Reprojeto de Sistemas Legados Baseado em Componentes de Software

Valdirene Fontanette, Vinícius C. Garcia, Adriano A. Bossonaro, Angela B. Perez, Antonio F. do Prado
Universidade Federal de São Carlos, Departamento de Computação
São Carlos, Brazil, CEP 13565-905
{valdirene, vinicius, bossonaro, angela, prado}@dc.ufscar.br

Abstract

This work presents a strategy for Component-Based ReDesigning of Legacy Systems, which integrates a software transformation system, a CASE tool and other technologies to reconstruct legacy systems using distributed components.

The strategy is accomplished in 4 steps: Organize Legacy Code, Recover Project, Reproject and Reimplement. In the **Organize Legacy Code** step, the legacy code is organized, using transformations, according to the the Object-Orientation principles, segmenting it in supposed classes, attributes and methods. The **Recover Design** step starts from the organized legacy code and also using software transformations, generates its design description in the MDL (Modeling Domain Language) language. The MDL descriptions are imported in a CASE tool, recovering the legacy code design, represented with UML techniques. In the **ReDesign** step, the Software Engineer redesigns the system with distributed software components. New functional and non-functional requirements can be added in this step. Finally, in the **Reimplement** step, the system reimplementation is done in an executable language target of the reimplementation. The system final code can be done through the CASE tool or using transformations.

Keywords: Software Engineering, Software Reengineering, Reverse Engineering, Object Orientation, Software Transformation, Distributed Components and Reuse.

Resumo

Este artigo apresenta uma estratégia para o Reprojeto de Sistemas Legados Baseado em Componentes de Software, que integra um sistema de transformação, uma ferramenta CASE e outras tecnologias para reconstruir sistemas legados usando componentes distribuídos.

A estratégia é realizada em 4 passos: Organizar Código Legado, Recuperar Projeto, Reprojetar e Reimplementar. No passo **Organizar Código Legado**, o código legado é organizado, usando transformações, de acordo com os proncípios da Orientação a Objetos, segmentando o código em supostas classes, atributos e métodos. O passo **Recuperar Projeto** parte do código legado organizado e também usando transformações de software, gera sua descrição na linguagem MDL (Modeling Domain Language). Estas descrições são importadas numa ferramenta CASE, recuperando o projeto do código legado, representado com notação UML. No passo **Reprojetar**, o Engenheiro de Software reprojeta o sistema usando componentes distribuídos. Novos requisitos funcionais e não-funcionais como plataforma, podem ser adicionados neste passo. Finalmente, no passo **Reimplementar**, faz-se a reimplementação do sistema numa linguagem alvo da reimplementação neste caso, Java. O código final do sistema pode ser gerado pela ferramenta CASE ou usando transformações.

Palavras-Chaves: Engenharia de Software, Reengenharia de Software, Engenharia Reversa, Orientação a Objetos, Transformação de Software, Componentes Distribuídos e Reuso.

1 Introdução

Sistema de software é um artefato evolutivo e requer constantes modificações, seja para corrigir erros, melhorar desempenho, adicionar novos requisitos ou mesmo para adaptá-lo para novas plataformas de hardware e software.

A dificuldade em atualizar estes softwares para a utilização de novas tecnologias tem motivado os pesquisadores a investigar soluções que diminuam os custos de desenvolvimento, garantam um tempo de vida maior para o sistema e facilitem a sua manutenção[1,2].

O conhecimento adquirido com estes sistemas antigos, denominados sistemas legados, é utilizado como base para a evolução contínua e estruturada do software. O código legado possui lógica de programação, decisões de projeto, requisitos de usuário e regras de negócio, que podem ser recuperados, facilitando sua reconstrução, sem perda da semântica.

A Reengenharia de Software é também uma forma de reutilização que permite obter o entendimento do domínio da aplicação, recuperando as informações das etapas de análise e projeto, organizando-as de forma coerente e reutilizável.

Explorando as diferentes tecnologias da Engenharia de Software, pesquisou-se uma Estratégia a para o reprojeto de Sistemas Legados baseado em componentes de software, cujo objetivo é recuperar o modelo de análise de sistemas legados, reconstruindo-os para serem executados em novas plataformas de hardware e software, e incorporar novas tecnologias que surgem. Dentre essas tecnologias está o uso de componentes de software, o que proporciona ao sistema reconstruído melhor qualidade e capacidade de reuso.
A estratégia suporta a manutenção do sistema, inclusive com alteração da sua estrutura original, garantindo sua evolução através do uso de componentes de software.

Segundo Sametinger[3], componentes reutilizáveis são artefatos autocontidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras em conformidade com um dado modelo de arquitetura de software, documentação apropriada e um grau de reutilização definido.

Na estratégia, os componentes podem ser utilizados no reprojeto e reimplementação do sistema segundo o método *Catalysis*[4].

Este artigo está organizado da seguinte maneira: a seção 2 apresenta os principais mecanismos utilizados na estratégia como o sistema de transformação Draco, a ferramenta MVCASE e o método Catalysis, a seção 3 apresenta a estratégia proposta descrevendo cada um de seus passos, a seção 4 apresenta um estudo de caso validando a estratégia, a seção 5 apresenta trabalhos relacionados . Finalmente a seção 6 apresenta as conclusões desta pesquisa.

2 Principais Mecanismos da Estratégia

Dentre os principais mecanismos utilizados para dar suporte à estratégia proposta estão: o sistema de transformação Draco, a ferramenta CASE MVCASE[5] e o método de desenvolvimento baseado em componentes Catalysis[3].

Diferentes sistemas de transformação têm sido usados em projetos de engenharia e reengenharia de software destacando-se o *IllumaSM*[6], *Popart*[7]. Outro importante sistema de transformação de software que vem sendo utilizado é o *Draco.* Um primeiro protótipo do sistema transformacional Draco foi construído por Neighbors[2]. Posteriormente, o sistema transformacional Draco foi refinado e reconstruído na PUC-RJ[8], usando novos domínios e funcionalidades.

O Sistema de Transformação (ST) Draco, foi construído com o objetivo de testar, desenvolver e colocar em prática o paradigma Draco, criado para implementar as idéias de transformação de software orientada a domínios. Com a estratégia proposta por Prado[8] é possível a reconstrução de software pela "transformação" direta do código fonte de uma linguagem para linguagens de outros domínios. Um domínio no Draco é definido através de uma *Linguagem;* esta por sua vez é formada por uma *Gramática*, um P*arser* e *Prettyprinter,* ou *unparser*, definidos a partir desta gramática, conforme mostra a Figura 1. Uma **Gramática** consiste em símbolos terminais, não-terminais, um símbolo de partida e produções. Um **Parser** é responsável por analisar os programas da linguagem e gerar a representação interna de uma DAST *(Draco Abstract Syntax Tree)* no Draco; e o **P*rettyprinter*** ou *unparser*, faz a formatação da DAST, tornando-a novamente textual na linguagem do domínio. Por último, os ***transformadores***, mapeiam estruturas sintáticas de uma linguagem para outras estruturas sintáticas, que podem estar na mesma linguagem de domínio, chamados transformadores *Intradomínio*, ou em outra linguagem de um domínio, chamados transformadores *Interdomínios*. São usados diferentes pontos de controle nas transformações. Normalmente tem-se o *LHS* (*Left Hand Side*) e o *RHS* (*Right Hand Side*), que definem os padrões de reconhecimento e substituição, respectivamente.

O ST Draco dispõe ainda de uma Base de Conhecimento (*Knowledge Base*), que permite armazenar fatos e regras com informações sobre o código legado, que são consultadas durante as transformações. Por exemplo, a Base de Conhecimento pode armazenar fatos relacionados com a declaração e utilização de variáveis em um programa. Outro recurso do Draco é o *Workspace*, usado para agrupar ou desagrupar comandos do código legado em blocos, durante a sua organização segundo os princípios da orientação a objetos. A Base de Conhecimento também facilita a execução de transformações, que necessitam de informações capturadas por outras transformações.

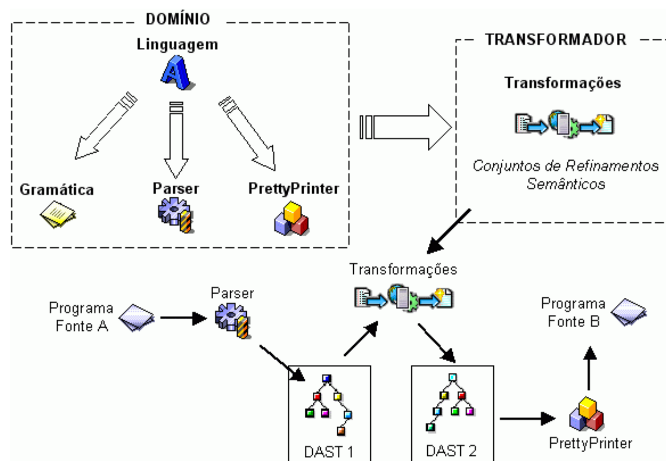

Figura 1- Partes de um Domínio no Draco

Para facilitar o desenvolvimento dos domínios no ST Draco foi construída a ferramenta denominada *Draco Domain Editor* (DDE) que suporta a edição textual e gráfica das gramáticas, que dão origem aos *parsers* e *unparsers* dos domínios, dos transformadores Inter e Intradomínios e de *scripts* para execução dos transformadores.

O DDE auxilia na utilização do ST Draco disponibilizando, em um mesmo ambiente, recursos para a edição dos domínios e para aplicação das transformações. Anteriormente, sem o DDE, o Engenheiro de Software tinha que editar o domínio em outro ambiente diferente do ST Draco.

Para que se tenha um melhor entendimento do DDE a Figura 2 mostra a tela para a edição da gramática de um domínio definido no ST Draco. À esquerda (1) tem-se uma estrutura de diretórios onde estão localizados os domínios do ST Draco. Selecionando-se, por exemplo, o domínio Clipper pode-se visualizar sua gramática na forma textual (2) ou gráfica (3). Ainda na ferramenta (2), o Engenheiro de Software pode editar a gramática, obtendo-se a representação gráfica da sua árvore gramatical (3).

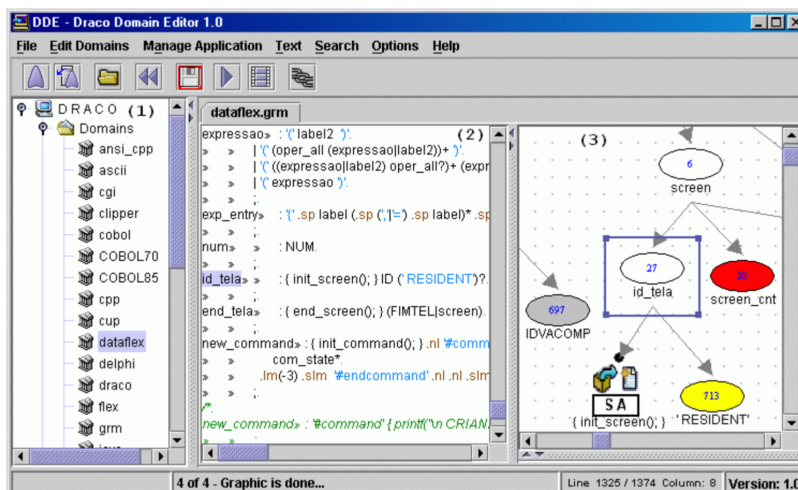

Figura 2 – Draco Domain Editor (DDE)

A MVCASE[5] é uma ferramenta CASE orientada a objetos que suporta a especificação de requisitos do sistema usando a notação UML[9]. Permite a construção de modelos gráficos e textuais que representam o sistema em diferentes níveis de abstração. Os modelos gráficos de uma especificação na MVCASE facilitam a comunicação entre os desenvolvedores do sistema e seus usuários.

A MVCASE suporta também o Desenvolvimento Baseado em Componentes (DBC), utilizando a tecnologia *Enterprise Java Beans* (EJB)[10] para implementação. O Engenheiro de Software pode construir o Modelo de Componentes Distribuídos de um sistema, com base no seu Modelo de Objetos. A MVCASE pode também gerar o

código Java/EJB dos componentes. Além do código Java/EJB, a MVCASE gera as descrições em XML, que disponibilizam os componentes, em um servidor EJB, para serem reutilizados pelas diferentes aplicações.

A ferramenta MVCASE foi desenvolvida visando suportar a estratégia, através da construção de um ambiente integrado à aplicação da estratégia. A escolha da MVCase se deve também pelo fato de ela ser uma ferramenta desenvolvida em java e portanto multi-plataforma, assim como o ST Draco que foi desenvolvido em C e é de livre distribuição.

Catalysis[3] é um método para DBC que utiliza a notação UML para modelar Sistemas com Componentes Distribuídos. Tem como base os princípios: Abstração, Precisão e Componentes Plug-In. O princípio *Abstração* orienta o Engenheiro de Software na busca dos aspectos essenciais do sistema, dispensando detalhes que não são relevantes para o contexto do sistema. O princípio *Precisão* objetiva descobrir erros e inconsistências na modelagem, e o princípio *Componentes "Plug-In"* visa a reutilização de componentes para construir outros componentes.

O método apresenta modelos precisos e um processo de desenvolvimento completo e sistemático, que cobre as diferentes fases do ciclo de vida do componente, desde a identificação e análise dos seus requisitos até seu projeto e implementação. É dividido em 3 níveis: *Domínio do Problema*, que dá ênfase na identificação dos requisitos do sistema, especificando "o que" o sistema deve fazer para solucionar o problema, *Especificação dos Componentes*, onde se define comportamento e responsabilidades dos componentes e *Projeto Interno dos Componentes*, que dá ênfase no projeto físico dos componentes, preocupando-se com seus requisitos não-funcionais, e com suas distribuições físicas.

Foram estudados vários métodos[11,12,13] para Desenvolvimento Baseado em Componentes e optamos por Catalysis em razão da clara divisão entre as fases para o desenvolvimento dos componentes e dos princípios que ele se baseia.

3 ReProjeto de Sistemas Legados Baseado em Componentes de Software

Combinando as tecnologias do sistema de transformação Draco[8], da ferramenta MVCASE[5], as principais técnicas de Engenharia Reversa[14] e do método DBC *Catalysis*[3], definiu-se uma estratégia para o Reprojeto de Sistemas Legados Baseado em Componentes de Software, realizada em 4 passos: Organizar Código Legado, Recuperar Projeto, Reprojetar e Reimplementar, conforme mostra a Figura 3.

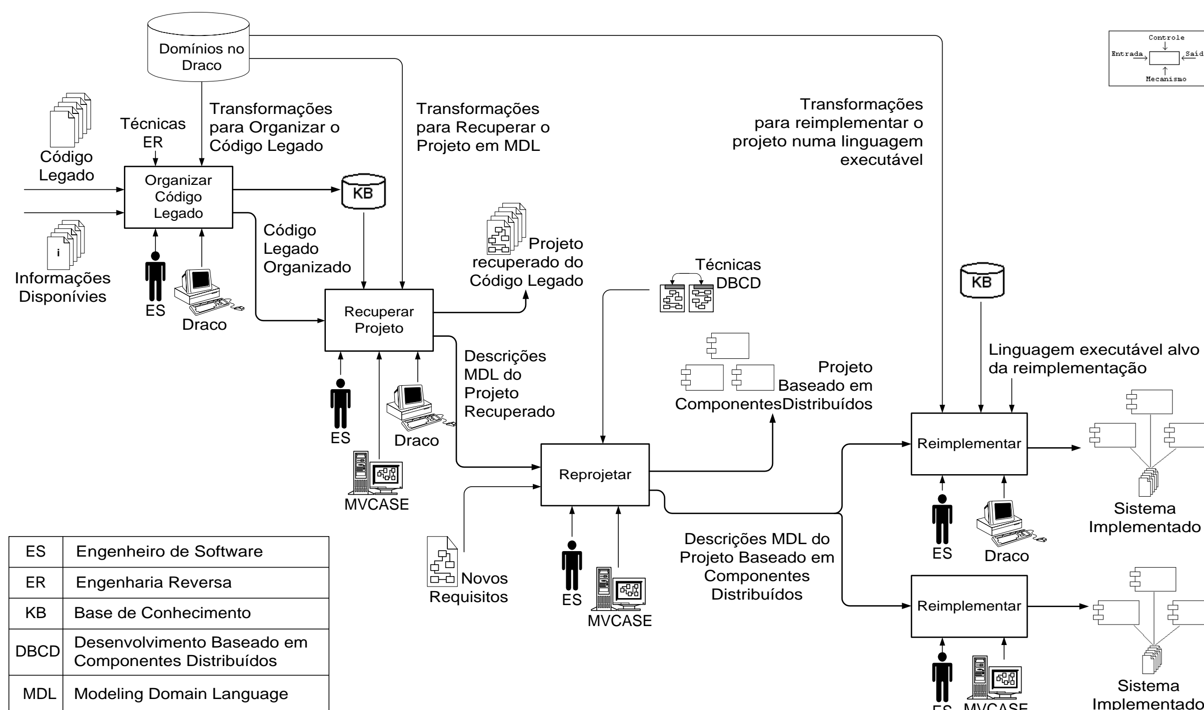

Figura 3 – Estratégia de Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações

3.1 Organizar Código Legado

Este passo, **Organizar Código Legado**, é um passo preparatório para facilitar a transformação de um código procedural para Orientado a Objetos.

O Engenheiro de Software com o auxílio do ST Draco e de técnicas de Engenharia Reversa, organiza o código legado, em classes com seus atributos e métodos. O código legado, escrito de forma procedural, possui comandos e declarações que podem ser organizados, sem prejuízo da sua lógica e semântica, de forma a facilitar sua transformação para o paradigma orientado a objetos, que tem a classe como a unidade básica.

Primeiramente reúne-se toda a documentação disponível sobre o sistema legado, principalmente os seus programas fontes. Elabora a relação CHAMA/CHAMADO[14], para determinar a seqüência de programas a serem submetidos às transformações de organização do código legado. Esta atividade é realizada manualmente pelo Engenheiro de Software. Em seguida, o código legado é submetido a um transformador, que contém transformações desprovidas de padrão de substituição (RHS), com objetivo de armazenar na Base de Conhecimento, fatos e regras, com informações sobre a organização do código legado. Dentre estas transformações, destacam-se as utilizadas para:

a) Identificar supostas classes através do reconhecimento de comandos de abertura e acesso aos arquivos de dados, estruturas de dados;

b) Identificar supostos atributos, reconhecidos pelos campos dos arquivos de dados, comandos de acesso ao banco de dados, campos de estruturas de dados;

c) Identificar supostas classes de interface. Em sistemas orientados a menus, cada tela de interação dá origem a uma suposta classe de interface e os objetos de interface dão origem a um suposto atributo da classe de interface; Em sistemas não orientados a menus, cada bloco de comandos do código legado, que não faz referência a arquivo de dados, é definido como um suposto método de uma suposta classe de interface. Esta suposta classe não possui objetos de interface e, portanto, não dá origem a nenhum suposto atributo;

d) Identificar supostos métodos, de uma suposta classe, a partir das unidades de programas que podem ser procedimentos, funções ou blocos de comandos executados pelo disparo de um evento de interface ou pela chamada de outra unidade de programa, respeitando o escopo, dependência de dados e visibilidade do código legado. Os supostos métodos são classificados em construtores **(c)**, quando alteram uma estrutura de dados, e observadores **(o)**, quando somente consultam a estrutura de dados. Um suposto método que faz referência à somente um arquivo de dados, é relacionado como suposto método da suposta classe identificada para este arquivo. Se o suposto método não consulta nem modifica nenhum arquivo de dados, é relacionado com uma suposta classe de interface. Quando um suposto método refere-se a mais de uma suposta classe ele deve ser alocado usando os seguintes critérios[14]:

   - Se um suposto método for construtor de uma suposta classe e observador de outra **(oc)**, será alocado na suposta classe que faz referência como construtor;
   - Se um suposto método for observador de uma suposta classe e construtor de várias **(oc+)**, será alocado na primeira suposta classe que faz referência como construtor**;**
   - Se um suposto método for observador de mais de uma suposta classe e construtor de apenas uma **(o+c)**, será alocado na primeira suposta classe que faz referência como observador;
   - Todas as unidades de programas candidatas a supostos métodos de uma mesma suposta classe são reunidas em um único arquivo;

e) Identificar relacionamentos das supostas classes, através de variáveis e comandos que manipulam dados de um arquivo de dados. É verificado se um campo de um arquivo, identificado como chave, tem seus valores consultados e atribuídos a uma variável, e posteriormente o valor dessa variável é armazenado em campos de outro arquivo ou é utilizado para fazer busca em registros de outros arquivos.

f) Identificar conexões de mensagens, a partir da interface do sistema, para definir os fluxos de execução de cada cenário de uso do sistema. Nos sistemas orientados a menus, parte-se de cada opção do menu para definir os diferentes cenários de uso do sistema.

A Figura 4 mostra as transformações *IdentifySelect* (à direita) e *IdentifySupposedClass* (à esquerda) usadas para identificar as supostas classes e seus supostos atributos. A primeira transformação *IdentifySelect* reconhece o comando SELECT do Clipper que define uma área de dados, usada por arquivos de extensão .dbf. No ponto de controle POST-MATCH cria-se um fato na Base de Conhecimento que relaciona esta área de dados com uma suposta classe. A transformação *IdentifySupposedClass* reconhece o comando USE do Clipper que dá origem a uma suposta classe. O ponto de controle LHS (1) especifica a regra de produção *statements* do parser Clipper que reconhece o comando USE, que identifica um arquivo de dados indexado. No ponto de controle POST-MATCH (2), inicialmente localiza o arquivo com extensão *.dbf*. Em (3) é criado o fato *SupposedClass* na Base de Conhecimento, associando a área do arquivo com a *CurrentClass*, que tem o mesmo nome do arquivo. Em (4) são identificados os atributos de cada campo do arquivo *.dbf*, como nome (*AttribName*), tipo (*AttribType*), tamanho (*AttribSize*) e parte decimal (*AttribDec*). E é criado o fato *SupposedAttribute* relacionando os com a suposta classe.

```
TRANSFORM IdentifySelect
LHS: {{dast clipper.statements
        SELECT [[NUM express1]]
}}
POST-MATCH: {{dast txt.decls
...
KBAssertIfNew("SupposedClass([[AreaNum]])");
KBWrite(kb);
...
```

```
TRANSFORM IdentifySupposedClass
LHS: {{dast clipper.statements    (1)
     USE [[iden FileName]] INDEX [[iden* Indexes]]
}}
POST-MATCH: {{dast txt.decls    (2)
...
sprintf(File, "%s", expand("[[CurrentClass]].dbf"));
Read_Class(File);
...
KBAssertIfNew("SupposedClass([[AreaNum]],[[CurrentClass]]")
KBWrite(kb);    (3)
KBAssertIfNew("SupoosedAttribute([[CurrentClass]],    (4)
    [[AttribName]], [[AttribType]],[[AttribSize]],
    [[AttribDec]])");
KBWrite(kb);
```

Figura 4 - Transform que identifica as supostas classes e seus supostos atributos

Outras transformações são usadas para identificar supostas classes de interface, supostos métodos, relacionamentos entre supostas classes e o fluxo de execução do código legado. Além dos fatos sobre supostas classes, supostos atributos e métodos outros fatos contendo informações sobre o código legado também são armazenados na Base de Conhecimento.

Numa segunda etapa, do passo Organizar Código Legado, com base nas informações coletadas, na Base de Conhecimento, pelo primeiro transformador Intradomínio, aplica-se um segundo transformador Intradomínio, responsável por segmentar o código legado em supostas classes com seus supostos atributos, métodos e relacionamentos.

Concluído este passo, tem-se o código legado organizado seguindo os princípios de orientação a objetos.

3.2 Recuperar Projeto

No segundo passo, **Recuperar Projeto**, obtém-se o projeto do código legado. Primeiramente, o Engenheiro de Software parte do código legado organizado e, novamente com o auxílio do ST Draco, obtém as especificações UML do projeto do código legado. As especificações UML, geradas pelo ST Draco, são armazenadas em descrições na linguagem de modelagem *MDL (Modeling Domain Language).*

Os comportamentos dos supostos métodos, separados em arquivos no passo Organizar Código Legado, e cujos protótipos já foram criados nas respectivas classes, são transformados diretamente para Java.

A Base de Conhecimento é consultada para a criação das classes, atributos, protótipos dos métodos e relacionamentos entre as classes.

A Figura 5 mostra uma transformação Interdomínios, que cria as classes e gera suas descrições na linguagem MDL.

```
TRANSFORM  Program
LHS:  {{dast clipper.program
   [[TAG  tag]]  [[ID ProgName]]
   [[prog_elements Elements]]
}}
POST-MATCH: {{dast txt.decls
…
 Cont = 1;
 sprintf (ClassNum, "%d", Cont);
 sprintf (Query, "SupposedClass(*x, *y, %s)", ClassNum)
 while (KBSolve (Query)) {
  sprintf (ClassName, "%s", KBRetrieve ("*y", 1));
  KBAssert ("Class([[ClassName]], [[ClassNum]])");
  KBWrite ("RBCD.kb");
…
  sprintf (Query,"SupposedAttribute(%s, *v, *w, *x, *y, *z,
  %d)",ClassName, Cont);
  while (KBSolve(Query)) {
…
  sprintf (query, "SupposedMethod (%s, *y, %d)"
  if (KBSolve (query))  {
   TEMPLATE ("TOperation")
```

```
TEMPLATE  Tclass
RHS: {{dast mdl.class_Object
  (object Class[[STRI NameWithAspas]]
    quid[[STRI NumWithAspas]]
    operations(list Operations
     [[operations* Coper]])
        class_attributes(list
        class_attribute_list
     [[ClassAttribute* Cattr]])
}}
```

```
TEMPLATE Tattributes
RHS:{{dast mdl.Attribute
 (object ClassAttribute[[STRI
              AttribName]]
    quid [[STRI AttribSize]]
    type [[STRI AttribType]])
}}
```

```
TEMPLATE Toperation
RHS: {{dast mdl.operations
   (object Operation [[STRI
 Nfunction]]
     quid  [[STRI Nid]])
}}
```

Figura 5 – Transform que gera a descrição MDL de uma classe e de seus atributos e métodos

Em seguida, o Engenheiro de Software, usando a ferramenta MVCase, importa as descrições MDL para obter o projeto recuperado do código legado. O projeto recuperado é representado pelos Modelos de classes, com seus atributos, métodos e relacionamentos, e pelos Modelos de Interações. Os Modelos de Interações são representados em Diagramas de Seqüência que definem os fluxos de execução do sistema, com suas conexões de mensagens.

3.3 Reprojetar

No terceiro passo, **Reprojetar**, o Engenheiro de Software com o auxílio da ferramenta MVCASE e de técnicas DBC

Distribuídos, faz o reprojeto orientado a componentes distribuídos do projeto recuperado do código legado, obtendo um novo projeto baseado em componentes distribuídos.

Neste passo, adaptamos os mesmos princípios da técnica de Framework de Modelos, do método Catalysis, que originalmente trabalha com tipos num nível maior de abstração, para trabalhar com classes, já que as mesmas foram recuperadas no passo anterior. Obtendo assim, um modelo de classes mais genérico, que pode ser reutilizado. Deste modo, tem-se o reuso não só do código mas também de artefatos de alto nível (modelos).

O Reprojeto é realizado em 3 passos. no primeiro passo, *Definir Domínio do Problema*, define-se o domínio do problema. No segundo passo, *Especificar Componentes*, faz-se a generalização do Modelo de Classes obtido no passo anterior, Recuperar, em um Framework de Modelos e em seguida o modelo é instanciado através da Aplicação do Framework e no terceiro passo, Projetar Componentes, faz-se o projeto interno dos componentes, ou seja, “como” os componentes devem fazer para solucionar o problema. São especificados os requisitos não-funcionais, como a plataforma de hardware e software, a linguagem de implementação e as restrições do projeto. As informações provenientes do passo, anterior, Recuperar, como por exemplo, se uma classe é persistente ou de Interface, são utilizadas neste passo.

Uma vez projetados os componentes, eles são reutilizados no desenvolvimento de aplicações. Algumas aplicações se originam diretamente das classes identificadas como sendo de interface. Algumas aplicações podem ser criadas, reutilizando os componentes projetados, para adicionar novas funcionalidades ao sistema.

3.4 Reimplementar

Finalmente, no quarto passo da estratégia, faz-se a reimplementação do sistema reprojetado. A reimplementação pode ser realizada através de transformações das descrições MDL do reprojeto, com o auxílio do sistema de transformação Draco ou através da ferramenta MVCASE que gera código automaticamente a partir dos modelos de objetos e componentes do reprojeto.

Segue-se a apresentação de um estudo de caso que mostra o uso da estratégia proposta.

4 Estudo de Caso

Trata-se de um sistema para Oficina Auto-Elétrica e Mecânica que controla os serviços executados nos veículos e o estoque das peças utilizadas. O cliente vai até a oficina para solicitar um serviço em seu veículo. Um cliente pode ter diversos veículos. O mesmo veículo pode voltar à oficina diversas vezes, sendo elaborada, em cada uma delas, uma Ordem de Serviço distinta. Essa Ordem de Serviço contém dados do cliente, do veículo e dos reparos a serem feitos.

Quando o veículo é consertado, a Ordem de Serviço é completada, introduzindo-se as peças utilizadas e a mão-de-obra executada. Muitas vezes, o reparo pode exigir peças não existentes no estoque, que são adquiridas fora da oficina mecânica e que também participam da Ordem de Serviço. O registro dessas peças é importante para o gerente da oficina, pois essas são candidatas a serem estocadas no futuro.

Esse sistema foi desenvolvido em 1990 e tem cerca de 20.000 linhas de código Clipper.

Segue-se uma apresentação de cada passo da estratégia para a reengenharia deste sistema.

4.1 Organizar Código Legado

Inicialmente reuniu-se as informações disponíveis sobre o código legado, no caso apenas o código legado, e elaborou-se a relação *CHAMA/CHAMADO*, para determinar a seqüência de programas a serem submetidos ao primeiro transformador, *ClipperToKB*, que organiza o código legado em classes com seus atributos e metódos.

A Figura 6 mostra à direita a Base de Conhecimento (2) após o reconhecimento da suposta classe *Customer* através dos comandos *SELECT* e *USE*, do código legado *Customer.prg*, e de seu suposto método *Method1* (1).

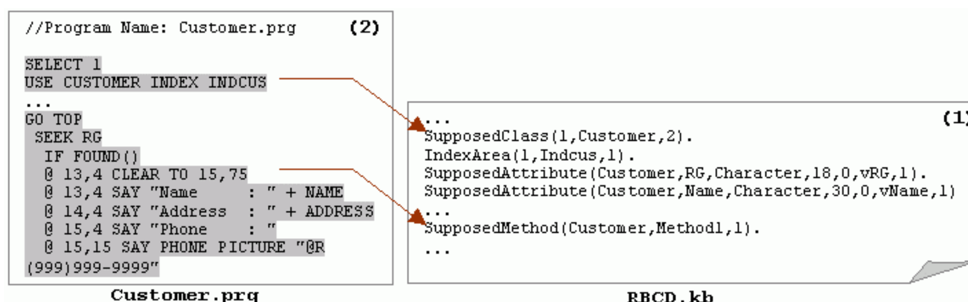

Figura 6 – Identificação de uma suposta classe e de seus atributos e métodos

Após identificar as supostas classes, seus supostos atributos e métodos, é aplicado um segundo transformador Intradomínio, *OrganizeClipper*, responsável por segmentar o código em supostas classes, atributos, métodos e relacionamentos.

A Figura 7 mostra à esquerda (1) o programa C*ustomer.prg* e à direita (2) seu correspondente código organizado. O código permanece na mesma linguagem, porém com classes, atributos e métodos . No caso, o código foi segmentado nas funções *CustomerOpen()*, *CustomerInit()* e *CustomerFind().*

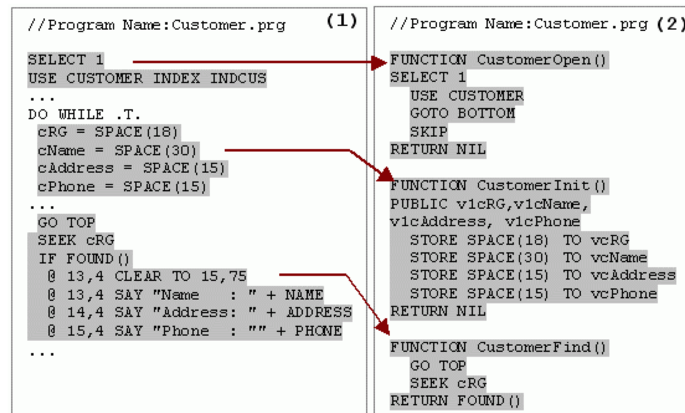

Figura 7 – Segmentação do código legado segundo os princípios da Orientação a Objetos

Da mesma forma procede-se em todo o código legado do sistema, obtendo-se no final deste passo o código segmentado e organizado em supostas classes, atributos, métodos e relacionamentos.

4.2 Recuperar Projeto

Neste passo aplica-se o terceiro transformador, *ClipperToMDL*, no código legado organizado para gerar as descrições MDL, que permitem recuperar o projeto do código legado.

A Figura 8 mostra à esquerda o código legado Clipper organizado e à direita a descrição MDL gerada, correspondente à especificação UML. Cada suposta classe, suposto método, e suposto atributo dão origem, respectivamente, à especificação de uma classe, operação, e atributo em UML, gerando a respectiva descrição em MDL. Por exemplo, a suposta classe identificada como *Customer* (1) dá origem à Class "*Customer*". O suposto método identificado como *PROCEDURE proc40* (2), dá origem à Operation "*proc40*", da classe "*Customer*". Os supostos atributos *vcRG* (3) e *vcName* (4) dão origem aos ClassAttributes "*vcRG*" e "*vcName*", respectivamente. No caso pode-se ver ainda que o comportamento de *proc40* do Clipper, foi transformado diretamente para Java, que é a linguagem alvo da re-implementação. Assim, o protótipo de um método em uma classe é descrito em MDL, e seu corpo, é descrito diretamente em Java. A geração de código para reimplementar o sistema em Java, fica mais fácil uma vez que já se tem pronto os códigos dos métodos.

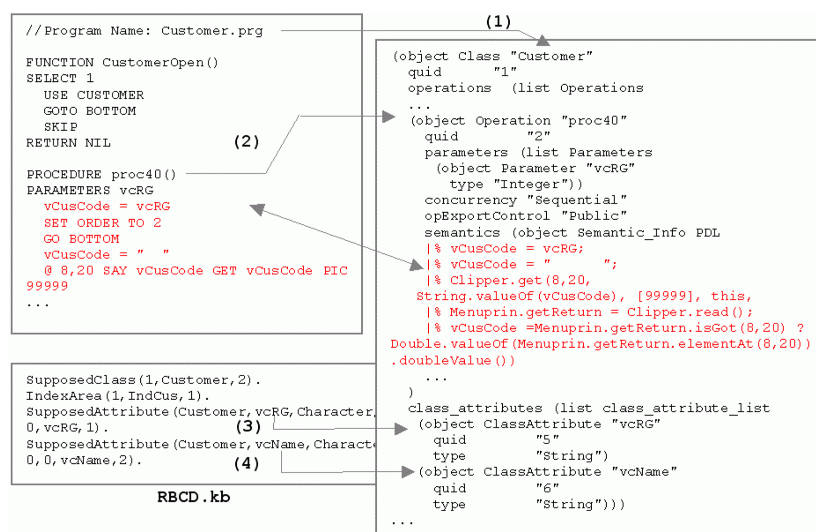

Figura 8 - Geração da descrição MDL de uma classe

Depois de concluídas as transformações de Clipper para MDL, o Engenheiro de Software pode importar as descrições MDL na MVCASE para obter o projeto recuperado do código legado, conforme mostra a Figura 9. No

caso, as classes Customer, Sale, ServiceOrder e Part foram identificadas como classes persistentes, por serem de banco e AddCustomerScreen como classe de interface.

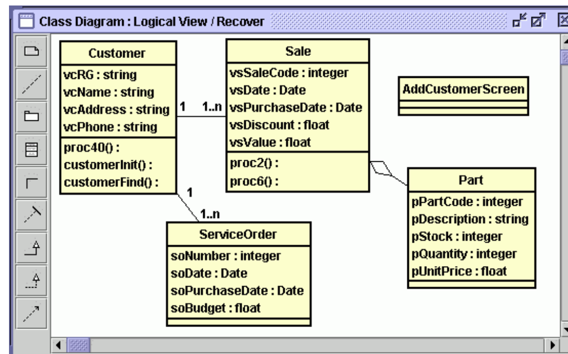

Figura 9 – Modelo de Classes recuperado na MVCASE

4.3 Reprojetar

Neste passo o Engenheiro de Software faz o reprojeto do sistema recuperado, usando como base os princípios do método Catalysis para construir o Modelo de Componentes. Assim, a partir do modelo de classes do passo anterior, realizam-se três atividades para a construção dos componentes. Estas atividades compreendem: Definir Domínio do Problema, Especificar Componentes e Projetar Componentes

4.3.1 Definir Domínio do Problema

Neste passo especifica-se os requisitos do sistema, ou seja, "o quê" os componentes devem fazer para solucionar o problema. Como o sistema é fruto do processo de Reengenharia o domínio do problema já foi definido pelo projeto recuperado no passo anterior, Recuperar.

4.3.2 Especificar Componentes

Para se obter o reuso não só do código, mas dos modelos em alto nível, faz-se a generalização do Modelo de Classes. No caso o Modelo de Classes da Figura 9 foi generalizado obtendo-se o Framework de Modelos da Figura 10. Os símbolos "<>" identificam quais são as classes genéricas e que devem ser associadas com as classes de uma aplicação.

A Figura 11 mostra esta associação entre classes do Framework de Modelos Vendas e Serviços e as classes, através da Aplicação do Framework.

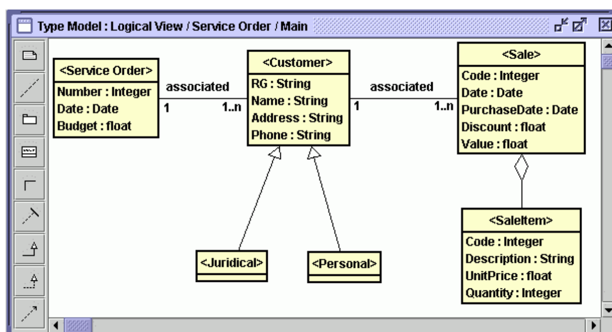

Figura 10 – Framework de Modelos

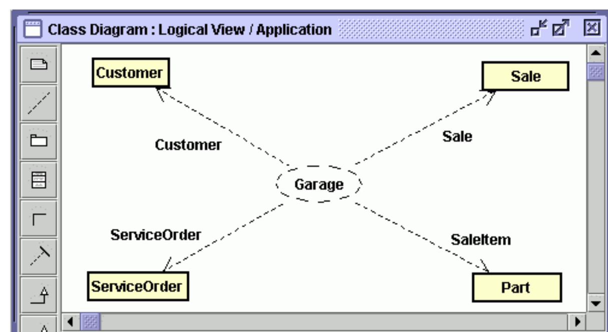

Figura 11 – Aplicação do Framework

Em seguida, são identificados os componentes de negócio do domínio. Como os atributos e operações encapsuladas em classes são recuperados no passo anterior, Recuperar, a tarefa se resume a identificar os componentes a partir das classes e especificar seus serviços através das Interfaces.

4.3.3 Projetar Componentes

Nesse passo é feito o projeto interno dos componentes, preocupando-se com outros requisitos não-funcionais, como por exemplo linguagem de programação e distribuição. A Figura 12 mostra o Projeto Interno do Componente Customer, originado da classe Customer, segundo a tecnologia EJB. Conforme identificado no passo anterior, Recuperar, a classe Customer é persistente e portanto originou um componente do tipo EntityBean[10]. De maneira similar, a classe AddCustomerServlet foi identificada como sendo de interface, e portanto originou o componente Web AddCustomerServlet, segundo a tecnologia Java/Servlet[15], como mostra a Figura 13.

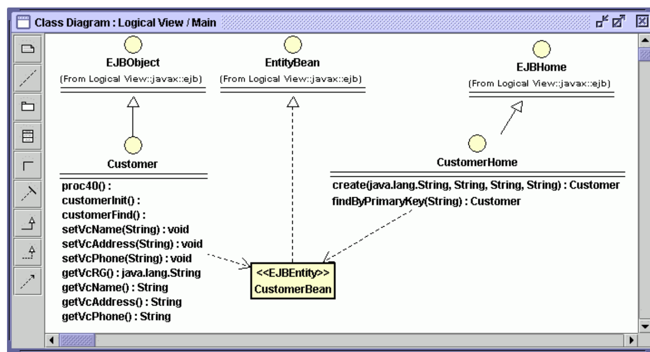

Figura 12 – Projeto Interno do Componente Customer

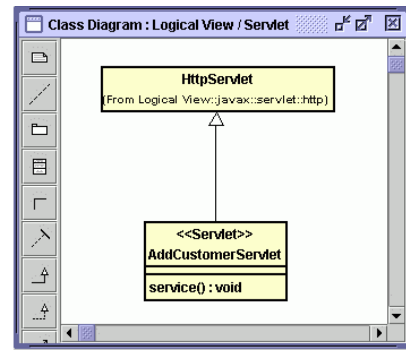

Figura 13 – Componente AddCustomerServlet

Uma vez projetados os componentes, pode-se reutilizá-los no desenvolvimento de aplicações. As classes identificadas como sendo de interface, no passo anterior, originaram algumas aplicações, como por exemplo a classe "AddCustomerScreen", que deu origem ao componente de aplicação "AddCustomerServlet". Novas aplicações podem ser construídas neste passo, adicionando novas funcionalidades ao sistema. A Figura 14 mostra três componentes de aplicação, "AddCustomerServlet", "RegisterSaleServlet" e "AddPartServlet", do tipo Servlet, que reutilizam os componentes "Customer", "Sale" e "Part", sendo que o componente de aplicação "AddCustomerServlet" foi originado de uma classe do sistema legado, e os componentes de aplicação "RegisterSaleServlet" e "AddPartServlet" foram construídos neste passo.

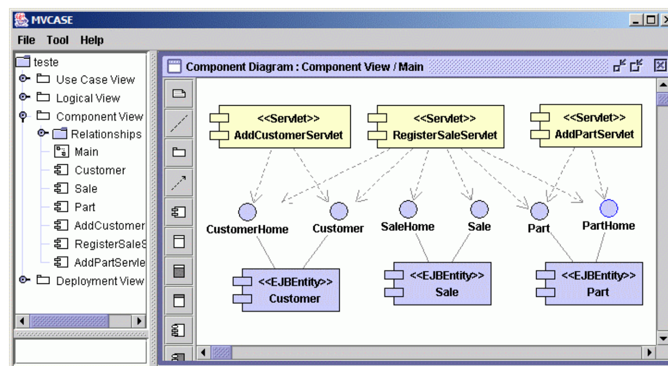

Figura 14 – Diagrama de Componentes

4.4 Reimplementar

A reimplementação pode ser realizada por transformação, usando o ST Draco ou diretamente pelo gerador de código Java da MVCASE.

Por exemplo, usando o ST Draco, aplica-se o quarto transformador, Interdomínio, *MDLToJava*, que transforma as descrições MDL em código Java. A Figura 15 mostra à esquerda a descrição MDL e à direita o correspondente código Java/EJB gerado pelas transformações. Cada descrição MDL de uma classe ou interface dá origem a um arquivo na linguagem Java, no caso *CustomerEJB.java*, *Customer.java* e *CustomerHome.java*.

Numa segunda alternativa, pode-se realizar a reimplementação através da ferramenta MVCASE que gera o código, conforme mostra a Figura 16. A geração do código EJB baseia-se no Modelo de classes dos componentes e no Modelo de componentes, especificados no reprojeto.

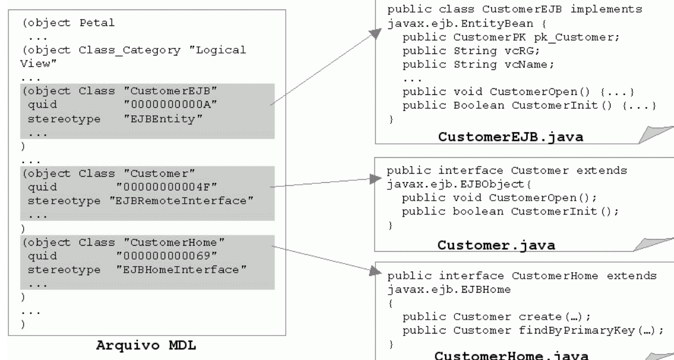

Figura 15 - Reimplementação usando Trasnformações

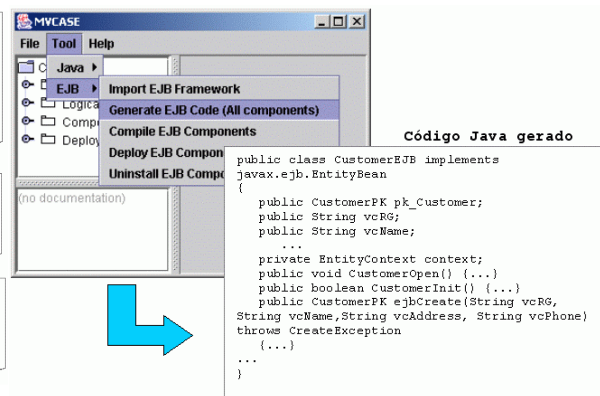

Figura 16 - Código gerado através da MVCASE

A Figura 17 mostra a arquitetura para execução do sistema reimplementado. Na camada (1) têm-se as aplicações Clientes, que podem ser JSPs ou Servlets em um Browser, via http usando um Servidor Web um Frame ou Applet, como aplicação Stand Alone. As aplicações Clientes acessam, via RMI, os componentes disponíveis no servidor J2EE, na camada Regras de Negocio (2). Os serviços de Banco de Dados ficam na camada (3), e podem ser acessados pelos componentes, através da comunicação do Servidor J2EE com o Servidor de Banco de Dados. Nesta arquitetura os componentes podem ser distribuídos, residindo em diferentes computadores.

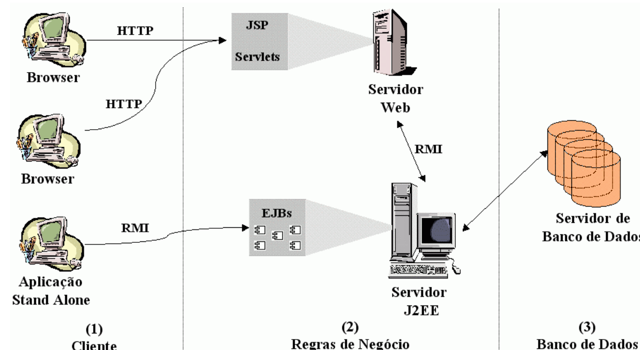

Figura 17 - Arquitetura do sistema reimplementado

Os resultados das execuções são analisados pelo Engenheiro de Software para verificar se atendem aos requisitos do sistema. Estes resultados fornecem um *feedback* aos passos anteriores, orientando nas correções para validar se a implementação atende aos requisitos especificados para o sistema.

A funcionalidade do sistema legado é mantida no sistema reimplementado com a vantagem de facilitar a manutenção, uma vez que o código está organizado e encapsulado em componentes. Como as transformações foram construídas preservando a semântica da linguagem origem na linguagem alvo da implementação, as falhas podem ser do próprio código legado ou das alterações introduzidas no reprojeto.

5 Trabalhos Relacionados

Em[16] os autores descrevem uma estratégia de desenvolvimento de sistemas distribuídos orientados a objetos, dividido em 3 passos: Especificação do Sistema Distribuído, onde o sistema é modelado usando a notação UML, Distribuição dos Objetos, onde é definida a arquitetura do sistema distribuído baseado nos serviços e frameworks disponíveis na plataforma JAMP (Java Architecture for Media Processing) e Implementação do Sistema Distribuído, onde é realizada a implementação baseado nas especificações dos passos anteriores. Esta estratégia difere do nosso trabalho, utilizando frameworks disponíveis na plataforma JAMP, onde toda a estratégia é suportada. Além disso, trabalha com objetos, não componentes, diminuindo o reuso no desenvolvimento das aplicações.

Em[17] é apresentado uma estratégia para desenvolvimento de sistemas baseados em componentes distribuídos, que usa Programação Orientada a Aspectos (AOP) para descrever e implementar as dependências entre os componentes. Comparado a esta proposta, o nosso trabalho parte da concepção do domínio, obtido pela recuperação do projeto do sistema legado, identificando todos os elementos até a implementação dos componentes de software. Obtendo assim, o reuso não só do código, mas também dos modelos de alto nível de abstração.

A Rational Rose[18] é uma ferramenta CASE que oferece uma imensa gama de recursos para suportar o processo de desenvolvimento de software baseado em componentes. Essa ferramenta suporta a especificação usando a notação UML e a geração do código dessas aplicações. Comparada a esta ferramenta, a MVCASE se destaca por possuir um repositório de componentes integrado, permitindo o armazenamento e busca de componentes para o reuso no desenvolvimento das aplicações. E por ser uma ferramenta acadêmica, é gratuita.

6 Conclusão

Este artigo apresentou uma estratégia para transformar sistemas legados, orientados a procedimentos, em sistemas orientados a componentes distribuídos, através do reprojeto usando componentes.

O sistema de transformação automatiza grande parte das tarefas da reengenharia, principalmente na organização do código legado e na geração das especificações e do código em uma nova linguagem. Uma das vantagens do uso do sistema de transformação vem da possibilidade de se trabalhar com descrições em uma linguagem, contendo descrições em outras linguagens, como no caso das especificações em *Catalysis*, contendo as mini especificações dos métodos, escritas em Java. Outra vantagem vem da capacidade de se trabalhar com linguagens de diferentes domínios de aplicação ou modelagem.

A estratégia cobre todo o ciclo da reengenharia de um sistema procedural, destacando-se a recuperação do projeto

do código legado permitindo o reprojeto orientado a componentes de software, em uma ferramenta CASE, com reimplementação semi-automática em uma linguagem orientada a objetos, através de uma ferramenta CASE ou através de transformações.

O aproveitamento das idéias do método de DBC Catalysis proporcionou o reuso não só de código, mas também de modelos em alto nível de abstração. O uso de componentes proporcionou ao sistema reconstruído maior qualidade e manutenibilidade. Após a reconstrução do sistema, os componentes construídos podem ser reutilizados para construir novas aplicações, adicionar novos requisitos ou modificar funcionalidades já existentes.

Outros dois estudos de casos foram realizados aplicando a estratégia em duas empresas. A primeira [19] aplicou a estratégia em um sistema legado (1.500.000 linhas) implementado em Progress4GL, obtendo um novo sistema em Java. A Segunda [20] aplicou a estratégia em um sistema implementado (5.3 milhões de linhas) em DataFlex Procedural, obtendo um sistema em Visual DataFlex.

A estratégia proposta dá mais um passo na automatização de grande parte das tarefas do Engenheiro de Software, o que pode contribuir na redução do tempo e custos de reconstrução de um sistema, através do reuso nos diferentes níveis de abstração, desde os requisitos, análise e projeto, até a implementação.

7 Referências

[1] GAMMA, E. et al. *Design Patterns*. **Elements of Reusable Object-Oriented Software**. Ed. Addison-Wesley. USA.1995.

[2] NEIGHBORS, J.M. **The Draco approach to Constructing Software from Reusable Components**. *IEEE Transactions on Software Engineering*. v.se-10, n.5, pp.564-574, September, 1984..

[3] SAMETINGER, J. **Software Engineering with Reusable Components**. Springer 1997

[4] D'SOUZA, D. and A. Wills, **"Objects, Components and Frameworks with UML–The Catalysis Approach"**, USA: Addison Wesley, 1995.

[5] PRADO, A. F., LUCRÉDIO, D; Ferramenta **MVCASE - Estágio Atual: Especificação, Projeto e Construção de Componentes** - Sessão de Ferramentas do XV Simpósio Brasileiro de Engenharia de Software - SBES'2001. Rio de Janeiro-RJ, Brasil. 3 – 5 de Outubro, 2001.

[6] Reasoning Systems Incorporated. *Refine User's Guide,* **Reasoning Systems Incorporated**, Palo Alto, 1992.

[7] WILE, D. *Popart:* **Producer of Parsers and Related Tools System Builders *Manual*.** Technical Report, *USC/Information Sciences Institute*, 1993.

[8] PRADO, A.F. **Estratégia de Engenharia de Software Orientada a Domínios.** Rio de Janeiro-RJ, 1992. Tese de Doutorado. Pontifícia Universidade Católica. 333p.

[9] BOOCH, G. et al. **The Unified Modeling Language** – User Guide. USA: Addison Wesley, 1999.

[10] **Enterprise Java Beans technology**, Sun Microsystems. URL: http://java.sun.com/products/ejb/index.html, 2001.

[11] CHESSMAN, J., Daniels, J., 2000. *UML Components:* **A Simple Process for Specifying Component-Based Software**. Addison-Wesley. USA, 1nd edition.

[12] ATKISON, C., et al, 2000. **Component-Based Software Engineering: The KobrA Approach.** In *ICSE'2000, 22th International Conference on Software Engineering, 3rd Workshop on Component-Based Software Engineering*. ACM Press.

[13] JACOBSON, I., et al., 2001. **The Unified Software Development Process**. Addison-Wesley. USA, 4nd edition.

[14] PENTEADO, R.D. **Um Método para Engenharia Reversa Orientada a Objetos**. São Carlos-SP, 1996. Tese de Doutorado. Universidade de São Paulo. 251p.

[15] Java Servlets URL: http://developer.java.sun.com/developer/onlineTraining/Servlets/Fundamentals

[16] GUIMARÃES, M., P., et al..1999. **Development of ObjectOriented Distributed Systems (DOODS) using Frameworks of the JAMP platform**. In Fisrt Workshop on Web Engineering, in conjunction with 19th International Conference in Software Engineering (ICSE).

[17] CLEMENT, P., J., Sánchez, F., Pérez, M., A., 2002. **Modeling with UML Component-based and Aspect Oriented Programming Systems**. In WCOP 2002. The 7th Workshop for Component-Oriented Programming. In conjuntion with the 16th European Conference on Oject-Oriented Programming (ECOOP).

[18] Rational Rose Tool, 2001. Rational the software development company. Available in 10/07/2001. URL: http://www.rational.com.

[19] NOGUEIRA, A. R. **,"Transformação de DataFlex Procedural para Visual DataFlex Orientado a Objetos reusando um Framework"**, São Carlos/SP, 2002, Dissertação de Mestrado. Universidade Federal de São Carlos.

[20] NOVAIS, E. R. A.; **"Reengenharia de Software Orientada a Componentes Distribuídos"**, São Carlos/SP, 2002, Dissertação de Mestrado, Universidade Federal de São Carlos.