

# Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações

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

## Resumo

Este artigo apresenta uma estratégia para Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações, 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**, que organiza código legado, usando transformações, de acordo com os princípios da Orientação a Objetos, segmentando o código em supostas classes, atributos e métodos; **Recuperar Projeto**, que parte do código legado organizado e também usando transformações de software, gera sua descrição na linguagem MDL (Modeling Domain Language), que importadas numa ferramenta CASE, recuperam o projeto do código legado; **Reprojetar**, onde o Engenheiro de Software reprojeta o sistema usando componentes de software distribuídos; e **Reimplementar**, onde a reimplementação do sistema é feita numa linguagem executável alvo da reimplementação.

**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 reuso 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 de Reengenharia de Sistemas Legados usando transformações, cujo objetivo é recuperar o modelo de análise de sistemas legados, reconstruindo-os para serem executados em novas plataformas de hardware e software. A estratégia suporta a manutenção do sistema, inclusive com alteração da sua estrutura original, garantindo sua evolução através do uso de componentes de software.

Um componente pode ser visto como um bloco de software, independente, contendo dados e operações, que pode ser utilizado na reconstrução de um sistema. Um dos atributos principais desejáveis em um componente de software relaciona-se a sua possibilidade de operação em diversos padrões de plataforma computacional.

Na estratégia, os componentes podem ser utilizados no reprojetar e reimplementação do sistema segundo o método *Catalysis*[3]. Esta alternativa permite ao Engenheiro de Software, remodelar o sistema para ser executado numa

arquitetura distribuída.

Para melhor compreensão da estratégia proposta, segue uma apresentação das principais tecnologias integradas para suportar a Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações.

## 2. Sistema Transformacional Draco

Diferentes sistemas de transformação têm sido usados em projetos de engenharia e reengenharia de software destacando-se o *IllumaSM*[4], *Popart*[5]. Outro importante sistema de transformação de software que vem sendo utilizado é o *Draco*[6,7,8,9,10].

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[10]. Com a estratégia proposta por Prado[10] 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 parser e Prettyprinter, ou unparser, definidos a partir desta gramática. 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.

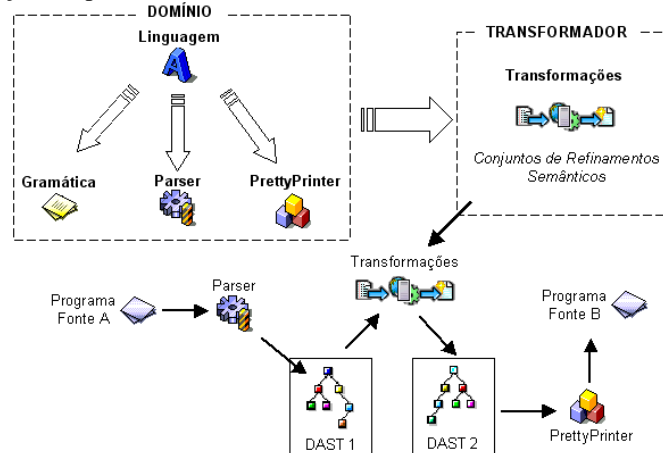


Figura 1- Partes de um Domínio no ST Draco

## 3. Ferramenta MVCASE

A MVCASE[11] é uma ferramenta CASE orientada a objetos que suporta a especificação de requisitos do sistema usando técnicas UML[12]. 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)[13] para implementação. O Engenheiro de Software pode transformar o Modelo de Objetos de um sistema em Modelo de Componentes Distribuídos, que podem ser do tipo *EJB Entity* ou *EJB Session*. 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.

## 4. Método DBC Catalysis

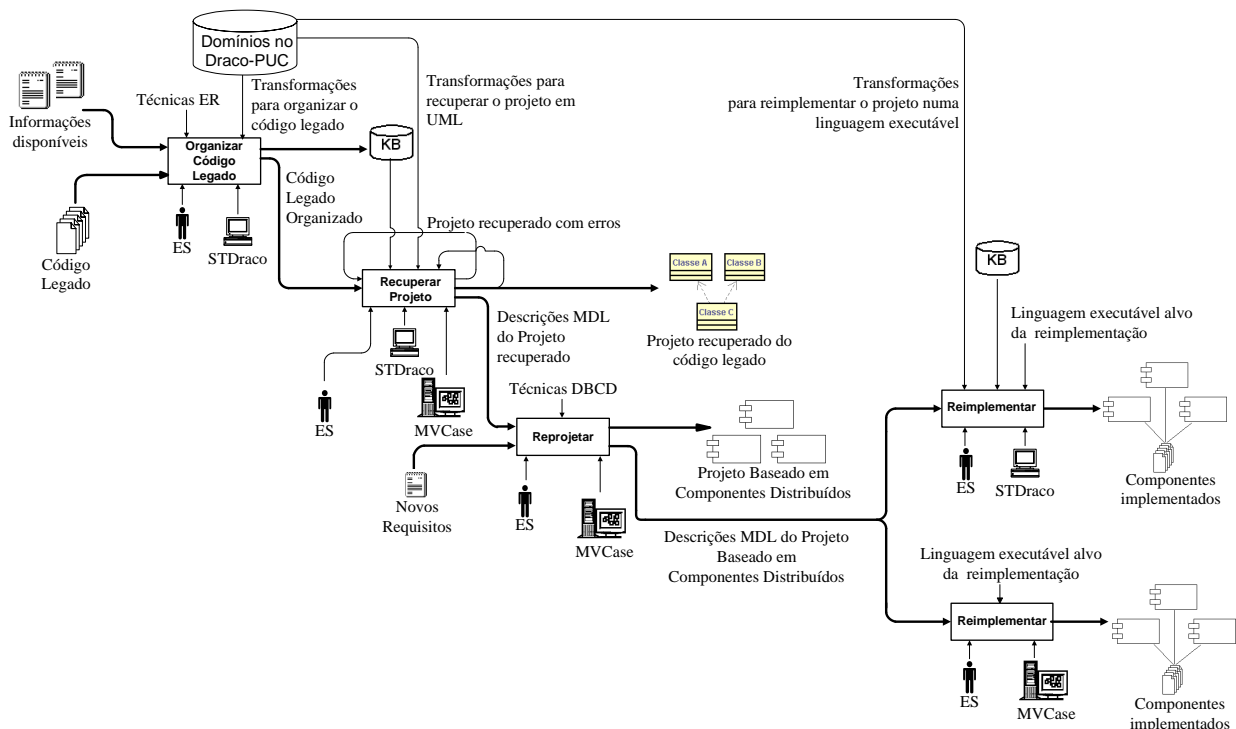
*Catalysis*[3] é um método para DBC que utiliza técnicas UML para modelar Sistemas com Componentes Distribuídos. Tem como base os princípios: Abstração, Precisão e Componentes Plug-In. O princípio *Abstração* orienta o Engenheiro de Software na busca dos aspectos essenciais do sistema, dispensando detalhes 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.

*Catalysis* 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.

*Catalysis* 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.

## 5. Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações

Combinando as tecnologias do sistema de transformação Draco[10], da ferramenta MVCASE[11], de ER[14] e do método DBC *Catalysis*[3], definiu-se uma estratégia para Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações, realizada em 4 passos: Organizar Código Legado, Recuperar Projeto, Reprojetar e Reimplementar, conforme mostra a Figura 2.



**Figura 2 – Estratégia de Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações**

### 5.1. 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, segundo os princípios da orientação a objetos. 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.

O Engenheiro de Software reúne 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.

Numa segunda etapa, do passo Organizar Código Legado, com base nas informações coletadas, na Base de Conhecimento, pelo primeiro transformador Intradomínio, aplica-se o segundo transformador Intradomínio, que segmenta 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. A organização do código fonte legado, segundo os princípios da Orientação a Objetos, facilita a transformação para UML, que é o objetivo do segundo passo da estratégia.

### 5.2. 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*).

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

### 5.3. Reprojetar

No terceiro passo, **Reprojetar**, o Engenheiro de Software com o auxílio da ferramenta MVCASE e de técnicas DBC Distribuídos, faz o reprojetado orientado a componentes distribuídos do projeto recuperado do código legado, obtendo um novo projeto baseado em componentes distribuídos.

O projeto recuperado serve como base para a especificação e projeto dos componentes. Este passo é direcionado pelo método de desenvolvimento Catalysis com seus três níveis: Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes.

No primeiro nível o Engenheiro de Software especifica os Modelos de Casos de Uso, que representam os cenários de uso do sistema. No segundo nível refinam-se os Modelos de Casos de Uso em Modelos de Tipos que representam atributos e operações, sem pensar na implementação. No terceiro nível refinam-se o Modelo de Tipos em Modelo de Classe, com seus atributos e métodos. Estas classes dão origem aos componentes. Para cada classe de um componente definem-se suas interfaces.

#### 5.4. Reimplementar

Finalmente, no quarto passo da estratégia, faz-se a reimplementação do sistema reprojeto. 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 geração do código EJB baseia-se no Modelo de classes dos componentes e no Modelo de componentes, especificados no reprojeto.

### 6. Estudo de Caso

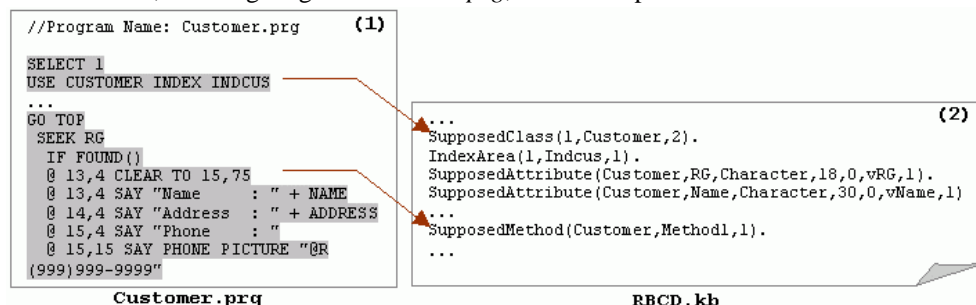
Trata-se de um sistema para Oficina Auto-Elétrica e Mecânica que controla os serviços executados nos veículos e o estoque das peças utilizadas. O cliente vai até a oficina para solicitar um serviço em seu veículo. Um cliente pode ter diversos veículos. O mesmo veículo pode voltar à oficina diversas vezes, 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. Os possíveis modelos de veículos devem ser cadastrados pelo sistema, pois são utilizadas tabelas para cobrança de serviços, tanto elétricos como mecânicos, de acordo com o modelo de veículo.

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.

#### 6.1. Organizar Código Legado

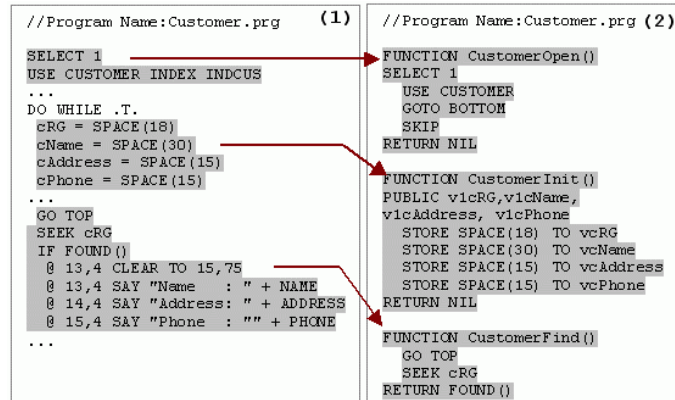
Inicialmente reuniu-se as informações disponíveis sobre o código legado, no caso apenas o código legado, e elaborou-se a relação *CHAMA/CHAMADO*, para determinar a sequência de programas a serem submetidos ao primeiro transformador, *ClipperToKB*, que organiza o código legado segundo os princípios da Orientação a Objetos. A Figura 3 mostra à direita a Base de Conhecimento após o reconhecimento da suposta classe *Customer* através dos comandos *SELECT* e *USE*, do código legado *Customer.prg*, e de seu suposto método *Method1*.



**Figura 3 - Identificação de uma suposta classe e de seus supostos atributos e métodos**

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

A Figura 4 mostra à esquerda (1) o programa *Customer.prg* e à direita (2) seu correspondente código organizado. O código permanece na mesma linguagem, porém com características da Orientação a Objetos. No caso, o código foi segmentado nas funções *CustomerOpen()*, *CustomerInit()* e *CustomerFind()*.



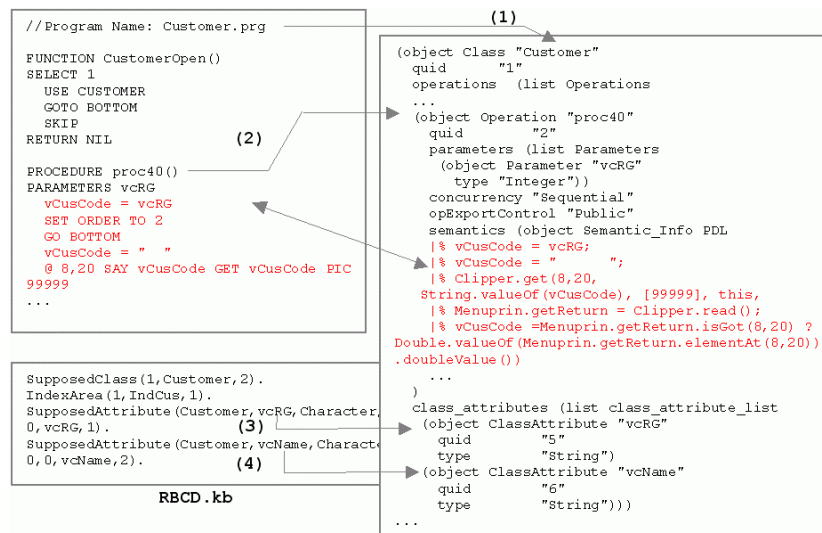
**Figura 4 - Segmentação do código legado segundo os princípios da Orientação a Objetos**

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.

## 6.2. 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 5 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. A base de conhecimento *RBCD.kb* é consultada para obter os fatos e regras sobre as supostas classes, atributos e métodos. 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 5 - 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 6. O projeto recuperado é representado por técnicas UML, em Modelos de classes e de Interações.

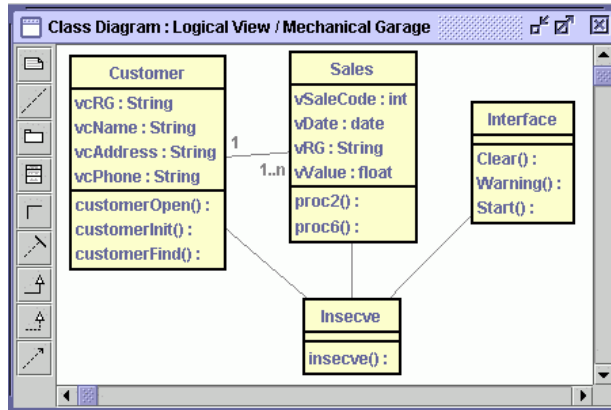


Figura 6 - Modelo de classes recuperado na MVCASE

### 6.3. Reprojetar

Neste passo o Engenheiro de Software pode reprojetar o projeto recuperado, por exemplo, usando componentes distribuídos.

Seguindo o método *Catalysis* pode-se refinar os modelos recuperados para obter um novo projeto reestruturado com componentes distribuídos, conforme mostra a Figura 7. Por exemplo, refinando o Modelo de classes da Figura 6 pode-se obter o Modelo de classes de componentes da Figura 7.3, referente ao nível Projeto Interno dos Componentes. No caso o objeto *Customer* foi transformado no componente *Customers*, segundo a tecnologia EJB. Foram criadas a classe persistente *CustomersEJB* e suas interfaces *CustomersHome* e *Customers*, que reutilizam as classes *EntityBean*, *EJBHome* e *EJBObject* da biblioteca Java/EJB importada na MVCASE. A Figura 7 mostra ainda o Ator *Customer* interagindo com o sistema através do caso de uso *AddServiceOrder* (1) e seus detalhes especificados no Diagrama de Sequência (2), modelos correspondentes aos níveis Domínio do problema e Especificação dos Componentes, respectivamente.

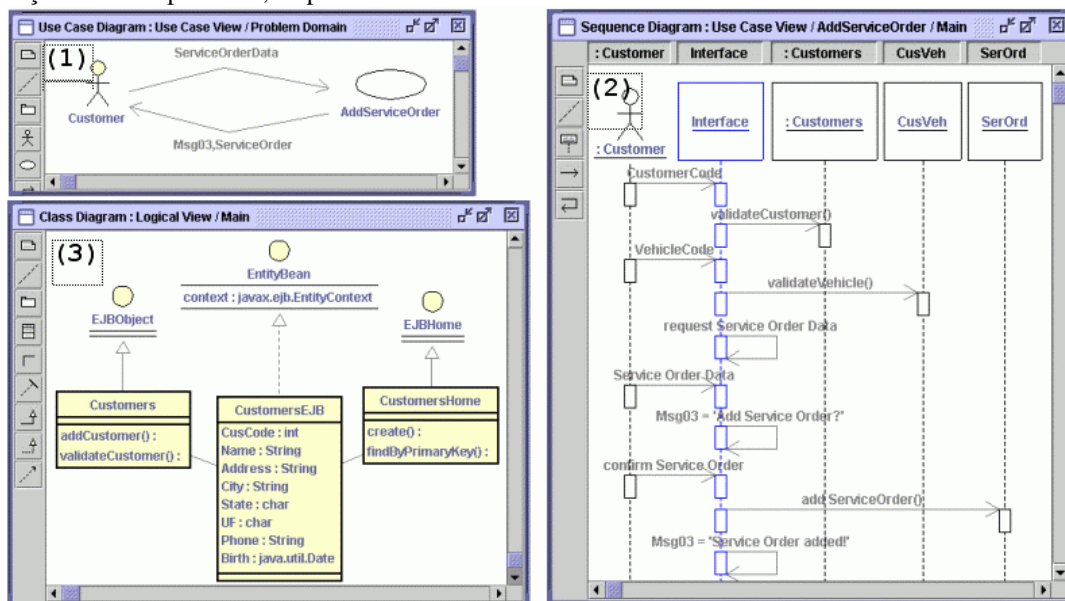


Figura 7 - Caso de Uso, Diagrama de Sequência e Diagrama de Classes do sistema reprojetado

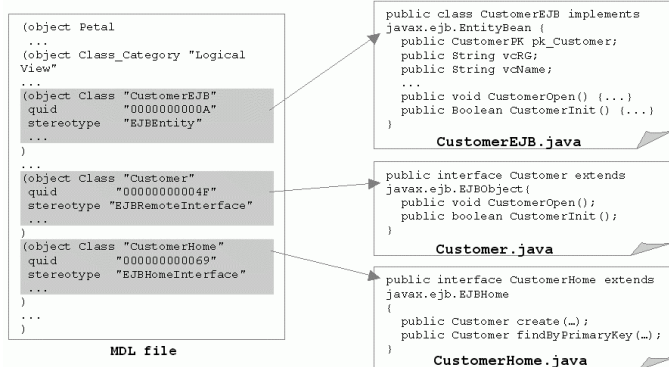
### 6.4. Reimplementar

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

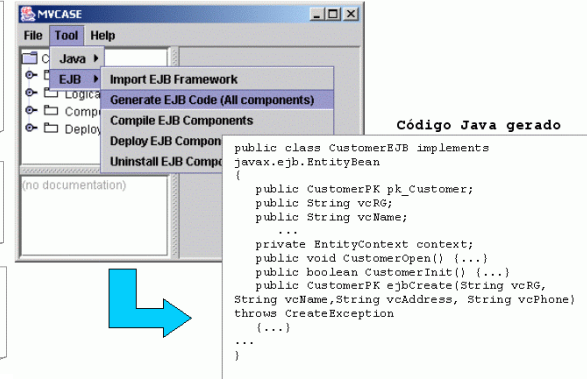


Por exemplo, usando o ST Draco, aplica-se o quarto transformador, Interdomínio, *MDLToJava*, que transforma as descrições MDL em código Java. A Figura 8 mostra à esquerda a descrição MDL e à direita o correspondente código Java/EJB gerado pelas transformações. Cada descrição MDL de uma classe ou interface dá origem a um arquivo na linguagem Java, no caso *CustomerEJB.java*, *Customer.java* e *CustomerHome.java*.

A reimplementação pode também ser feita através da ferramenta MVCASE que gera o código, conforme mostra a Figura 9. A geração do código EJB baseia-se no Modelo de classes dos componentes e no Modelo de componentes, especificados no reprojeto. Se o Engenheiro de Software mantiver o comportamento dos métodos em Java, o código gerado poderá ser mais completo, caso contrário, tem-se apenas o código das estruturas das classes com seus atributos e protótipo dos métodos.



**Figura 8 - Reimplementação usando Transformações**



**Figura 9 - Código gerado através da MVCASE**

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.

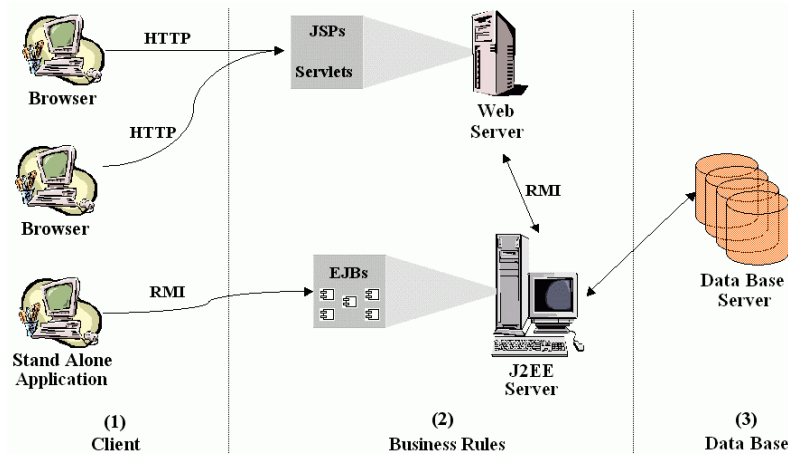
O tempo total para transformar o sistema de Auto-Elétrica e Mecânica de Clipper para Java, considerando a carregamento da Base de Conhecimento e organização do código, foi de 1 hora e 46 minutos, conforme mostra a tabela da Figura 10, com os tempos de execução das transformações de cada programa do sistema. Nesta experiência foi utilizada a MVCASE apenas para recuperar o projeto do código legado. Numa segunda experiência, a MVCASE foi usada tanto para recuperar o projeto quanto para gerar código Java, sem utilizar o transformador *MDLToJava*. Este tempo poderia se repetir para outros sistemas do mesmo porte. É verdade que este tempo não levou em conta o tempo gasto para construção dos domínios e das transformações, que normalmente leva cerca de 6 meses, devido a complexidade e a necessidade de um conhecimento profundo sobre os domínios a serem construídos. Contudo, a grande vantagem vem da reutilização das transformações em outros sistemas, quando então o tempo será de apenas algumas horas, enquanto que na reengenharia manual poderia levar meses com uma equipe de várias pessoas.

Código Legado	ClipperToKB	OrganizeClipper	ClipperToMDL	MDLToJava	Total
Catproc.prg	0:02:06	0:03:01	0:02:12	0:02:18	0:09:37
Indexa.prg	0:01:01	0:02:23	0:01:13	0:01:22	0:05:59
Modos1.prg	0:02:14	0:03:12	0:02:10	0:03:17	0:10:53
Modos2.prg	0:02:02	0:05:10	0:04:06	0:03:20	0:14:38
Ostela1.prg	0:02:03	0:03:11	0:02:05	0:02:10	0:09:29
Ostela2.prg	0:02:21	0:04:12	0:02:00	0:02:22	0:10:55
Elimios.prg	0:01:03	0:01:20	0:02:13	0:02:13	0:06:49
Altvcve.prg	0:01:09	0:02:10	0:01:26	0:01:16	0:06:00
Elicve.prg	0:01:25	0:01:14	0:02:10	0:01:07	0:05:56
Insecve.prg	0:01:07	0:02:05	0:03:12	0:02:16	0:08:40
Impos1.prg	0:02:08	0:03:05	0:01:25	0:01:02	0:07:40
Impos2.prg	0:02:26	0:02:01	0:01:15	0:01:33	0:07:15
Menuprin.prg	0:01:20	0:01:10	0:01:05	0:02:12	0:05:47
<b>Total</b>	<b>0:21:45</b>	<b>0:33:34</b>	<b>0:25:52</b>	<b>0:25:08</b>	<b>1:46:18</b>

**Figura 10 - Tempo de transformação da Auto-Elétrica e Mecânica**



A Figura 11 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 11 - Arquitetura do sistema reimplementado**

## 7. Conclusão

Este artigo apresentou uma estratégia para transformar código fonte de sistemas legados, orientados a procedimentos, para sistemas orientados a componentes distribuídos, usando transformações, para serem executados em plataformas modernas de hardware e software.

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.

Na ferramenta CASE, obtém-se o Projeto Recuperado do Código Legado. As descrições das especificações em *Catalysis*, que representam o projeto, persistidas em um arquivo pela ferramenta CASE, são utilizadas para gerar código do sistema em uma linguagem orientada a objetos. Recuperando o projeto do sistema pode-se trabalhar em alto nível de abstração, delegando a geração de código para o sistema de transformação. Caso necessário, novos requisitos podem ser adicionados ao sistema até obter a sua versão final. A reconstrução baseada em componentes, tomando como partida os modelos obtidos pelas transformações, é uma opção que pode tornar o sistema orientado a objetos distribuídos. A reimplementação fica facilitada com o reprojeto onde são especificados os requisitos não-funcionais do sistema, incluindo, se for o caso, o projeto interno dos componentes, como no caso de JavaEJB. Embora a ferramenta CASE, possa também gerar a implementação do sistema, a partir de suas especificações, a estratégia fica mais genérica e flexível deixando esta tarefa para o sistema de transformação onde se pode variar a linguagem alvo da reimplementação.

Outros dois estudos de casos foram realizados aplicando a estratégia em duas empresas. A primeira [9] aplicou a estratégia em um sistema legado implementado em Progress4GL, obtendo um novo sistema em Java. Esse sistema possui cerca de 1500 programas fontes, totalizando em torno de 1.500.000 linhas de código em *Progress*. A Segunda [8] aplicou a estratégia em um sistema implementado em DataFlex Procedural, obtendo um sistema em Visual DataFlex. Este sistema possui cerca de 1.900 programas totalizando 5.3 milhões de linhas de código.

Esses estudos mostraram a viabilidade de uso do Draco-PUC, com o auxílio dos scripts de transformação do DDE, por pessoas que não participaram do desenvolvimento das gramáticas e transformadores.

A estratégia de reengenharia 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.

## 8. 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] D’SOUZA, D. and A. Wills, **“Objects, Components and Frameworks with UML–The Catalysis Approach”**, USA: Addison Wesley, 1995.
- [4] Reasoning Systems Incorporated. *Refine User’s Guide*, Reasoning Systems Incorporated, Palo Alto, 1992.
- [5] WILE, D. *Popart: Producer of Parsers and Related Tools System Builders Manual*. Technical Report, USC/Information Sciences Institute, 1993.
- [6] FUKUDA, A. P. **Refinamento Automático de Sistemas Orientados a Objetos Distribuídos**. São Carlos/SP, 2000. Dissertação de Mestrado. Universidade Federal de São Carlos.
- [7] JESUS, E.S. **Engenharia Reversa de Sistemas Legados Usando Transformações**. São Carlos/SP, 2000. Dissertação de Mestrado. Universidade Federal de São Carlos.
- [8] 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.
- [9] NOVAIS, E. R. A.; **“Reengenharia de Software Orientada a Componentes Distribuídos”**, São Carlos/SP, 2002, Dissertação de Mestrado, Universidade Federal de São Carlos.
- [10] PRADO, A.F. **Estratégia de Engenharia de Software Orientada a Domínios**. Rio de Janeiro-RJ, 1992. Tese de Doutorado. Pontifícia Universidade Católica. 333p.
- [11] 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.
- [12] BOOCH, G. et al. **The Unified Modeling Language** – User Guide. USA: Addison Wesley, 1999.
- [13] **Enterprise Java Beans technology**, Sun Microsystems. URL: <http://java.sun.com/products/ejb/index.html>, 2001.
- [14] PENTEADO, R.D. **Um Método para Engenharia Reversa Orientada a Objetos**. São Carlos-SP, 1996. Tese de Doutorado. Universidade de São Paulo. 251p.