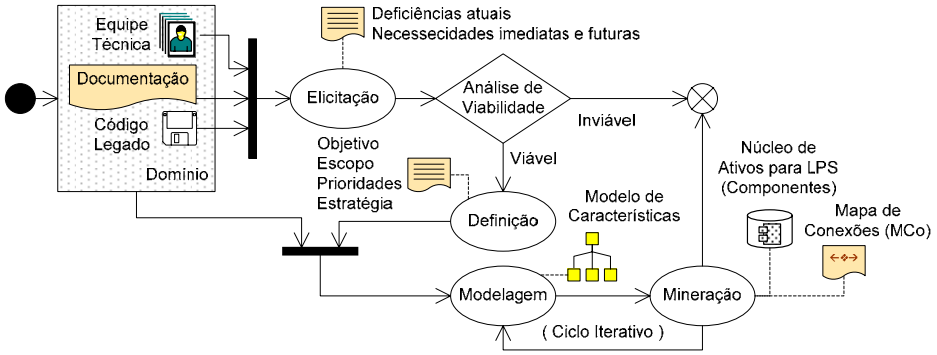

Figura 3. Modelo de processo para a mineração de ativos para LPS

5. Revitalização de Códigos Legados a partir da Mineração de Ativos

A realização do processo para a mineração de ativos para LPS descrito na Seção 4 resulta em: a) um modelo de *features*, que retrata as propriedades de um domínio, b) em um MCo, que documenta a conexão dessas *features* com o código dos sistemas legados pertencentes a esse domínio e c) em um núcleo de componentes de software, cujas interfaces reproduzem as funções atuais presentes nesse código. Embora possam ser utilizados de forma independente para a criação de novos produtos, esses componentes podem também, de forma apropriada, ser reconectados ao código legado para melhorar a sua estruturação e agregar novas funções à aplicação, proporcionando uma sobrevida ao produto original. Essa forma de revitalização considera a construção de *gateways*, como proposto por O'Brien (2005), permitindo o uso de diferentes paradigmas para o projeto dos componentes. O MCo e a documentação das interfaces dos componentes produzidos durante a mineração de ativos fornecem as informações necessárias para a construção dos *gateways*, que devem atuar como adaptadores de interface. Para a reconstrução dos sistemas originais usando os artefatos recém-criados, deve-se remover do código legado as implementações das funções contidas nos componentes mantendo, porém, suas chamadas, que serão redirecionadas para os componentes pelos *gateways*. Dessa forma, a revitalização dos sistemas legados é realizada sem nenhuma alteração na estrutura original de seus códigos, tornado o processo transparente para os mantenedores desses sistemas.

A revitalização, quando realizada de forma gradativa, deve facilitar a realização de testes para a validação dos sistemas reconstruídos, uma vez que as alterações são focadas em *features* isoladas e inteiramente implementadas nos componentes.

A seguir, um estudo de caso é apresentado para exemplificar o processo de revitalização aqui descrito.

6. Estudo de Caso

Para a realização deste estudo de caso foram tomadas três aplicações legadas similares para Transferências Eletrônicas de Fundos (TEF), desenvolvidas para terminais POS (*Point of Sale*), pertencentes à empresa VeriFone do Brasil, responsável por gerir o sigilo das informações contidas nos artefatos disponibilizados. Esses terminais são equipamentos pequenos, leves, portáteis e versáteis, com um conjunto reduzido de componentes periféricos padronizados, controlado por um sistema embutido microprocessado, que atendem às necessidades de um amplo segmento de mercado. No entanto, o custo, a capacidade de processamento e armazenamento de informações, a interface simplificada com o usuário e os dispositivos de comunicação, segurança e leitura de cartões com chip e trilha magnética direcionam seu uso para a realização de TEF com cartões de pagamento. A Figura 4 mostra um terminal POS e seus principais componentes periféricos.

Figura 4. Terminal POS e seus principais componentes periféricos[3]

Os documentos existentes das três aplicações foram entregues juntamente com os códigos legados. Essas aplicações são distintas, sem documentação de projeto, possuem estrutura modular e código escrito em linguagem C, com níveis insatisfatórios de reuso. As variabilidades existentes são decorrentes das freqüentes evoluções tecnológicas do hardware e foram implementadas na forma de blocos de código condicionais, denominados Módulos Selecionáveis[4]. O objetivo da empresa é evoluir rapidamente cada uma de suas aplicações legadas para diferentes plataformas de hardware, porém similares, preservando as regras de negócio nelas embutidas, as quais já passaram por um processo rigoroso de certificação e não devem ser modificadas. Programadores experientes da empresa atuaram como Especialistas do Domínio durante o processo. O processo para a mineração de ativos para LPS e a revitalização dos códigos legados a partir dos componentes produzidos foi realizado como segue.

[3] Terminal POS Vx-570 da empresa VeriFone Inc. (http://www.verifone.com)
[4] Denominação interna, criada pelos programadores da empresa.

Em uma reunião inicial envolvendo áreas estratégicas da empresa foi feito o levantamento das deficiências e das necessidades imediatas e futuras das três aplicações, na visão de cada participante. De posse dessas informações e dos artefatos fornecidos foi possível efetuar uma análise preliminar de viabilidade técnica do projeto de revitalização das aplicações. Diante do interesse em preservar as regras de negócio e permitir variações do hardware decidiu-se pela revitalização gradativa e concomitante das aplicações fornecidas, com enfoque nas *features* do hardware, iniciando com variabilidades de baixo impacto operacional, para facilitar a validação preliminar do processo a partir de seus primeiros resultados. Definida a estratégia, iniciou-se o estudo do domínio com a participação dos especialistas designados pela empresa. Após a leitura da documentação os códigos legados foram analisados e as dúvidas esclarecidas para entendimento dos elementos do hardware e seu uso pelas aplicações. O conhecimento adquirido foi documentado na forma de um modelo de *features*, mostrado na Figura 5. *Features* físicas e virtuais do hardware foram associadas nesse modelo para retratar não apenas a constituição física do equipamento, mas também a sua estruturação lógica do ponto de vista do sistema operacional. Como resultado pode-se observar, por exemplo, o console (*feature* lógica) constituído de tela e teclado (*features* físicas). Essa associação facilitou o mapeamento do modelo de *features* para um modelo de classes e posteriormente de componentes.

Os especialistas do domínio verificaram que informações importantes associadas às *features* do hardware não puderam ser claramente representadas no modelo, resultando na omissão de requisitos importantes. Para suprir essa deficiência foi criada uma extensão para o modelo de *features* denominada Nota, descrita a seguir.

Notas são representadas como na UML, podendo ser conectadas a uma ou mais *features* por meio de âncoras. Por questão de legibilidade, possuem internamente apenas números seqüenciais, que fazem referência a textos explicativos externos ao modelo. De um modo geral, as Notas agregam ao modelo de *features* detalhes do domínio que não podem ser representados graficamente e que desempenham um papel importante na tomada de decisões de projeto. Podem, ainda, conter informações temporárias em um determinado nível de abstração, que servirão como guias para refinamentos do modelo em etapas posteriores do projeto. Quando o texto de uma nota descreve uma informação relevante para a implementação da *feature*, uma letra 'i' é acrescentada à frente do número seqüencial.

Na seqüência foram iniciadas as atividades de mineração dos ativos para LPS descritas na Seção 4, com base na estratégia inicialmente definida. Como as aplicações legadas não implementam apoio à conexão de teclado externo e sendo essa uma variabilidade de baixo impacto operacional e importante para a estratégia de negócios da empresa, optou-se por validar o processo a partir de sua implementação. Foram realizadas buscas nos códigos legados das três aplicações para identificar as funções diretamente associadas à *feature* escolhida. Para apoiar o desenvolvimento do componente correspondente, foram adicionadas informações sobre propriedades desejáveis da *feature* e importantes limitações dos teclados interno e externo, que foram anotadas como extensão do MCo, ilustrado na Tabela 1.

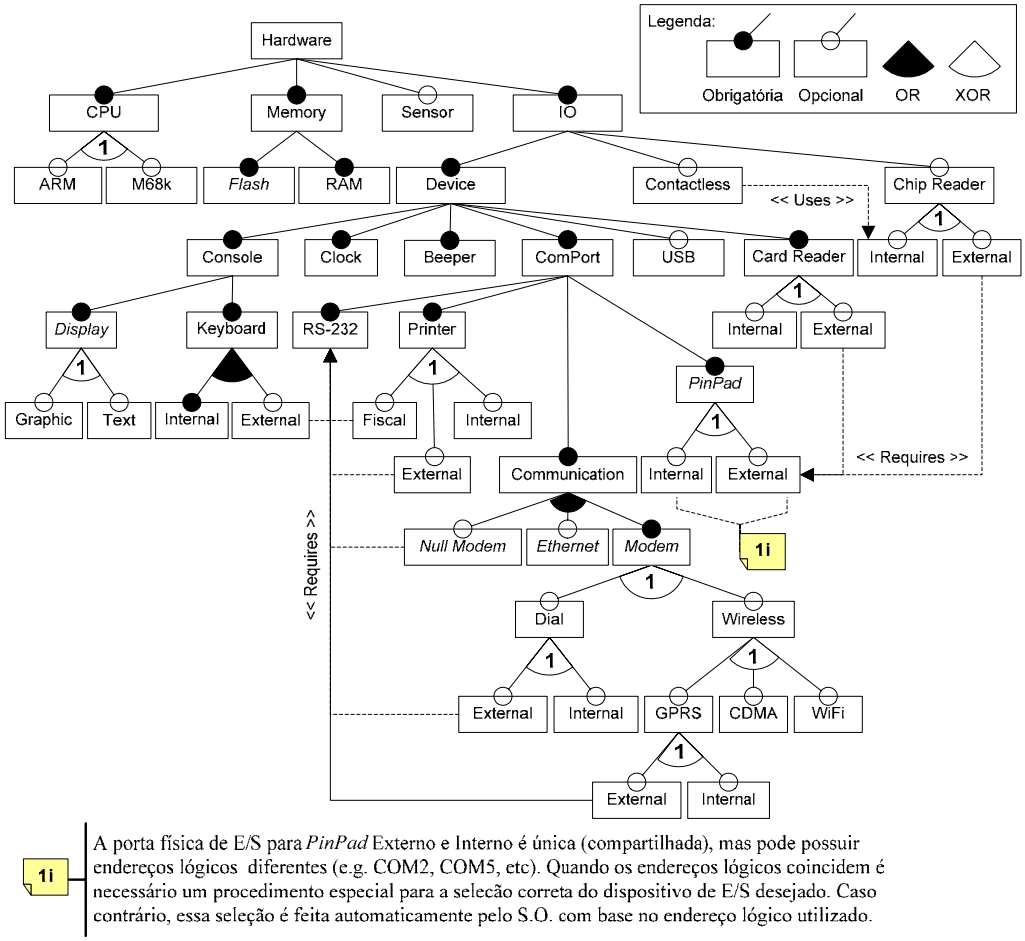

Figura 5. Modelo de *features* do hardware de terminais POS[5]

Tabela 1. Mapa de Conexões (MCo) da *feature Keyboard*

***Feature*:**		**Hardware -> IO -> Device -> Console -> Keyboard**	
Função	**Parâmetros**	**Retorno**	**Comentários**
`KBHIT`	Nenhum	Status: bool	Retorna TRUE se houver tecla pressionada.
`get_char`	Nenhum	Tecla : int	Aguarda o pressionamento de uma tecla e retorna seu valor.
Limitações: ▪ O sistema operacional não permite escritas no *buffer* do teclado interno. ▪ O teclado externo não possui um buffer de teclas provido pelo sistema operacional. **Novos Requisitos:** ▪ Os teclados, interno e externo, devem compartilhar um *buffer* comum de teclas, provido pelo componente, o qual deve permitir operações de escrita e leitura. ▪ A fim de flexibilizar o seu uso, a conexão do teclado externo pode ser feita nas portas RS-232 ou PinPad do terminal POS, com parâmetros de comunicação configuráveis.			

O desenvolvimento do componente foi iniciado a partir do MCo e do modelo de *features* do hardware, a partir do qual foi instanciado um sub-modelo para retratar apenas as *features* a serem tratadas, como mostra a Figura 6. Nesse sub-modelo foram

[5] Os nomes das *features* estão em inglês por exigência da empresa.

realizados e documentados refinamentos em relação ao modelo inicial, para apoiar os novos requisitos contidos no MCo. Uma dependência do teclado externo em relação à *feature PinPad* e uma nova *feature* de software, inexistente no código legado, os arquivos tipo INI[6], para permitir o armazenamento dos parâmetros de comunicação, foram adicionados.

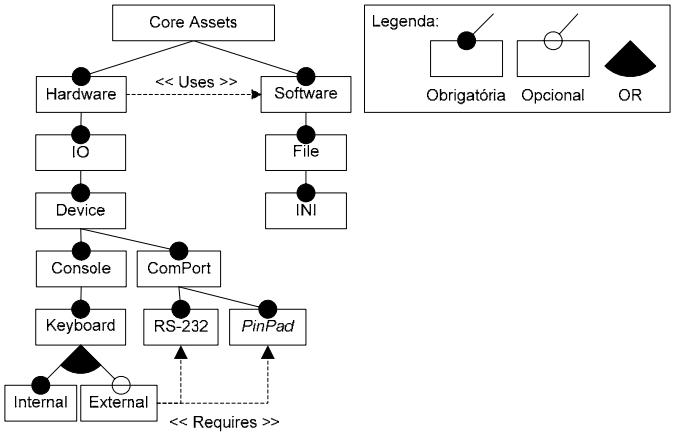

Figura 6. Sub-Modelo de *Features* para o componente *Keyboard*

Nesse modelo as *features Internal Pinpad* e *External Pinpad* não foram retratadas, pois apenas a capacidade de comunicação serial da *feature PinPad*, provida pela *feature ComPort*, deve ser implementada. O mapeamento do modelo de *features* da Figura 6 para um modelo de classes em UML foi feito a partir das informações contidas no MCo e na documentação do sistema operacional, que contêm os serviços disponibilizados para cada dispositivo de hardware. O resultado é o modelo de classes mostrado na Figura 7, no qual os teclados interno e externo foram unificados pela classe *Keyboard*, à qual foi adicionada a capacidade de escrita, atendendo aos requisitos do projeto. À classe *ComPort* foi adicionado um método para a configuração dos parâmetros de comunicação, para atender a esse mesmo conjunto requisitos.

A classe para tratamento de arquivos do tipo INI, não incluída neste artigo, implementa a leitura de sessões do arquivo, delimitadas por colchetes, armazena suas chaves de configuração na memória e permite a consulta de seus valores. A partir desse modelo foi desenvolvido um modelo de componentes para documentar os serviços oferecidos pelos novos ativos reutilizáveis criados, como mostra, de forma simplificada, a Figura 8. A Figura 9 mostra o arquivo de configuração HARDWARE.INI criado, com as duas possibilidades de conexão para o teclado externo, EXT-RS232 e EXT-PINPAD, em conformidade com os requisitos dessa *feature*.

Uma vez criados os componentes foi iniciada a implementação dos *gateways*, guiado pelo conteúdo do MCo. Como as funções listadas representam chamadas diretas à API (***A**pplication **P**rogramming **I**nterface*) do sistema operacional, não houve a necessidade de remover suas respectivas implementações do código legado. Porém, devido ao conflito de nomes, essas foram renomeadas e receberam um prefixo _G_ para identificar que estão implementadas no *gateway*. Nenhuma outra alteração foi feita no código legado das aplicações.

[6] Arquivo padrão do sistema operacional Windows para o armazenamento de configurações do ambiente.

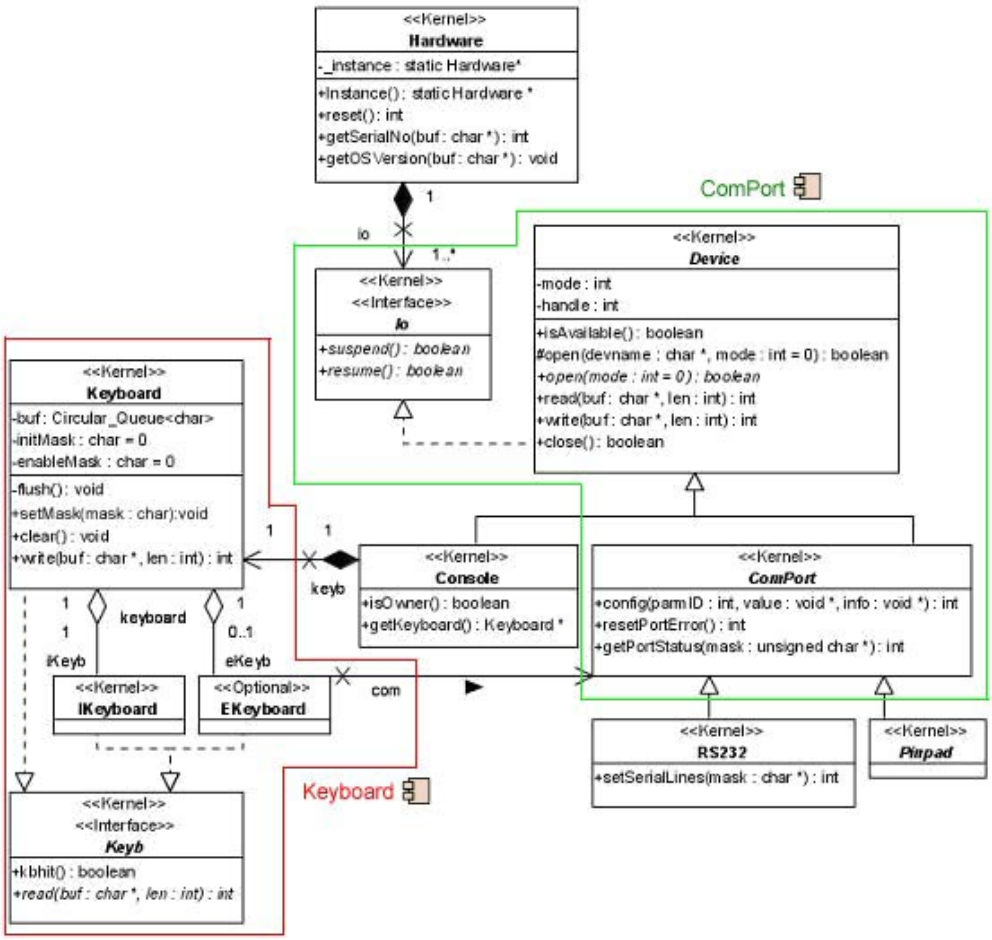

Figura 7. Modelo de Classes da *Feature Keyboard*

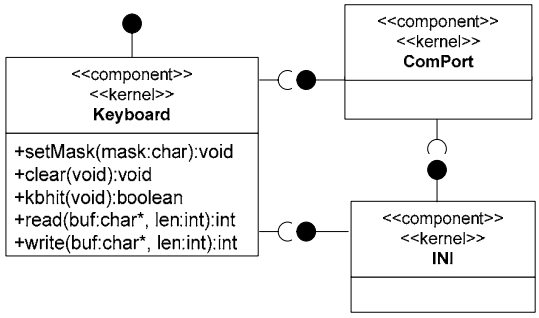

Figura 8. Modelo Simplificado de Componentes da *Feature Keyboard*

```
[CONSOLE]
KEYBOARD=EXT-PINPAD
[EXT-RS232]
# <1 = COM1, 2 = COM2>
COM=1
# MODE: <protocol>, <baudrate>, <data length>, <parity>, <stop bits>, <stx_char>, <etx_char>
MODE=CHAR,115200,8,N,1
[EXT-PINPAD]
COM=2
MODE=CHAR,19200,7,E,1
```

Figura 9. Arquivo HARDWARE.INI com as duas possibilidades de conexão para o teclado externo, EXT-RS232 e EXT-PINPAD

O *gateway* desenvolvido para a *feature Keyboard* é mostrado na Figura 10. Maiores detalhes dessa implementação foram omitidos devido à restrição de espaço.

```
using namespace std;

#include "Console.h"
#include "Keyboard.h"
#include "Sdk.h"
#include "Gateway.h"
namespace Hw
{    class Console   ;
     class Keyboard  ;
}
Hw::Console*    _c  = NULL  ;
Hw::Keyboard*   _k  = NULL  ;

void initConsole(void)
{    /* Defines static variability control through hardware.ini configuration file */
          _c = new Hw::Console( "HARDWARE.INI", "CONSOLE" );
     /* Address the Keyboard component .. */
     _k = _c->getKeyboard(); /* .. configured by the key "Keyboard =" in the [CONSOLE] section */
}
int _G_kbhit (void)  /* Keyboard MCo's KBHIT() function gateway */
{        if  (   !_k )initConsole( ) ; /* Console owns the Keyboard and must be initialized firstly */
     return  (int)_k->kbhit( )       ; /* Redirect the call to the Keyboard component */
}
int _G_getchar(void) /* Keyboard MCo's get_char() function gateway */
{
     char    ch                      ;
     while(!_G_kbhit())              ; /* Once a key is detected ..  */
               _k->read(&ch, 1)      ; /* .. reads it                */
     return  (int)    ch             ;
}
```

Figura 10. *Gateway* para a *feature Keyboard*

Após a sua construção, o *gateway* foi compilado com os códigos legados das aplicações juntamente com a implementação dos componentes, que agora passam a compor o núcleo de ativos reutilizáveis da empresa e podem apoiar o rápido desenvolvimento de produtos similares. Como resultado ressalta-se que foi possível operar as aplicações originais com ambos os teclados, de maneira configurável, preservando seus respectivos conteúdos, conforme solicitação inicial do cliente. Os resultados foram apresentados ao cliente, que aprovou a continuidade da revitalização, para a criação de componentes para as demais *features* do hardware.

7. Considerações Finais e Trabalhos Futuros

Para o domínio de sistemas embutidos, o apoio às variabilidades do hardware permite a oferta simultânea de diferentes produtos similares ao mercado, constituindo uma importante estratégia de negócios para as empresas. No domínio de terminais POS, o rigoroso processo de certificação das aplicações para TEF requer que as regras de negócio, embutidas nos códigos legados, permaneçam inalteradas e isoladas de erros durante a criação de novos produtos. Especialistas do domínio são comuns nesses ambientes, facilitando a realização de muitas das atividades propostas neste trabalho.

O estudo de caso apresentado, mostrou que é possível gerenciar as variabilidades do hardware por meio de componentes de software parametrizáveis, isolados do código legado e conectados a ele por meio de *gateways*. Exemplificou o uso de mineração de componentes para promover a revitalização de múltiplas aplicações similares, concomitantemente. Ao seu final, foram obtidos resultados satisfatórios de reuso, mesmo com a realização de uma revitalização incompleta, porém planejada.

O apoio unificado aos teclados interno e externo pelo componente *Keyboard* revelou uma situação não prevista inicialmente, na qual a junção de características, Figura 2(b), foi útil para atender aos requisitos de uma *feature.*

A estrutura Unix-like do sistema operacional dos terminais POS utilizados não permite às aplicações o acesso direto ao hardware, cujos recursos são disponibilizados por meio de uma API de comandos. Essa particularidade facilitou a construção do MCo, diminuindo a necessidade de interpretações do código legado.

Apesar de aplicada ao domínio de softwares convencionais, a técnica proposta por Mehta e Heineman (2002) pôde ser adaptada para o domínio de terminais POS, que possui *features* particulares, descritas anteriormente. A ela foram feitas as seguintes melhorias: 1) Além de modernizar o código legado, por meio da extração de *features* e inserção de componentes equivalentes, foi possível revitalizá-lo, estendendo sua funcionalidade para atender a novos requisitos; 2) O conjunto de componentes criados permitiu não só o reuso de *features* do software legado em outras aplicações, mas também a implementação de variabilidades do domínio em seu próprio código, permitindo a rápida criação de uma família de produtos; 3) A substituição dos testes de regressão por modelos de *features*, para apoiar a extração de *features* do código legado, permitiu a visualização das similaridades entre diferentes aplicações e das variabilidades do domínio, possibilitando a criação de interfaces de componentes mais flexíveis e a revitalização concomitante de produtos similares, além da recuperação progressiva da documentação, guiada pelas prioridades estabelecidas pelo cliente; 4) O uso de *gateways* para isolar o código dos componentes do código legado permitiu o uso de paradigmas mais modernos de desenvolvimento para a sua criação e também tornou o processo de revitalização transparente para os mantenedores do código legado, que puderam efetuar manutenções em pleno andamento da revitalização; 5) A revitalização gradativa dos códigos legados foram realizadas com baixo risco para o cliente, que pôde avaliar precocemente os custos e eficiência do processo a partir dos primeiros resultados parciais obtidos.

Durante o mapeamento do modelo de *features* em classes de objetos e componentes foi observada a ocorrência repetitiva de determinadas soluções, para as quais será realizado um estudo futuro para a identificação de possíveis padrões. Ferramentas generativas e de apoio à gerência de configuração devem ser construídas para facilitar a criação e a manutenção de famílias de produtos. Devem ser pesquisadas técnicas alternativas para a extração de características a partir de códigos legados.

8. Referências Bibliográficas

Ajila S.A.; Tierney P.J. (2002). The FOOM Method - Modeling Software Product Line in Industrial Settings. Proceedings of the International Conference on Software Engineering Research and Practice, Las Vegas, USA. <http://www.sce.carleton.ca/faculty/ajila/SERP02 - Camera Ready.pdf>. Acesso: 24.06.07

Atkinson, C.; Bayer, J.; Bunse, C.; Kamsties, E.; Laitenberger, O.; Laqua, R.; Muthig, D.; Paech, B.; Wust, J.; Zettel, J. (2001) Component-Based Product Line Engineering with UML. 1st Edition, Addison-Wesley Professional, USA.

Bergey J.; O'Brien L.; Smith D. (2000). Mining Existing Assets for Software Product Lines. Technical Note, CMU/SEI-2000-TN-008, SEI, USA.

Booch, G.; Rumbaugh, J.; Jacobson, I. (1999). The Unified Modeling Language Reference Manual, Addison-Wesley, USA.

Bosch, J.; Ran, A. (2000). Evolution of Software Product Families. Proceedings of the International Workshop on Software Architectures for Product Families. LNCS, v. 1951/2000, p. 168-183, Springer-Verlag, UK.

Clements, P.; Northrop, L. (2001). Software Product Lines: Practices and Patterns. 1st Edition, Addison-Wesley Professional, USA

Crnkovic I. (2005). Component-based software engineering for embedded systems. Proceedings of the 27th International Conference on Software engineering, p. 712-713. ACM Press, USA.

Czarnecki, K.; Eisenecker, U.W. (2000). Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional, USA.

Gomaa H. (2004). Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley Professional, USA.

Graaf, B.; Lormans, M.; Toetenel H. (2003). Embedded Software Engineering: The State of the Practice. IEEE Software, v. 20, p. 61-69.

Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson S. (1990). Feature-Oriented Domain Analysis (FODA): Feasibility Study. CMU/SEI-90-TR-21, SEI, USA.

Krueger C. W. (1992). Software reuse. ACM Computing Surveys, v.24(2), p. 131-183.

Mehta A., Heineman G.T. (2002). Evolving legacy system features into fine-grained components. Proceedings of the 24th International Conference on Software Engineering, Orlando, FL. p. 417-427. ACM Press, USA.

O'Brien, L.(2005). Reengineering. SEI, USA. <http://www.cs.cmu.edu/~aldrich/courses/654-sp05/handouts/MSE-Reeng-05.pdf>. Acesso: 17.06.07.

Riebisch, M.; Böllert, K.; Streitferdt, D.; Philippow, I. (2002) Extending Feature Diagrams with UML Multiplicities. 6th World Conference on Integrated Design & Process Technology, USA. <http://www.theoinf.tu-ilmenau.de/~riebisch/publ/IDPT2002-paper.pdf>. Acesso: 17.06.07.

Sochos, P.; Riebisch, M. e Philippow I. (2006). The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, p. 308-318.

Szyperski C. (2002) Component Software: Beyond Object-Oriented Programming. 2nd Edition, Addison-Wesley, USA.

Vahid, F.; Givargis, T. (2002), Embedded System Design: A Unified Hardware/Software Introduction. John Wiley & Sons, USA.

Ziadi, T.; Jézéquel, J-M.; Fondement, F. (2003). Product line derivation with UML. In Proceedings Software Variability Management Workshop, University of Groningen. <http://lglpc35.epfl.ch/lgl/members/fondement/docs/papers/Ziadi03b.pdf>. Acesso: 17.06.07.