Reengenharia de Software para Plataformas Distribuídas Orientadas a Objetos

Elisângela Sato de Jesus
Ana Paula Fukuda
Antonio Francisco do Prado
e-mail: {apfukuda, esato, prado} @dc.ufscar.br
Universidade Federal de São Carlos
Departamento de Computação
C.P. 676 - 13565-905 - São Carlos (SP)

Resumo

Este artigo apresenta uma estratégia para Reengenharia de Software que reconstrói sistemas legados, tornando-os operacionais em novas plataforma de hardware e software. A partir do código fonte do sistema legado, é feita a sua reorganização segundo os princípios da orientação a objetos, usando técnicas da abordagem Fusion/RE. O código obtido é então transformado em especificações na linguagem de modelagem MDL. Estas especificações são importadas pela ferramenta CASE Rational Rose, obtendo-se o projeto orientado a objetos do sistema legado. Na ferramenta Rational Rose o engenheiro de software pode reprojetar o sistema, distribuindo seus objetos, utilizando técnicas do método Catalysis, para executar em uma arquitetura Cliente/Servidor. Após o reprojeto, o sistema orientado a objetos distribuído é persistido novamente em especificações na linguagem de modelagem MDL. Finalmente o sistema é reimplementado automaticamente, em uma linguagem orientada a objetos. Com o objetivo de explorar novas idéias na área de geradores de software, adotou-se o sistema transformacional Draco-PUC e a ferramenta Rational Rose como principais tecnologias deste projeto.

Palavras Chaves: Reengenharia de Software, Engenharia Reversa, Orientação a Objetos, Sistema Transformacional.

Abstract

This paper presents a reengineering strategy that rebuilds systems legacies, turning them operational in new hardware and software platforms. Starting from the source code of the legacy system its organization carried out according to the principles of object orientation, using techniques of the Fusion/RE approach. After organized according to the object orientation paradigm, the code is transformed to MDL modeling language specifications. These specifications are imported by the CASE tool Rational Rose, obtaining the oriented object project of legacy system. In Rational Rose tool the Software Engineer can redesign the system, distributing its objects in a Client/Server architecture using techniques of Catalysis method. After redesigned, the distributed oriented objects system is persisted again in MDL modeling language specifications. Finally, the system is automatically reimplemented in an object oriented language. Aiming to explore new ideas in the area of software generators, the transformational system Draco-PUC and the Rational Rose tool as main technologies for this project were adopted.

Key-Words: Software Reengineering, Reverse Engineering, Object Oriented, Transformational System

1. Introdução

Reengenharia de software, também chamada renovação ou recuperação de software, não apenas recupera informações do projeto de um sistema existente, como também usa estas informações para reconstruir o sistema, procurando melhorar sua qualidade global e reduzir custos com manutenção.

Chikofsky [3] define reengenharia como o processo de análise e alteração de um sistema, reconstruindo-o e reimplementando-o com novas tecnologias.

Segundo Jacobson [8] existem diferentes tipos de reengenharia: com a troca parcial ou completa da implementação sem a troca da funcionalidade, em que há mudança do paradigma de desenvolvimento e/ou da linguagem de implementação, preservando o funcionamento do sistema; e com troca da funcionalidade, em que se aplica o processo tradicional de desenvolvimento de software, modificando o funcionamento do sistema no modelo de análise e implementado-o novamente.

O objetivo de aplicar técnicas de reengenharia de software em sistemas é facilitar sua evolução disciplinada, desde o estado corrente até o novo estado desejado. Isto é possível através das operações de engenharia avante, de transformação e de engenharia reversa, que agem nos diferentes níveis de abstração do ciclo de vida do software [20]. A operação de engenharia avante é o processo tradicional de desenvolvimento de software que parte de um alto nível de abstração, passando pela análise de requisitos e projeto até a implementação do sistema. A operação de transformação refere-se ao processo de migração de sistemas, de antigos ambientes computacionais para modernas plataformas computacionais, mantendo a mesma funcionalidade e os mesmos dados do sistema legado[1]. A engenharia reversa é uma operação contrária à engenharia avante, que tem início no código fonte de implementação do sistema legado, recuperando ou recriando seu projeto e compreendendo os requisitos que foram implementados.

Este artigo apresenta uma estratégia de reengenharia cujo o objetivo é reconstruir sistemas legados para serem executados em novas plataformas de hardware e software. A estratégia, que parte do código fonte do sistema legado e das informações, caso estejam disponíveis, recupera o projeto do sistema, reprojeta-o para uma plataforma distribuída e faz sua reimplementação automática em uma linguagem orientada a objetos.

O artigo está organizado da seguinte forma: a seção 2 apresenta as principais técnicas usadas na estratégia de reengenharia, a seção 3 apresenta a estratégia proposta da Reengenharia de Software para Plataformas Orientadas a Objetos Distribuídas, a seção 4 apresenta a utilização da estratégia através de um estudo de caso e finalmente, a seção 5 apresenta as conclusões desta pesquisa.

2. Principais Técnicas Usadas na Estratégia de Reengenharia

Os sistemas legados, na maioria dos casos, não possuem documentação e quando possuem, estas podem estar desatualizadas, devido às possíveis modificações realizadas no sistema e que não foram adicionadas à documentação, o que dificulta o trabalho de manutenção. Para manter estes sistemas, técnicas de reengenharia de software são aplicadas ao código fonte, com o objetivo de reconstruí-los.

Como a reconstrução manual de sistemas legados é um processo lento e trabalhoso, pesquisas estão sendo direcionadas para a utilização de tecnologias de sistemas transformacionais

[1] Sistema implementado em um ambiente computacional antigo e que ainda se encontra em uso.

na área de reengenharia de software. Um sistema transformacional que vem sendo utilizado nesta área é o Draco-PUC, como pode ser constatado em [1, 17].

O Draco-PUC [10, 11, 13, 16] implementa as idéias de transformação orientada a domínios. Pela estratégia proposta por Prado [16], é possível a reconstrução de um software pelo "porte" direto do código fonte de uma linguagem para linguagens de outros domínios. Um domínio, de acordo com o paradigma Draco, é constituído de três partes: uma **linguagem**, definida por um *parser* que é responsável por analisar os sistemas da linguagem e gerar a representação interna do Draco, uma *Abstract Syntax Tree (AST),* denominada no Draco de DAST; um ***prettyprinter*** ou *unparser,* que realiza a formatação da DAST, tornando-a novamente textual na linguagem do domínio; e um ou mais **Transformadores**, que mapeiam estruturas de uma linguagem para estruturas na mesma linguagem do domínio, chamados de transformadores Intra-Domínio, e que mapeiam as aplicações descritas na linguagem de um domínio para descrições de uma linguagem de outro domínio, chamados transformadores Inter-Domínios. Os transformadores são responsáveis pela automatização ou semi-automatização do processo de construção de software.

Um transformador possui um conjunto de transformações. Uma transformação é composta basicamente pelos pontos de controle LHS (*Left Hand Side*) e RHS (*Right Hand Side*). O LHS, ou padrão de reconhecimento, define a sintaxe da linguagem fonte para a transformação e o RHS (*Right Hand Side*), ou padrão de substituição, define a sintaxe da linguagem alvo para a transformação. Além dos padrões de reconhecimento e de substituição, uma transformação pode conter outros pontos de controle, aos quais pode-se associar código para o desempenho de tarefas, relacionadas com pré e pós condições da transformação.

Outra técnica usada na estratégia de reengenharia é a abordagem Fusion/RE [14, 15] que faz a recuperação do projeto atual do sistema legado.

A abordagem Fusion/RE apresenta técnicas para recuperar sistemas legados utilizando o paradigma da orientação a objetos, sem que os mesmos tenham sidos desenvolvidos com essa tecnologia. Esta abordagem tem como objetivo recuperar informações e gerar os mesmos modelos propostos pelo método Fusion [4], para facilitar a compreensão do sistema que está sendo analisado. Após a geração dos modelos, estes são utilizados para orientar o engenheiro de software no processo de organização do código, dando a ele características de orientação a objetos. O processo de organização de código é denominado segmentação e o seu resultado, o sistema já organizado, é denominado sistema segmentado.

Para o reprojeto do sistema a estratégia usa técnicas do método *Catalysis*[5] para desenvolvimento de software orientado a objetos que utiliza Componentes Distribuídos, *Design Patterns* [5, 7] e *Frameworks* [24]. Este método suporta características de métodos de desenvolvimento de software recentes e sua notação é baseada na *Unified Modeling Language* (UML) [21, 22]; é um método baseado nos princípios de **abstração**, **precisão** e **componentes "plug-in"**. O princípio **abstração** orienta o desenvolvedor na busca dos aspectos essenciais, dispensando detalhes que não são relevantes para o contexto do sistema. O princípio **precisão** visa descobrir erros e inconsistências no projeto e o princípio **componentes "plug-in"** busca o reuso de componentes para construir outros componentes.

O processo de desenvolvimento de software em *Catalysis* é dividido em três níveis: Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes. Estes níveis correspondem às fases tradicionais do ciclo de vida do software: elicitação, análise e projeto dos requisitos. Assim como no ciclo tradicional de desenvolvimento de software, em *Catalysis*, pode-se retornar à fase anterior para corrigir erros e inconsistências entre os modelos gerados em cada fase.

No nível **Domínio do Problema** é dada ênfase no entendimento do problema, isto é, especifica-se "o quê" o sistema deve atender para solucionar o problema. Nesta fase, utilizam-se técnicas de entrevista coletiva e informal com o usuário, como *brainstorming* [2], que são documentados com *storyboards*[5] e *mind maps*[5]. ***Storyboards*** são representações de diferentes situações e cenários no domínio do problema. ***Mind maps*** são representações estruturadas dos termos importantes do domínio do problema, relatados na entrevista com os usuários. Essas representações identificam a ligação entre os termos obtidos no *brainstorming,* definindo assim, uma terminologia do domínio do problema.

Uma vez definida a terminologia do domínio do problema, pode-se criar modelos de colaboração e de *use case* [9], que representam os atores (agentes externos do sistema) e suas interações com o sistema. Há dois tipos de modelos de colaboração do domínio do problema: "*as-is*" e "*as-be*". O modelo "*as-is*" descreve "como é" o sistema atual no Domínio do Problema. O modelo "*as-be*", também chamado de Modelo de Tipos, descreve "como será" o sistema a ser desenvolvido no Domínio do Problema.

Descrito o modelo "as-is" e estabelecido o modelo "*as-be"*, definem-se as fronteiras, os atores e as ações externas que estimulam o sistema, num *Diagrama de Contexto* (DC).

O nível **Especificação dos Componentes** descreve o comportamento do sistema de uma forma não ambígua. Este nível tem início com o mapeamento do modelo de Tipos obtido no Domínio do Problema, para um Modelo de Tipos que especifica o comportamento dos objetos. Esse modelo mostra os atributos e as operações dos tipos de objetos, sem se preocupar com a implementação. Portanto o projeto funcional é independente do aspecto físico da arquitetura distribuída, simplificando a transição para quaisquer arquiteturas distribuídas, que é a preocupação do próximo nível. A partir do Modelo de Tipos constróem-se Diagramas de Seqüência que têm por objetivo mostrar os cenários de execução das operações ao longo do tempo. Outras técnicas, como *Statecharts*[Dso98] *e Snapshops*[5], podem ser utilizadas para a especificação dos componentes. Ainda nesse nível utiliza-se a linguagem *Object Constraint Language* - OCL [22], que tem sintaxe baseada em objetos, para escrever detalhes das especificações dos componentes com base nos Modelos de Tipos.

No último nível, **Projeto Interno dos Componentes**, define-se "como" serão implementados os requisitos especificados do sistema, preocupando-se com a distribuição física dos objetos. Este nível inicia com a definição do Diagrama de Componentes e com o Modelo de Classes do Sistema, mostrando as classes com seus atributos, operações, relacionamentos e a distribuição dos objetos. Esse modelo é derivado do Modelo de Tipos do nível Especificação dos Componentes. Nesse nível, utiliza-se o Diagrama de Interação, que é derivado do Diagrama de Seqüência obtido na Especificação dos Componentes, para mostrar a interação entre os objetos. Técnicas para representar a arquitetura das plataformas Física e Lógica do sistema também são usadas neste nível.

Combinando a tecnologia do sistema transformacional Draco-PUC com técnicas da abordagem Fusion/RE e do método *Catalysis* propõe-se uma estratégia de reengenharia, que organiza o código fonte do sistema legado, recupera seu projeto, reprojeta-o para ser executado em uma plataforma distribuída e finalmente faz a sua reimplementação em uma linguagem orientada a objetos apresentada a seguir.

3. Reengenharia de Software para Plataformas Distribuídas Orientadas a Objetos

Integrando o sistema transformacional Draco-PUC com técnicas de Fusion/RE e da ferramenta CASE *Rational Rose* propõe-se uma estratégia de Reengenharia de Software para Plataformas Distribuídas Orientadas a Objetos, que é realizada em quatro passos: **Segmentar**, **Transformar**, **Reprojetar** e **Reimplementar** sistema, como mostra a Figura 1.

No primeiro passo da estratégia, **Segmentar Sistema**, faz-se a organização do Código Fonte do Sistema Legado segundo o paradigma da orientação a objetos, utilizando um Transformador Intra-Domínio, construído no sistema transformacional Draco-PUC. Este transformador intra-domínio, contém transformações que mapeiam a sintaxe e a semântica dos comandos da linguagem do Código Fonte do Sistema Legado, para o Código Fonte Segmentado, implementado na mesma linguagem, mas organizado segundo os princípios da orientação a objetos. A descrição das transformações, foi orientada por técnicas da abordagem Fusion/RE e pela sintaxe e semântica da linguagem do Código Fonte do Sistema Legado.

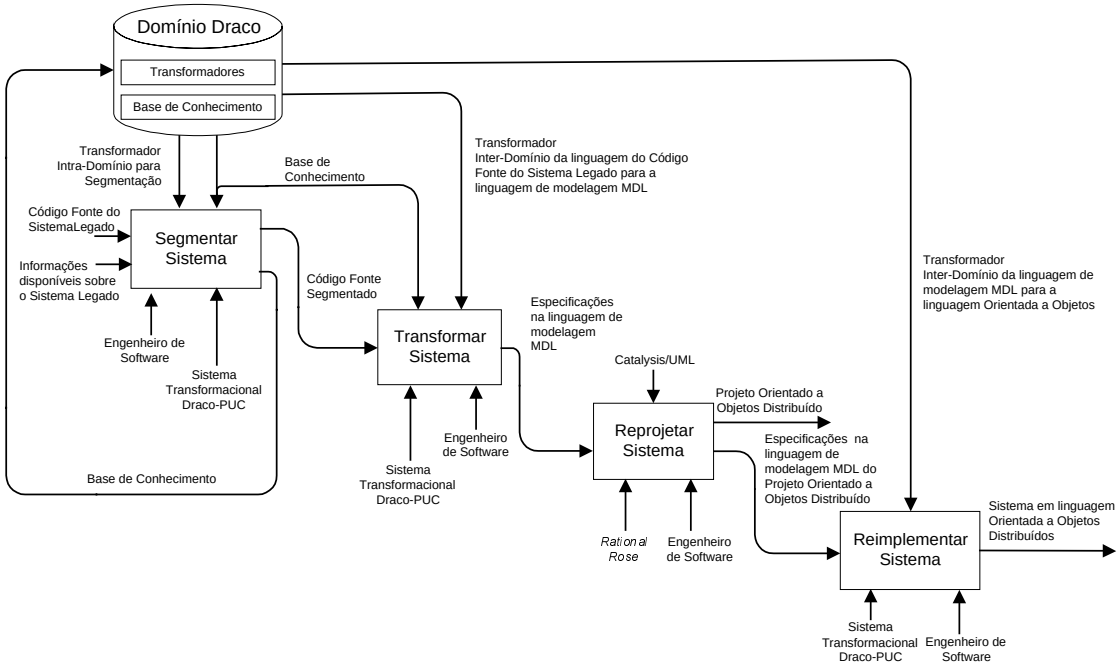

Figura 1 – Estratégia de Reengenharia de Software para Plataforma Distribuída Orientada a Objetos

A segmentação do sistema utiliza técnicas da abordagem Fusion/RE para identificar estruturas no Código Fonte do Sistema Legado que possuem correspondentes no paradigma da orientação a objetos. O domínio da linguagem do Código Fonte do Sistema Legado orientou na definição das regras de reconhecimento e de substituição para as transformações. Como saídas deste passo têm-se o Código Fonte do Sistema legado organizado segundo os princípios da orientação a objetos, denominado Código Fonte Segmentado, e uma Base de Conhecimento também é gerada para armazenar informações obtidas durante a análise do Código Fonte do Sistema Legado. Estas informações resolvem problemas relacionados com as diferenças sintáticas e semânticas entre os paradigmas procedural e orientado a objetos. Neste passo necessita-se da intervenção humana para tomadas de decisões, tornando-o semi-automático.

No segundo passo da estratégia, **Transformar Sistema**, faz-se a transformação do Código Fonte Segmentado, implementado na mesma linguagem do sistema legado, para especificações na linguagem de modelagem da ferramenta CASE *Rational Rose* [18] denominada MDL [6]. Neste passo utiliza-se, no sistema transformacional Draco-PUC, o Transformador Inter-Domínios da linguagem do Código Fonte do Sistema Legado para a linguagem de modelagem MDL, cujas transformações fazem automaticamente o mapeamento sintático e semântico do Código Fonte Segmentado para Especificações na linguagem de modelagem MDL. As transformações foram descritas com base nos domínios da linguagem do sistema legado e da linguagem de modelagem MDL. O domínio da linguagem do sistema

legado orientou na definição das regras de reconhecimento das transformações. O domínio da linguagem de modelagem MDL orientou na definição das regras de substituição das transformações. Quando uma regra de transformação reconhece um padrão sintático do código fonte legado, este é substituído pelo seu correspondente padrão sintático, definido na linguagem de modelagem MDL.

A Base de Conhecimento obtida no passo anterior, Segmentar Sistema, também é consultada neste passo, para a resolução das diferenças sintáticas e semânticas nas transformações.

No terceiro passo da estratégia, **Reprojetar Sistema,** o Engenheiro de Software, usando a ferramenta CASE *Rational Rose*, importa as especificações na linguagem de modelagem MDL. Com o projeto do sistema representado graficamente na ferramenta *Rational Rose*, torna-se possível compreender e reprojetar o sistema com as técnicas do método *Catalysis* [5] para realizar a distribuição dos objetos, corrigir erros e atender aos novos requisitos.

O método *Catalysis* é utilizado para reprojetar o sistema, segundo seus três níveis: Domínio do Problema, Especificação e Projeto Interno dos Componentes. Como saídas deste passo tem-se o Projeto Orientado a Objetos Distribuído do sistema e suas especificações na linguagem de modelagem MDL.

Finalmente, no quarto passo da estratégia, **Reimplemetar Sistema,** faz-se a reimplementação em uma linguagem orientada a objetos. Este passo usa um Transformador Inter-Domínios, que faz a implementação automática da especificação da linguagem de modelagem para uma linguagem orientada a objetos. A construção deste transformador foi orientada pelos domínios da linguagem de modelagem MDL e da linguagem orientada a objetos. O domínio da linguagem de modelagem MDL orientou na definição das regras de reconhecimento para as transformações. O domínio da linguagem orientada a objetos orientou na definição das regras de substituição das transformações. Neste passo, o sistema transformacional Draco-PUC é utilizado pelo engenheiro de software para transformar automaticamente as especificações na linguagem de modelagem MDL para a linguagem orientada a objetos.

A próxima seção apresenta um estudo de caso que mostra o uso da estratégia apresentada.

4. Estudo de Caso

Trata-se de um sistema verídico, utilizado para controlar os serviços e o estoque de peças de uma oficina auto-elétrica e mecânica. Este sistema foi usado para testar a estratégia porque está implementado na linguagem Clipper [19, 23], que é uma linguagem não orientada a objetos e comumente utilizada no mercado para a implementação de sistemas de informação. A Figura 2 apresenta uma descrição resumida do seu funcionamento.

Quando um cliente solicita a realização de serviços de auto-elétrica ou mecânica, é levado em consideração que ele pode ter mais de um veículo e que o mesmo veículo pode voltar a oficina mais de uma vez. Cada vez que um veículo for reparado é gerada uma Ordem de Serviço. A Ordem de Serviço é completada com a discriminação das peças utilizadas e da mão-de-obra executada. Muitas vezes o reparo pode exigir peças não existentes no estoque, que são adquiridas fora da oficina e também acrescentadas à Ordem de Serviço. O registro destas peças é importante para o gerente da oficina, pois elas são candidatas a serem estocadas no futuro. O modelo de veículo é importante para estimar o valor dos serviços prestados, baseado em uma tabela de cobrança dos serviços por modelo de veículo.

Figura 2 - Sistema para oficina Auto-Elétrica e Mecânica

Segue-se uma explicação de cada passo da estratégia da reengenharia proposta, usando o estudo de caso.

4.1. Segmentar Sistema

A aplicação da estratégia tem início com o código fonte Clipper do sistema legado. No primeiro passo da estratégia, **Segmentar Sistema**, faz-se a segmentação do código legado, segundo os princípios da Orientação a Objetos. A parte sombreada da Figura 3 mostra um fragmento do código fonte do sistema legado da oficina, em Clipper, usado para explicar a estratégia de reengenharia.

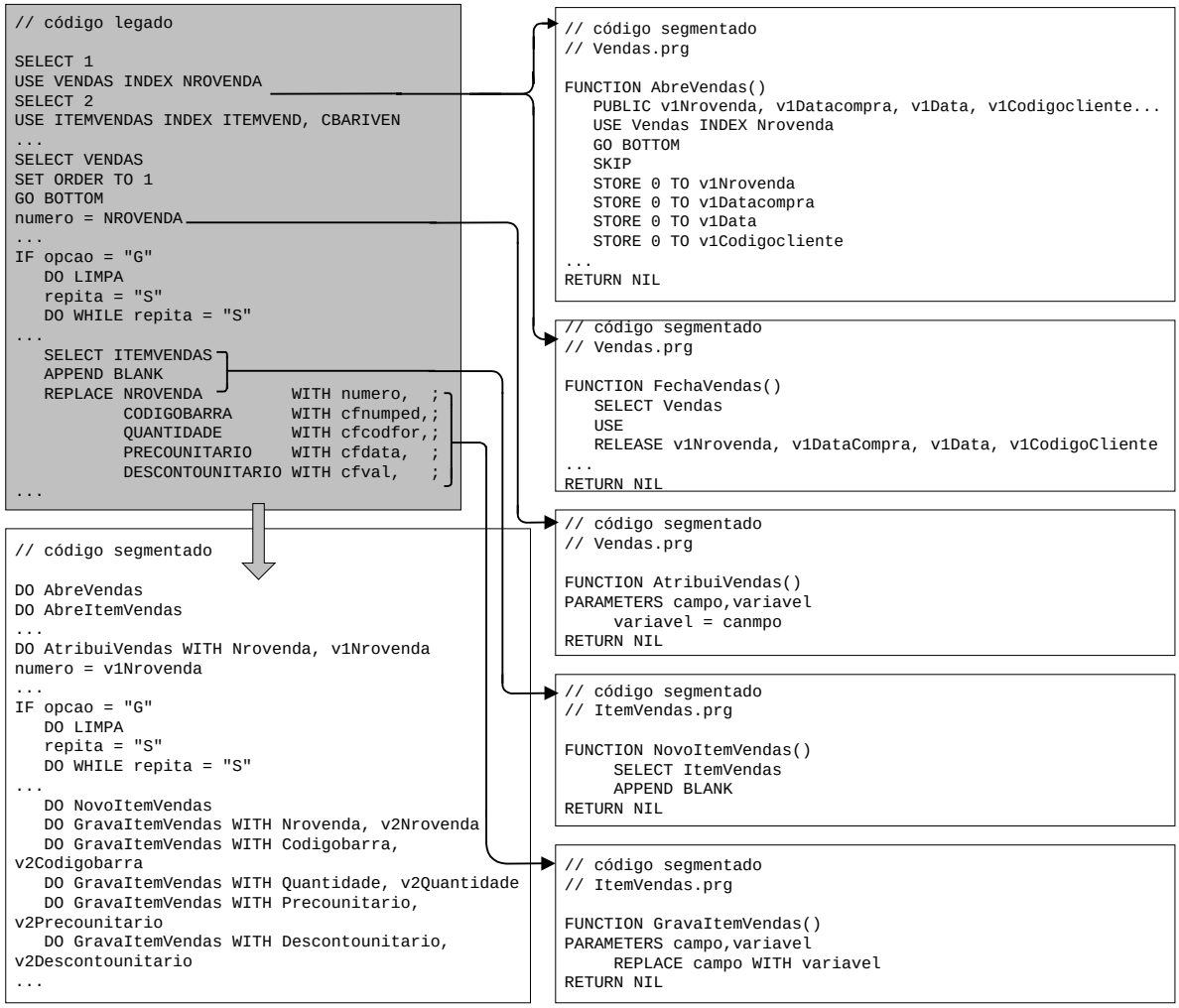

Figura 3 – Segmentação do código fonte do sistema legado

A segmentação do Código Fonte do Sistema Legado é realizada pela aplicação do Transformador Intra-Domínio, no sistema transformacional Draco-PUC, que organiza o código Clipper, segundo os princípios da orientação a objetos. Por exemplo, a partir do comando `USE VENDAS`, que faz a abertura do arquivo de dados `Vendas.dbf` no código fonte do sistema legado, é criado o arquivo `Vendas.prg`, que contém funções, no código fonte segmentado, que alteram e/ou consultam o arquivo de dados `Vendas.dbf`, como `AbreVendas()` e `FechaVendas()`. O arquivo `Vendas.prg` corrensponde a uma

suposta classe, segundo a orientação a objetos, e as funções `AbreVendas()` e `FechaVendas()` correspondem a supostos métodos. A função `AbreVendas()` faz a declaração e a inicialização das variáveis de memória que irão representar, ao longo do código fonte segmentado, os valores dos campos do arquivo de dados `Vendas.dbf`. A função `FechaVendas()` elimina essas variáveis da memória. Outras funções, além de `AbreVendas()` e `FechaVendas()` também são adicionadas em `Vendas.prg`, como por exemplo, `AtribuiVendas()`. Esta função atribui valores dos campos do arquivo de dados `Vendas.dbf` para variáveis de memória, como é o caso da atribuição do campo `Nrovenda` para a variável de memória `v1Nrovenda`, ambos passados como parâmetros para a função `AtribuiVendas()`, no código fonte segmentado. A função `AtribuiVendas()` foi criada quando se identificou a atribuição do campo `NROVENDA` para a variável `numero`, no código fonte do sistema legado. Outro exemplo é o comando `APPEND BLANK`, que adiciona um registro vazio a um arquivo, seguido de comandos `REPLACE`, que gravam os dados neste registro. Neste caso, o transformador cria funções como `NovoItemVendas()` e `GravaItemVendas()`. A função `NovoItemVendas()`, chama o comando `APPEND BLANK`. A função `GravaItemVendas()`, recebe uma variável e um campo do arquivo de dados, passados como parâmetros, que são utilizados no comando `REPLACE`. Adicionalmente, cada comando do Código Fonte do Sistema Legado que deu origem a uma função, é substituído pelo comando `DO`, seguido do nome da função. Por exemplo, a função `DO NovoItemVendas()`, no código fonte segmentado, substitui o comando `APPEND BLANK`, do código fonte do sistema legado.

A Figura 4 mostra parte do Transformador Intra-Domínio usado na Segmentação. Como se trata de um transformador intra-domínio, as transformações usam nos pontos de controle de reconhecimento (LHS) e de substituição (RHS), os padrões sintáticos da mesma linguagem, no caso Clipper. Os demais pontos de controle são utilizados para armazenar e recuperar fatos na Base de Conhecimento, para a resolver os problemas relacionados com as diferenças semânticas entre o código procedural e o orientado a objetos e para solicitar a tomada de decisões.

Por exemplo, a transformação da Figura 4 reconhece uma suposta classe. No ponto de controle `LHS`, tem-se como padrão de reconhecimento a sintaxe do comando de abertura de arquivo, `USE [[ID nome_arq]]`. Este comando cria o arquivo `.prg`, cujo nome é especificado pela meta-variável `nome_arq`. De acordo com a transformação, assim que o comando `USE` é identificado, são criadas as funções para a declaração, inicialização e remoção das variáveis de memória, como mostra os padrões de substituição nos pontos de controle `RHS`, que usam os `TEMPLATES`[2] `T_abre` e `T_fecha`, respectivamente. Além dos padrões de reconhecimento, LHS, e de substituição, RHS, são mostrados no ponto de controle `POST-MATCH`, os comandos para armazenar e recuperar fatos na Base de Conhecimento, que são: `KBSolve`, que faz a busca por um fato; `KBRetrieve`, que recupera o fato; `KBDelete`, que elimina o fato; `KBAssertIfNew`, que armazena um novo fato; e `KBWrite`, que grava as modificações feitas na Base de Conhecimento.

Aplicadas as transformações no código fonte do sistema legado, com o sistema transformacional Draco-PUC, obtém-se o correspondente código fonte segmentado, que será utilizado no próximo passo da estratégia.

[2] Padrão formatado de uma dada categoria sintática.

4.2. Transformar Sistema

Para recuperar as informações do sistema, em um alto nível de abstração, faz-se no passo **Transformar Sistema,** a transformação do código fonte segmentado, obtido no passo anterior, para especificações na linguagem de modelagem MDL. O engenheiro de software utiliza a ferramenta *Rational Rose* para importar as descrições em MDL e obter o projeto do sistema atual. Utiliza-se para este passo, o Transformador Inter-Domínios da linguagem do Código Fonte do Sistema Legado para a linguagem de modelagem MDL. Este transformador reconhece cada comando no código fonte segmentado, e o transforma para o seu correspondente na linguagem de modelagem MDL.

```
TRANSFORM use_stat1
LHS: {{dast clipper.statements
        USE [[ID nome_arq]]
}}

POST-MATCH: {{dast txt.decls
...
  sprintf(classcorrente, "%s", expand("[[nome_arq]]"));
  sprintf(query, "Classe(%s, *y)", classcorrente);
  if (KBSolve(query)) {
    sprintf(numclass, "%s", KBRetrieve("*y", 1));
    KBDelete(query);
    KBAssertIfNew("Classe([[classcorrente]], [[numclass]])");
    KBWrite("UtilityRss.kb");
  }
  else {
    ...
    Read_Class(arquiv);
    ...
    get = Get_Atrib(contrec);
    while (get == 1) {
      ...
      KBAssertIfNew("atributo([[classcorrente]], [[natrib]],
[[tatrib]], [[watrib]], [[datrib]], [[varmem]])");
      KBWrite("UtilityRss.kb");
...

TEMPLATE T_abre
RHS: {{dast clipper.prog_head
FUNCTION [[ID abrfunc]] ()
        PUBLIC [[decl_atrib* list_var]]
      USE [[ID classcorrente]] NEW
        GOTO BOTTOM
        SKIP
        [[body_struct* list_store]]
RETURN NIL
}}

TEMPLATE T_fecha
RHS: {{dast clipper.prog_head
FUNCTION [[ID fecfunc]] ()
  SELECT [[ID classcorrente]]
  USE
  RELEASE [[iden* list_var]]
RETURN NIL
}}
```

Figura 4 – Parte do Transformador Intra-Domínio usado na segmentação

Por exemplo, o transformador inter-domínios, ao encontrar o comando `USE` dentro de uma função de abertura de arquivo, no código fonte segmentado, dá origem a especificação de uma classe, conforme mostra a Figura 5. O comando `USE Vendas` dentro da função `AbreVendas()`, dá origem à especificação da classe `Vendas` em MDL, descrita com `object Class "Vendas"`. Cada função em Clipper é transformada para um método de uma classe em MDL. Por exemplo, a declaração da função `AtribuiVendas()`, no código fonte segmentado, dá origem à especificação do método em MDL, `object Operation "atribuiVendas"`. Os atributos das classes criados quando o transformador reconhecer os comandos que inicializam as variáveis de memória. Por exemplo, o comando `STORE 0 TO v1NroVenda`, no código fonte segmentado, dá origem à especificação do atributo em MDL, `object ClassAttribute "NroVenda"`, uma vez que a variável de memória `v1NroVenda` está associada ao campo `NroVenda`, em um fato armazenado na Base de Conhecimento.

A linguagem de modelagem MDL possibilita que qualquer miniespecificação seja inserida nas pré-condições, pós-condições e semântica dos métodos. Para formalizar estas miniespecificações, foi utilizada uma linguagem pseudo-código denominada Linguagem Básica de Ensino (LBE) [12, 16]. Assim, o comportamento detalhado dos métodos foram capturados e transformados para miniespecificações, escritas em pseudo-código. A integração

das duas linguagens, MDL e LBE, foi possível, porque o Draco-PUC suporta descrições em múltiplas linguagens, como no caso da MDL como linguagem hospedeira e a LBE como linguagem embutida na MDL. Por exemplo, a Figura 5 mostra, em sombreado, a transformação do código fonte segmentado para descrições LBE, inseridas na semântica de um método.

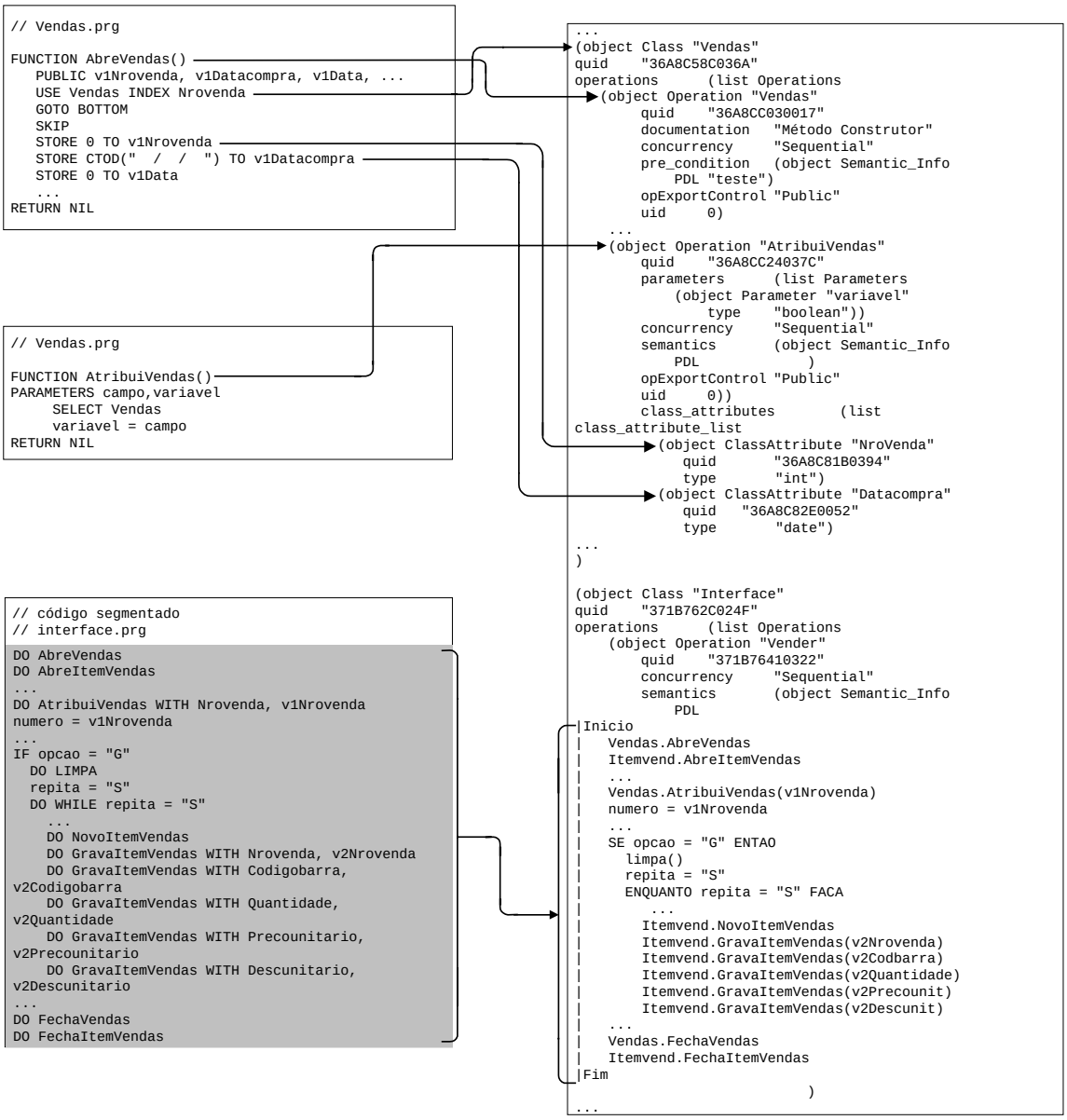

Figura 5 – Transformação do código fonte segmentado em especificações na linguagem de modelagem MDL

O Transformador Inter-Domínios da linguagem do Código Fonte do Sistema Legado para a linguagem de modelagem MDL tem uma estrutura semelhante à do Transformador Intra-Domínio mostrado na Figura 4.

As especificações na linguagem de modelagem MDL, geradas neste passo, permitem recuperar o projeto orientado a objetos do sistema na ferramenta *Rational Rose*, conforme descrito no próximo passo da estratégia.

4.3. Reprojetar Sistema

No terceiro passo, **Reprojetar Sistema**, o engenheiro de software, usa a ferramenta *Rational Rose* para importar as especificações em MDL e recuperar o Projeto Orientado a Objetos do sistema atual. Com base no projeto recuperado do sistema atual, faz-se o reprojeto do sistema para que este seja distribuído em um ambiente Cliente/Servidor. O reprojeto é realizado seguindo os três níveis do método *Catalysis,* conforme segue:

- **Nível Domínio do Problema**

No Nível Domínio do Problema, o engenheiro de software obtém o Modelo de Tipos, como mostra o exemplo da Figura 6. A Figura 6 mostra, à esquerda, parte da especificação em MDL que deu origem à classe `Vendas` com seus atributos e métodos, à direita. A ferramenta *Rational Rose* importa a descrição MDL e exibe a correspondente representação gráfica e textual.

Após obtido o Nível Domínio do Problema, modela-se o Nível Especificação dos Componentes, reespecificando os requisitos do sistema, orientados às regras do negócio e a funcionalidades do sistema.

```
...
(object Class "Vendas"
quid    "36A8C58C036A"
operations      (list Operations
   (object Operation "Vendas"
        quid    "36A8CC030017"
        documentation   "Método Construtor"
        concurrency     "Sequential"
        pre_condition   (object Semantic_Info
            PDL "teste")
        opExportControl "Public"
        uid     0)
    ...
    (object Operation "AtribuiVendas"
        quid    "36A8CC24037C"
        parameters      (list Parameters
            (object Parameter "variavel"
                type    "boolean"))
        concurrency     "Sequential"
        semantics       (object Semantic_Info
            PDL             )
        opExportControl "Public"
        uid     0))
        class_attributes(list class_attribute_list
          (object ClassAttribute "NroVenda"
             quid       "36A8C81B0394"
             type       "int")
          (object ClassAttribute "Datacompra"
             quid   "36A8C82E0052"
             type       "date")
          (object ClassAttribute "Data"
             quid   "36A8C83A0168"
             type       "date")
...
)

...
```

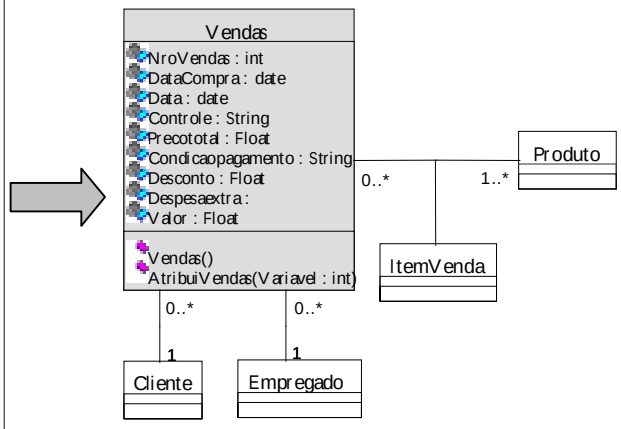

Figura 6 – Recuperação do Projeto do Sistema Legado a partir das especificações em MDL

- **Nível Especificação dos Componentes**

Na Especificação dos Componentes, a principal tarefa é refinar o Modelo de Tipos do nível anterior para retirar ambigüidades e prepará-lo para fazer a distribuição de seus objetos, em uma plataforma Cliente/Servidor. A Figura 7 mostra o Modelo de Tipos do Nível

Especificação dos Componentes, obtido a partir do Modelo de Tipos do nível anterior. Assim, para consistir o modelo foram retirados os atributos `Data`, `Precototal` e `Valor`. `Data` foi removido porque já existia o atributo `DataCompra`. `Precototal` e `Valor` foram removidos porque armazenam valores resultantes do cálculo de outros atributos.

Após a Especificação dos Componentes, passa-se para o Projeto Interno dos Componentes. Este nível preocupa-se com o Projeto Distribuído, que é caracterizado pela partição dos dados e das funcionalidades do sistema em uma arquitetura Cliente/Servidor.

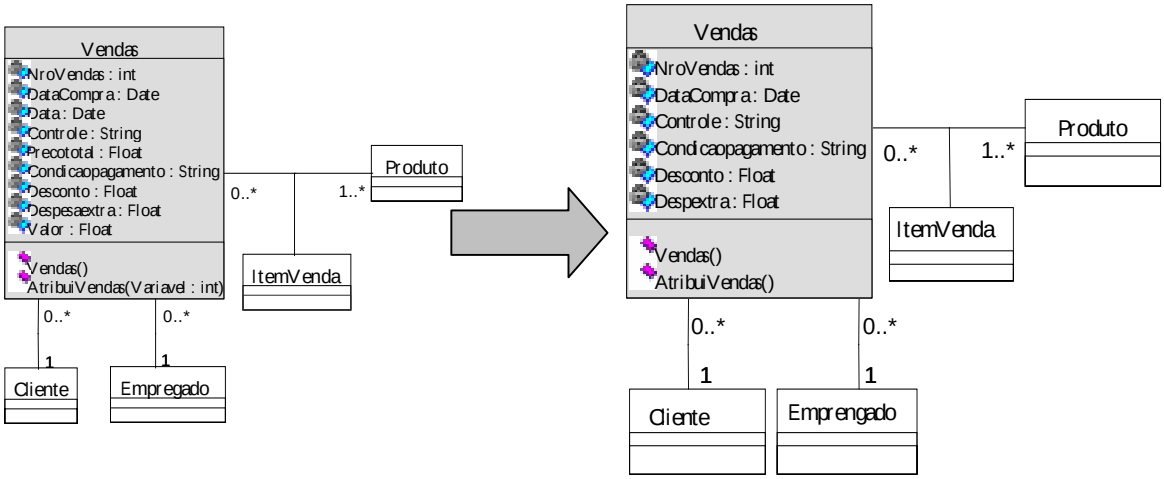

Figura 7 - Reprojeto do Sistema - Modelo de Tipos do Nível Especificação dos Componentes

- **Nível Projeto Interno dos Componentes**

A Figura 8 mostra o Diagrama de Componentes em UML, sendo que o pacote `package.client` representa os objetos da interface gráfica do sistema, localizados em uma plataforma local, utilizada pelos clientes para acessar o sistema. O pacote `package.server` representa os objetos, no servidor remoto central, responsáveis pelas funcionalidades do sistema pelo gerenciamento dos objetos persistentes. Os pacotes `package.API` e `com.cariboulake.util` são responsáveis pela comunicação entre os clientes e o servidor.

A Figura 9 mostra, à direita, o Modelo de Classes do Projeto Interno dos Componentes obtido com o refinamento do Modelo de Tipos do nível anterior, à esquerda. Deste modelo faz-se a transição para a implementação, uma vez que classes são estruturas da construção do código dos sistemas orientados a objetos. Cada classe, cujos objetos devem ser acessados pelo cliente, é dividida em uma interface e uma implementação. A interface, contendo os protótipos dos métodos que o Cliente pode solicitar ao Servidor, é conectada à interface `Remote`, do pacote `java.RMI` para utilizar métodos que tratam do acesso remoto aos objetos dos servidores. A implementação dos métodos, definidos na interface, é feita nas classes de implementação que têm o sufixo *Impl* e conectam-se por herança com a classe `UnicastRemoteObject` do pacote `java.RMI.server`, que gerencia a chamada dos métodos remotos. Um exemplo é a classe `Vendas`, cuja interface é chamada `Vendas` e sua classe de implementação `VendasImpl` conforme mostra a Figura 9.

Ao finalizar este passo, o Projeto Orientado a Objetos Distribuídos é persistido, pela ferramenta *Rational Rose,* em descrições na linguagem de modelagem MDL que serão utilizadas no próximo passo da estratégia.

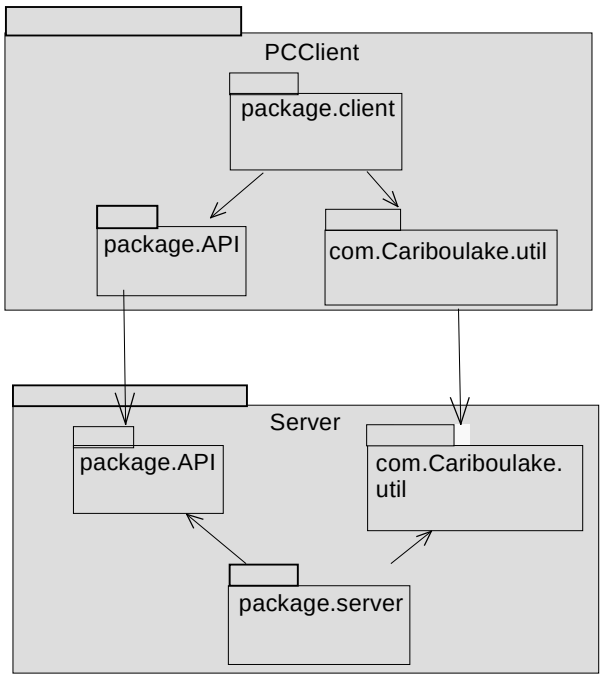

Figura 8 - Reprojeto do Sistema - Diagrama de Componentes do Nível Especificação dos Componentes

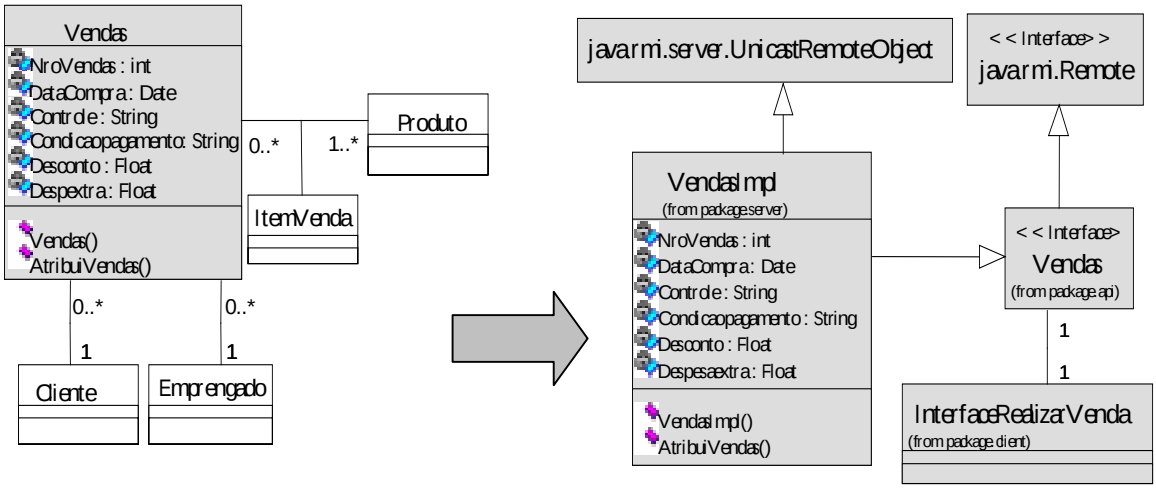

Figura 9 – Reprojeto do Sistema - Modelo de Classes do Nível Projeto Interno dos Componentes

4.4. Reimplementar Sistema

No quarto passo, **Reimplementar Sistema,** são aplicadas as transformações, da linguagem de modelagem MDL para a linguagem orientada a objetos Java/RMI, que transformam as especificações em MDL para componentes em Java/RMI.

A Figura 10 mostra parte da reimplementação, do projeto distribuído, na linguagem orientada a objetos Java/RMI. Por exemplo, a classe descrita na linguagem de modelagem MDL pelo comando `object Class "VendaImpl"`, é transformada para o comando em Java/RMI `public class VendaImpl`.

As miniespecificações em LBE, inseridas através da pré-condição, pós-condição e semântica, nas especificações MDL são transformadas em comandos dos métodos de cada classe.

Na reimplementação o Draco-PUC trata as miniespecificações em LBE como uma nova linguagem embutida na linguagem hospedeira MDL e aplica adequadamente as correspondentes transformações de LBE para Java/RMI. A Figura 10 mostra, na parte sombreada, uma descrição na linguagem de modelagem MDL, que traz embutido comandos LBE indicados pela "|" que são transformados nos comandos Java/RMI.

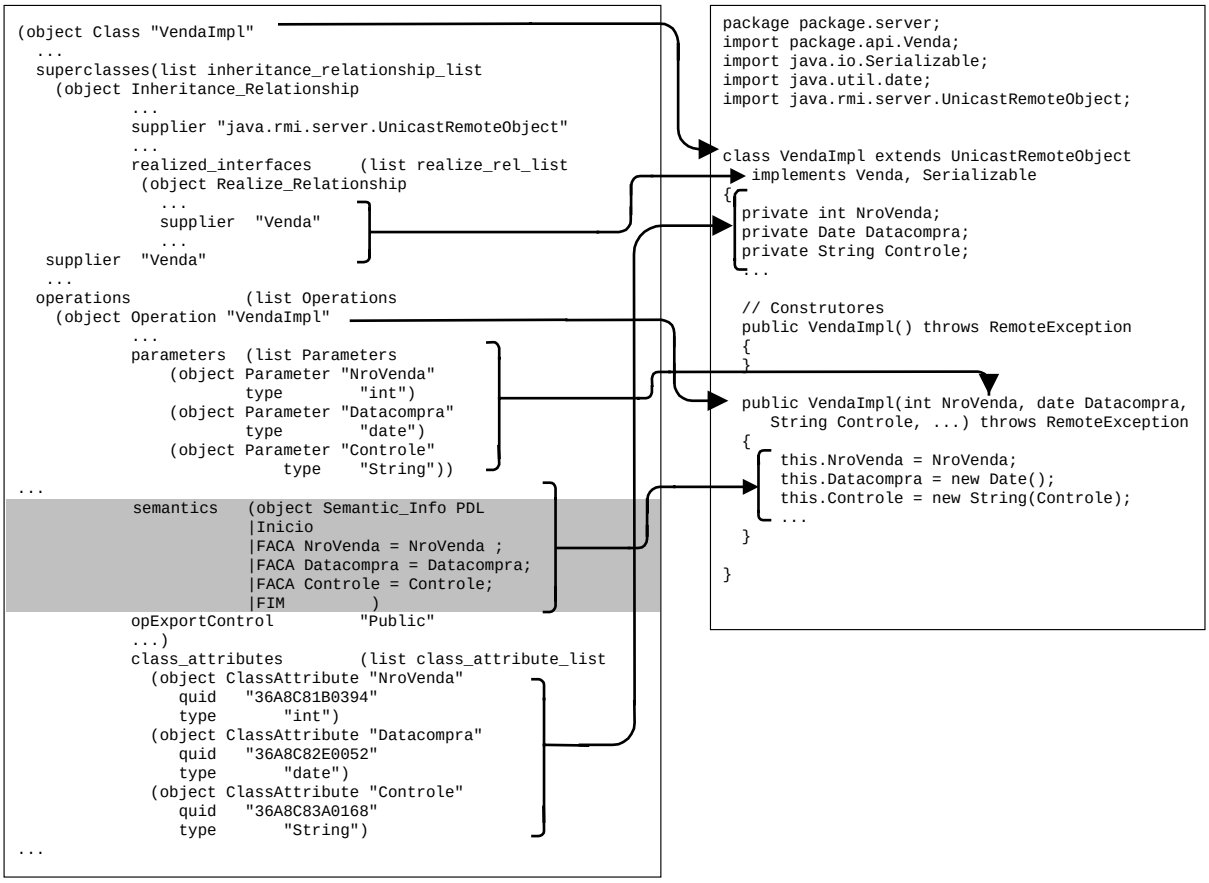

Figura 10 –Reimplementação do projeto Distribuído na Linguagem Orientada a Objetos Java/RMI

Este exemplo ilustra a aplicação das transformações. Entretanto as transformações são escritas no Draco-PUC, com base nas regras gramaticais das linguagens fonte e alvo das transformações. O sistema transformacional Draco-PUC dispõe de uma linguagem própria para facilitar a escrita das transformações, no nível das regras gramaticais. Na reengenharia do código fonte legado em Clipper para a linguagem orientada a objetos Java/RMI, foram escritas 900 transformações que aplicadas no sistema do Estudo de Caso, com aproximadamente 20 mil linhas de código Clipper, geraram em torno de 18 mil linhas do código Java/RMI.

Finalmente, o código gerado na reimplementação é executado por diferentes CPUs em uma plataforma Cliente/Servidor. 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.

4. Conclusão

Este artigo apresentou uma estratégia para transformar código fonte de sistemas legados, orientados a procedimentos, para sistemas orientados a objetos distribuídos. Com técnicas da abordagem Fusion/RE organizou-se o código fonte do sistema legado de acordo com os princípios do paradigma da orientação a objetos, sem perder sua funcionalidade. Após a organização do código fonte do sistema legado, este é transformado para especificações na linguagem de modelagem MDL. A ferramenta para especificação de sistemas orientados a objetos, já consolidada comercialmente, a *Rational Rose*, foi utilizada para recuperar o projeto atual do sistema especificado na linguagem de modelagem MDL e reprojetá-lo fazendo a distribuição dos seus objetos. Esta ferramenta persiste a representação gráfica e textual das técnicas da metodologia UML em descrições textuais na linguagem de modelagem MDL. Partindo destas descrições textuais obteve-se, por transformação, a reimplementação automática em Java/RMI.

Com os resultados obtidos comprovou-se a viabilidade de se transformar sistemas legados em sistemas orientados a objetos distribuídos, através da integração da abordagem Fusion/RE e da ferramenta *Rational Rose* com o sistema transformacional Draco-PUC.

A estratégia pode ser aplicada para outras linguagens fonte e alvo, diferentes do Clipper e Java/RMI, dada a capacidade do sistema transformacional Draco-PUC trabalhar com múltiplos domínios em diferentes níveis de abstração, como no caso das miniespecificações em pseudo-código embutidas na linguagem de modelagem MDL. A granularidade das transformações, no nível das regras gramaticais, garante que a semântica dos comandos da linguagem fonte seja mantida na linguagem alvo da reimplementação.

Esta estratégia de reengenharia, integrando diferentes tecnologias, auxilia os desenvolvedores de software na manutenção do sistema, para ser executado em uma plataforma atual de hardware e software distribuída orientada a objetos, além de atualizar o sistema possibilita a recuperação do projeto e sua modificação para atender a novos requisitos.

Referências

[1] ABRAHÃO, S.M., PRADO, A.F. Web-Enabling Legacy Systems Through Software Transformations. *IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems.* In Proceedings, pp, 149-152. Santa Clara – USA. April, 08-09, 1999.

[2] BRAINSTORMING. URL: http://www.exnet.iastate.edu./Pages/communities/ tools/decisions/brain.htm, 1998.

[3] CHIKOFSKY, J. E., Cross, J. H. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, v.7, n.1, pp. 13-17, 1990.

[4] COLEMAN D. et al. *Object Oriented Development: The Fusion Method.* Prentice Hall, 1994.

[5] D'SOUZA D., WILLS A., *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Addison-Wesley, 1998.

[6] FUKUDA, A. P. *Refinamento Automático de Sistemas Orientados a Objetos Distribuídos,* Qualificação de Mestrado, UFSCar, 1999.

[7] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES J. *Design Patterns: Elements of Reusable Object-Oriented software.* Addison –Wesley, 1995.

[8] JACOBSON, I., LINDSTROM, F. Re-engineering of Old Systems to na Object-Oriented Architecture. *Object-Oriented Programming Systems, Languages, and Applications – OOPSLA'91.* ACM Press. In Proceedings, pp.340-350. 1991.

[9] JACOBSON, I.; ERICSSON, M.; JACOBSON, A. *The Object Advantage - Business Process Reengineering with Object Technology*. Addison Wesley, 1995.

[10] LEITE, J.C.S, PRADO, A.F. Desing Recovery - A Multi-Paradigm Approach. *First International Workshop on Software Reusability.* In proceedings, pp.161-169. Dourtmund, Germany. July, 1991.

[11] LEITE, J.C.S., FREITAS, F.G., SANT'ANNA M. Draco-PUC Machine: A Technology Assembly for Domain Oriented Software Development**.** *3rd International Conference of Software Reuse***.** IEEE Computer Society Press. In proceedings, pp. 94-100. Rio de Janeiro, Rio de Janeiro. 1994.

[12] LIMA, M.A.V. et al. Ambiente "CASE" em Múltiplas Visões de Requisitos e Implementação Automática usando o Sistema Transformacional Draco. *XI Simpósio Brasileiro de Engenharia de Software – SBES'97***.** Fortaleza, Ceará. Anais, pp. 65-80. Outubro, 1997.

[13] 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.

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

[15] PENTEADO, R.D., BRAGA, R., MASIERO, P.C. Improving the quality of Legacy Code by Reverse Engineering. 4th *International Conference on Information Systems Analysis and Synthesis - SCI'98/ISAS98.* In Proceedings, pp.364-370. Orlando, USA. July, 1998.

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

[17] PRADO, A.F., PENTEADO, R.A.D., ABRAHÃO, S.M., FUKUDA, A. P. Reengenharia de Programas Clipper para Java. *XXIV Conferência Latino Americana de Informática - CLEI 98.* Memorias, pg. 383-394. Quito-Equador. 19-23 de Outubro, 1998.

[18] RATIONAL SOFTWARE CORPORATRION, *Rational Rose 98 Rose Extensibility User's Guide*, 1998.

[19] SPENCE, R. *Clipper 5.2.* São Paulo: MAKRON Books, 1994.

[20] Software Reengineering Technique Classification. URL: http://www.baiengineering.com/srtclass.html#771613

[21] FOWLER, M.; SCOTT, K. *UML Distilled - Applying the Standard Object Modeling Language*, Addison-Wesley, 1997.

[22] URL: http://www.rational.com/uml/references

[23] VIDAL, A.G. *Clipper 5.0.* Rio de Janeiro: Livros Técnicos e Científicos Editora, 1991.

[24] WILLS, A. C**.** "*Frameworks", Object Expert* 1(5). URL: http://www.trireme.com.papers/fworks.html, 1996.