Reengenharia Orientada a Objeto de Código Legado *Progress 4GL* para Plataformas Distribuídas

Antonio Francisco do Prado
prado@dc.ufscar.br
Edimilson Ricardo Azevedo Novais
novais@dc.ufscar.br
Universidade Federal de São Carlos
Rodovia Washington Luís (SP-310), Km 235
São Carlos – São Paulo – Brasil – CEP: 13565-905

Resumo

Este artigo apresenta uma estratégia para reconstrução de sistemas escritos na linguagem *Progress 4GL*. O código fonte legado é organizado para facilitar a transformação de um código procedural para orientado a objetos. A parte do código que está organizado segundo as idéias de classes, atributos e protótipos de métodos é transformado para a linguagem de modelagem UML. Os procedimentos do código *Progress 4GL* organizado, são transformados diretamente para Java. As especificações em UML são transformadas para Java e integradas com o código Java dos métodos. Usando uma ferramenta CASE, obtém-se o projeto atual do sistema que pode ser reespecificado gerando um novo projeto orientado a objetos distribuído e atualizado. O sistema reprojetado é novamente transformado para Java, obtendo-se a implementação final do sistema.

Palavras chaves: Reengenharia de Software, Engenharia Reversa, Orientação a Objetos e Sistema de Transformação.

Abstract

This paper presents a strategy for reconstruction systems written in Progress 4GL language. The legacy source code is organized to facilitate the transformation of procedural code into an object-oriented. The part of code that is organized according to the ideas of classes, attributes and methods is transformed into the UML modeling language. The procedures of the Progress 4GL organized code are directly transformed into Java. The UML specifications are transformed into Java and integrated with Java code of the methods. Using a Case tool obtains the current system's project that can be re-designing generating an up to date and distributed object-oriented project. The re-projected system is again transformed to Java, getting the final implementation of system.

***Key Words**: Software Re-engineering, Reverse Engineering, Object Oriented and Transformation System.*

1. Introdução

Uma das áreas da Engenharia de Software que vem se destacando é a Reengenharia de código legado. O objetivo da reengenharia de software é manter o conhecimento adquirido com os sistemas legados e utilizar estes conhecimentos como base para a evolução contínua e estruturada do software. O código legado possui uma lógica de programação, decisões de projeto, requisitos do usuário e regras de negócio que podem ser recuperados e reconstruídos sem perda da semântica. O software é reconstruído com inovações tecnológicas e novos requisitos podem ser adicionados para atender prazos, custos, correções de erros e melhorias de desempenho.

A indústria de software sente necessidade de aperfeiçoar o ciclo de vida de seus sistemas. O processo de desenvolvimento e operacionalização de um produto passa por 3 fases básicas: criação, estabilização e manutenção. Na fase de criação o sistema é analisado, projetado e implementado; na fase de estabilização podem ser incluídas novas funcionalidades para atender os requisitos não funcionais do sistema e finalmente na fase de manutenção o sistema pode ser alterado para corrigir falhas, atender novos requisitos ou melhorar seu desempenho. Nesta última fase, muitas vezes o sistema fica comprometido por manutenções estruturais não previstas no projeto original.

Um sistema comercial, como é o caso da maioria dos sistemas implementados em *Progress 4GL*, leva de um a dois anos para se estabilizar e atender o mínimo dos requisitos necessários. É grande a quantidade de sistemas implementados, inclusive em versões antigas da linguagem *Progress 4GL*, com interface caracter, cuja reengenharia é importante para atualizá-los para novas plataformas de hardware e software. A Linguagem *Progress 4GL* é uma linguagem de 4ª geração orientada a procedimentos, desenvolvida pela *Progress Software Corporation* (PSC), intimamente ligada ao Banco de Dados *Progress*, com grande uso em aplicações no mercado mundial. Possui os comandos básicos de uma linguagem de consulta a banco de dados com transações atômicas e *triggers* de eventos. Por ser orientada a procedimentos e pela grande quantidade de sistemas implementados, que são utilizados por médias e grandes empresas, a linguagem *Progress 4GL* foi escolhida, como linguagem do código fonte legado, para validar a estratégia de reengenharia proposta.

Motivados por estas idéias pesquisou-se uma estratégia de reengenharia cujo objetivo é reconstruir sistemas legados para serem executados em novas plataformas de hardware e software distribuídas. Partindo do código legado *Progress 4GL*, o desenvolvedor pode organizar o código segundo os princípios da Orientação a Objetos e obter o projeto atual do sistema em UML e Java. Em seguida pode-se reprojetar o sistema para atender novos requisitos e distribuir seus componentes. E finalmente pode-se reimplementar o sistema numa linguagem orientada a objetos como Java.

O artigo está organizado da seguinte forma: a seção 2 apresenta as principais técnicas usadas na estratégia de reengenharia, a seção 3 descreve a estratégia de Reengenharia Orientada a Objetos de Código Legado *Progress 4GL* para Plataformas Distribuídas, a seção 4 apresenta um estudo de caso como exemplo de uso da estratégia e finalmente, a seção 5 apresenta as conclusões desta pesquisa.

2. Principais Técnicas da Reengenharia de Software

Uma das principais técnicas usadas na Reengenharia de Software é de reconstrução do software, usando transformações. Diferentes Sistemas de Transformação têm sido usados destacando-se o Tampr [Boy89], Refine [Rea92], Popart [Wid93], TXL [Cor93], DMS [Bax97] e RescueWare [Faq98]. Outro importante sistema de transformação que vem sendo utilizado nesta área é o Draco-PUC, como pode ser constatado em [Lei91, Lei94, Pra92, San93, Pra98, Abr99].

O Draco-PUC [Nei84, Lei91, Pra92, Lei94] implementa as idéias de transformação de software orientada a domínios. Pela estratégia proposta por Prado [Pra92], 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* responsável por analisar os programas da linguagem e gerar a representação interna do Draco-PUC, uma *Abstract Syntax Tree (AST),* denominada no Draco-PUC de DAST; um ***prettyprinter*** ou *unparser,* que faz 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 os 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.

A técnica de engenharia reversa Fusion/RE [Pen96] é utilizada para obter o entendimento e revitalizar a estrutura do código legado segundo as idéias do paradigma de orientação a objetos, visando reutilizar toda a funcionalidade do código legado na reconstrução do sistema. A estratégia apresentada neste artigo usa apenas parte da abordagem Fusion/RE para recuperar o modelo de objetos do Modelo de Análise do Sistema Atual (MASA), com suas classes, elementos de dados, procedimentos e relacionamentos, que originalmente não estão orientado a objetos, e a partir daí, elaborar o modelo de objetos do Modelo de Análise do Sistema (MAS), corrigindo as anomalias dos procedimentos.

A tecnologia *Enterprise JavaBeans* (EJB) [Val99], é usada para o desenvolvimento de aplicações *multi-tier*. Esta tecnologia de implementação especifica modelos de componentes orientados a transação e baseado em servidores escritos em Java [Sun97]. Com a tecnologia EJB, os componentes que implementam as regras lógicas do negócio são disponibilizados no servidor EJB. O servidor EJB tem um sistema próprio de execução de componentes, que manipula as complexidades do sistema, como: *threads*, transações, gerenciamento de estado e recursos compartilhados [Fuk99].

Além destas técnicas, vem ganhando destaque o uso de ferramentas CASE no projeto ou reprojeto de sistemas a serem reconstruídos. Uma das ferramentas CASE conhecida é a *Rational Rose* [Rat98] que suporta a especificação do sistema em UML [UML97, UML98] e a geração parcial de código em uma linguagem executável.

Combinando estas diferentes técnicas de Engenharia Reversa Fusion/RE, o sistema de transformação Draco-PUC, a implementação de sistemas distribuídos com *Enterprise JavaBeans* (EJB) e a ferramenta CASE *Rational Rose*, definiu-se uma estratégia para a Reengenharia Orientada a Objetos de Código Legado *Progress 4GL* para Plataformas Distribuídas, que será apresentada a seguir.

3. Reengenharia Orientada a Objetos de Código Legado *Progress 4GL* para Plataformas Distribuídas

A estratégia de Reengenharia Orientada a Objetos de Código Legado *Progress 4GL* é realizada em três passos: **Organizar, Reimplementar e Reprojetar Sistema**, como mostra a Figura 1 [Pra92].

A seguir são apresentados dos passos desta estratégia de Reengenharia.

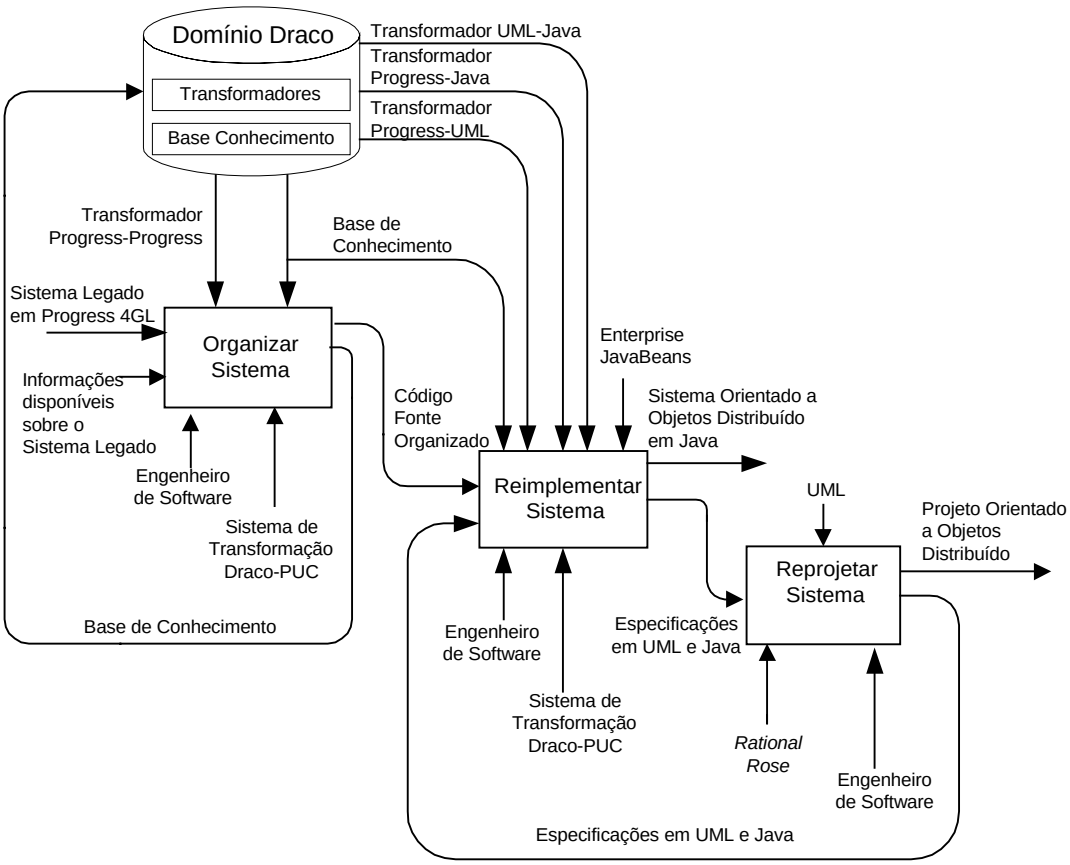

Figura 1 – Estratégia de Reengenharia Orientada a Objetos de Código Legado *Progress 4GL para Plataformas Distribuídas*

3.1 Organizar Sistema

Neste passo faz-se a organização do código legado *Progress 4GL* 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.

Para suportar este passo foi construído o domínio *Progress* no sistema de transformação Draco-PUC. Baseado na gramática livre de contexto do *Progress 4GL* gerou-se o *parser* e o *prettyprinter*. O p*arser* foi gerado a partir da definição das regras da gramática *Progress 4GL*, utilizando o gerador *Pargen* do Draco-PUC. O *prettyprinter* foi gerado automaticamente pelo subsistema *Ppgen* do Draco-PUC, a partir das definições da gramática. A figura 2 mostra parte da gramática *Progress 4GL*.

```
foreach_stat : 'FOR EACH' .sp iden .sp (expr .sp)? lock_opt?  ':'
               statements* end_proc                                ;
find_stat    :  'FIND' .sp iden .sp select_opt?
               lock_opt? (.sp 'NO-ERROR')?  end_proc                ;
display_stat : 'DISPLAY' (preprocessor | object_set) .sp frame_opt?
               end_proc                                            ;
create_stat   : 'CREATE' .sp iden end_proc                                  ;
atrib_stat    : expr end_proc                                     ;
assign_stat   : 'ASSIGN' (preprocessor | (.sp expr)+) frame_opt?
...
```

Figura 2 – Parte da gramática *Progress 4GL*

As transformações são escritas no Draco-PUC, com base nas regras gramaticais das linguagens fonte e alvo das transformações. O sistema de transformação Draco-PUC dispõe de uma linguagem própria para facilitar a escrita das transformações.

Para auxiliar o processo das transformações que necessitam concentrar ou espalhar comandos ao longo do código fonte organizado, utilizou-se uma Base de Conhecimento (*Knowledge Base*), que permite armazenar fatos e regras para consultas posteriores. Este é um recurso disponível no Draco-PUC, utilizado quando se faz necessário adiar decisões durante a aplicação das transformações. Os principais comandos para atualização da Base de Conhecimento são: *KBSolve*, que busca por um fato; *KBRetrieve*, que recupera o fato; *KBDelete*, que elimina o fato; *KBAssertifNew*, que armazena o fato; e *KBWrite*, que grava as modificações. Neste passo, a Base de Conhecimento armazena fatos relacionados com as supostas classes, atributos, métodos e relacionamentos. Estes fatos são extraídos da estrutura do banco de dados *Progress* por um programa auxiliar escrito em *Progress 4GL*.

O transformador intra-domínio, que organiza o código legado, utiliza nos pontos de controle de reconhecimento (LHS) e de substituição (RHS), os padrões sintáticos da mesma linguagem, no caso *Progress 4GL*. Os demais pontos de controle são utilizados para armazenar e recuperar fatos da base de conhecimento que auxiliam na solução de problemas de organização do código legado.

Em resumo, a biblioteca *ProgressToProgress* do domínio *Progress 4GL*, construída no Draco-PUC para organizar o código legado, contêm transformações para:

- Agrupar as variáveis globais;
- Obter o comportamento dos procedimentos relacionados com a interface;
- Reutilizar os procedimentos comuns em várias partes do código fonte;
- Separar o código fonte em procedimentos atômicos, associados a uma única classe candidata, respeitando o escopo, dependência de dados e visibilidade do código legado;
- Reunir em um único programa fonte, todos os procedimentos candidatos a métodos de uma suposta classe; e
- Armazenar na Base de Conhecimento os supostos atributos e relacionamentos, para serem consultados no próximo passo da estratégia.

Para organizar o código legado foram definidos padrões de reconhecimento do código *Progress 4GL* que são transformados em novos comandos, na mesma linguagem *Progress 4GL*, porém com as características da orientação a objetos. Destacam-se os seguintes padrões de reconhecimento:

- Suposta Classe: usado no reconhecimento dos comandos de criação de registros no banco de dados, comparando o nome da tabela do banco de dados com as supostas classes armazenadas na base de conhecimento, criada no início da organização a partir da estrutura das tabelas do banco de dados do sistema legado. As interfaces do usuário e seus procedimentos para validação de informações são reconhecidos como supostas classes de projeto;
- Suposto Atributo: usado no reconhecimento dos campos declarados nos comandos de acesso ao banco de dados, comparando-os com os supostos atributos armazenados na base de conhecimento, criada a partir da estrutura das tabelas do banco de dados do sistema legado;
- Suposto Método: usado no reconhecimento dos comandos que armazenam e recuperam informações de uma tabela do banco de dados, candidata a classe. Neste caso deve-se observar a visibilidade, o escopo e a dependência de dados. Na análise dos procedimentos que atualizam ou recuperam informações de mais de uma tabela do banco de dados, é eleita a tabela que tem mais dados alterados ou recuperados como possível classe do correspondente método deste procedimento. Os disparadores de eventos do banco de dados, "*triggers*", armazenados como fatos na base de conhecimento, são transformados em procedimentos candidatos a métodos

para criação, gravação e exclusão de objetos da classe. Os comandos de recuperação de dados FOR *EACH e FIND*, são encapsulados em um único procedimento de leitura para cada tabela, recebendo como parâmetros as informações usadas no seu comportamento. O comando para criação de um registro na base de dados *CREATE*, gera um procedimento candidato a método construtor de uma suposta classe. Os comandos de atualização de dados *ASSIGN*, espalhados pelo código legado, são encapsulados dentro de um único procedimento de atualização de dados para cada tabela do banco de dados, candidata a uma classe.

- Suposto Relacionamento: usado no reconhecimento dos relacionamentos entre as supostas classes. Baseia-se nos relacionamentos entre as tabelas do banco de dados. Uma tabela se relaciona com outra quando os atributos que formam um índice único na tabela, também são atributos em outra tabela. Para tabelas que são relacionadas, mas não possuem os mesmos atributos, é necessário analisar os comandos de leitura de informações no banco de dados para reconhecer o relacionamento.

A organização do código fonte legado, segundo os princípios da Orientação a Objetos, facilita a transformação para UML e Java, que é o objetivo do segundo passo da estratégia.

3.2 Reimplementar Sistema

Neste passo o engenheiro de software também utiliza o sistema de transformação Draco-PUC para gerar automaticamente as especificações nas linguagens UML e Java.

Para suportar este passo foram construídas as bibliotecas de transformação *ProgressToUML* e *ProgressToJava* do domínio *Progress 4GL*. Estas bibliotecas contêm as transformações que reconhecem padrões sintáticos e semânticos do código fonte legado e substituem pelo seus correspondentes padrões sintáticos e semânticos, definidos na linguagem alvo da transformação, no caso UML e Java.

Parte do código *Progress 4GL* organizado segundo classes, com seus atributos e protótipos de métodos, é transformado para especificações na linguagem de modelagem UML. Para persistência das especificações UML usa-se a mesma linguagem da ferramenta CASE *Rational Rose*, que armazena as especificações em arquivos com extensão ".MDL".

Em UML as miniespecificações dos métodos das classes podem ser descritas através de pré e pós-condições e semântica. Dado o conhecimento que grande parte dos desenvolvedores têm sobre linguagens de programação, decidiu-se pelo "porte" direto do código legado dos corpos dos procedimentos *Progress 4GL* para a linguagem Java. Assim, o transformador *ProgressToJava*, foi construído para transformar os códigos dos procedimentos *Progress4GL* em código Java. O código Java gerado fica embutido na UML, na especificação da semântica de cada método de uma classe. A integração das linguagens UML e Java foi possível porque o Draco-PUC suporta descrições em múltiplas linguagens.

A figura 3 apresenta um exemplo de transformação da biblioteca *ProgressToUML*. Do lado esquerdo, tem-se em destaque o padrão de reconhecimento *Procedure*, que define um procedimento na linguagem origem *Progress 4GL*. Do lado direito, tem-se o correspondente padrão de substituição *Operations*, que define um método na linguagem alvo UML.

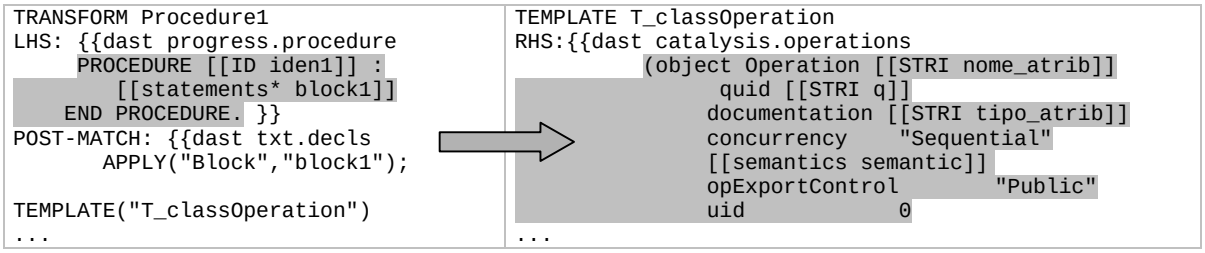

Figura 3 –Transformação Inter-Domínio *ProgressToUML*

Este passo usa também um Transformador Inter-Domínios já existente no sistema de transformação Draco-PUC, objeto de outra pesquisa [Fuk99]. O sistema de transformação Draco-PUC é utilizado pelo engenheiro de software para transformar automaticamente para Java, os Diagramas de Classes do sistema projetado, com seus atributos e protótipos de métodos especificados em UML. O código Java gerado neste passo é integrado com o código Java dos métodos, inseridos na parte semântica das especificações UML, obtendo-se a implementação final do sistema, conforme apresentado em [Fuk99].

Como os programas, gerados automaticamente em Java pelo transformador, estão baseados na estrutura Java/EJB, os objetos podem ser chamados em outras máquinas virtuais Java de forma remota, permitindo a execução do sistema distribuído em múltiplas plataformas. O pacote RMI, para chamada de métodos remotos na arquitetura Cliente/Servidor, foi utilizado no EJB para tornar mais simples e transparente a distribuição de objetos em um programa Java. Um Cliente RMI interage com objetos no Servidor através da Interface Remota do objeto servidor. Utiliza-se um *stub* para chamar um método da classe remota. Este *stub* é automaticamente gerado pelo EJB [Fuk99].

Com o objetivo de facilitar o processo de transformação, principalmente das classes que tratam funções de entrada e saída do *Progress 4GL*, foram criadas classes padrões em Java cujos métodos expressam a mesma semântica dos comandos *Progress 4GL*. Estas classes ficam em um pacote denominado *Progress* e usam comandos SQL para acessar um banco de dados conectado pelo protocolo JDBC/OBDC. Trata-se de classes abstratas que não são instanciadas, servindo apenas para reuso dos seus atributos e métodos através do princípio da herança.

3.3 Reprojetar Sistema

No terceiro passo da estratégia de reengenharia, Reprojetar Sistema**,** o Engenheiro de Software, usando a ferramenta CASE *Rational Rose*, importa as especificações na linguagem UML para obter o projeto do sistema atual. Nas especificações UML, a semântica dos métodos em cada classe está descrita diretamente em Java. Dessa forma o código Java que implementa os corpos dos métodos fica embutido na especificação UML. Com o projeto do sistema recuperado em UML, é possível compreendê-lo e reprojetá-lo. Tanto o projeto, como a implementação dos métodos, estão disponíveis na ferramenta CASE para o reprojeto, possibilitando a alteração do sistema para atender novas especificações de requisitos e fazer a distribuição do seus componentes em um ambiente Cliente/Servidor, caso seja necessário.

As especificações UML do sistema reprojetado, podem ser novamente transformadas para Java, obtendo-se a implementação final do sistema orientado a objetos distribuído.

4. Estudo de Caso

A estratégia proposta foi aplicada em diferentes estudos de caso, destacando-se um sistema exemplo que acompanha o ambiente de desenvolvimento *Progress*. Este sistema implementa o processo de venda de uma loja de materiais esportivos e permite a consulta das informações financeiras geradas pela venda.

Segue-se uma apresentação de cada passo da estratégia usada na reengenharia deste sistema.

4.1 Organizar Sistema

A aplicação da estratégia tem início com o código fonte *Progress 4GL* do sistema legado. Neste passo faz-se a organização do código legado segundo os princípios da orientação a objetos, utilizando o transformador intra-domínio *ProgressToProgress*, no sistema de transformação Draco-PUC.

A figura 4 mostra, à esquerda acima, um trecho do código legado *Progress 4GL*, à esquerda abaixo, o correspondente código organizado segundo os princípios da orientação a objetos e à direita, os procedimentos da tabela *CUSTOMER*, candidatos a métodos, criados durante a organização. O código é organizado para executar chamadas aos procedimentos armazenados em *CUSTOMER.P*.

Outras transformações são, da mesma forma, aplicadas para organizar o código fonte segundo as idéias do paradigma da orientação a objetos. Este código organizado é usado no próximo passo da estratégia.

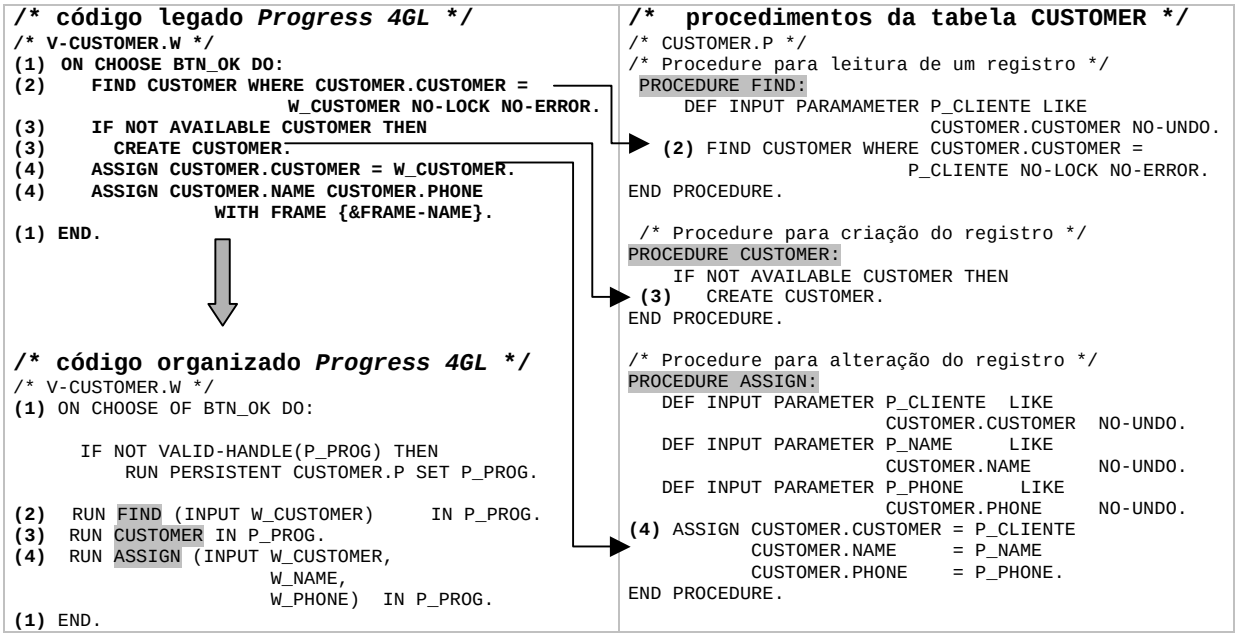

Figura 4 – Organização do código legado *Progress 4GL*

4.2 Reimplementar Sistema

Neste passo, utilizam-se os Transformadores Inter-Domínios *ProgressToUML, ProgressToJava* e *CatalysisToJava* [Fuk99a]. O código legado, organizado em blocos de comandos e declarações, que atuam sobre objetos comuns a determinado tipo, é transformado para descrições em UML. Classes encapsulando as declarações e os protótipos dos métodos são geradas para cada bloco de

código organizado. Os códigos de cada procedimento *Progress 4GL* são transformados diretamente em Java. Dessa forma, as descrições UML das classes geradas têm embutidas na parte semântica dos métodos os respectivos códigos Java.

Para melhor entendimento deste passo apresenta-se na figura 5 um exemplo de transformação, onde à esquerda tem-se o código legado organizado e à direita a correspondente descrição UML. Pode-se ver ainda, em destaque, o trecho de código organizado que foi transformado diretamente para Java. Este código fica embutido na parte semântica do método *BTN_OK_Clicked*, cujo protótipo está declarado na classe *V-CUSTOMER*.

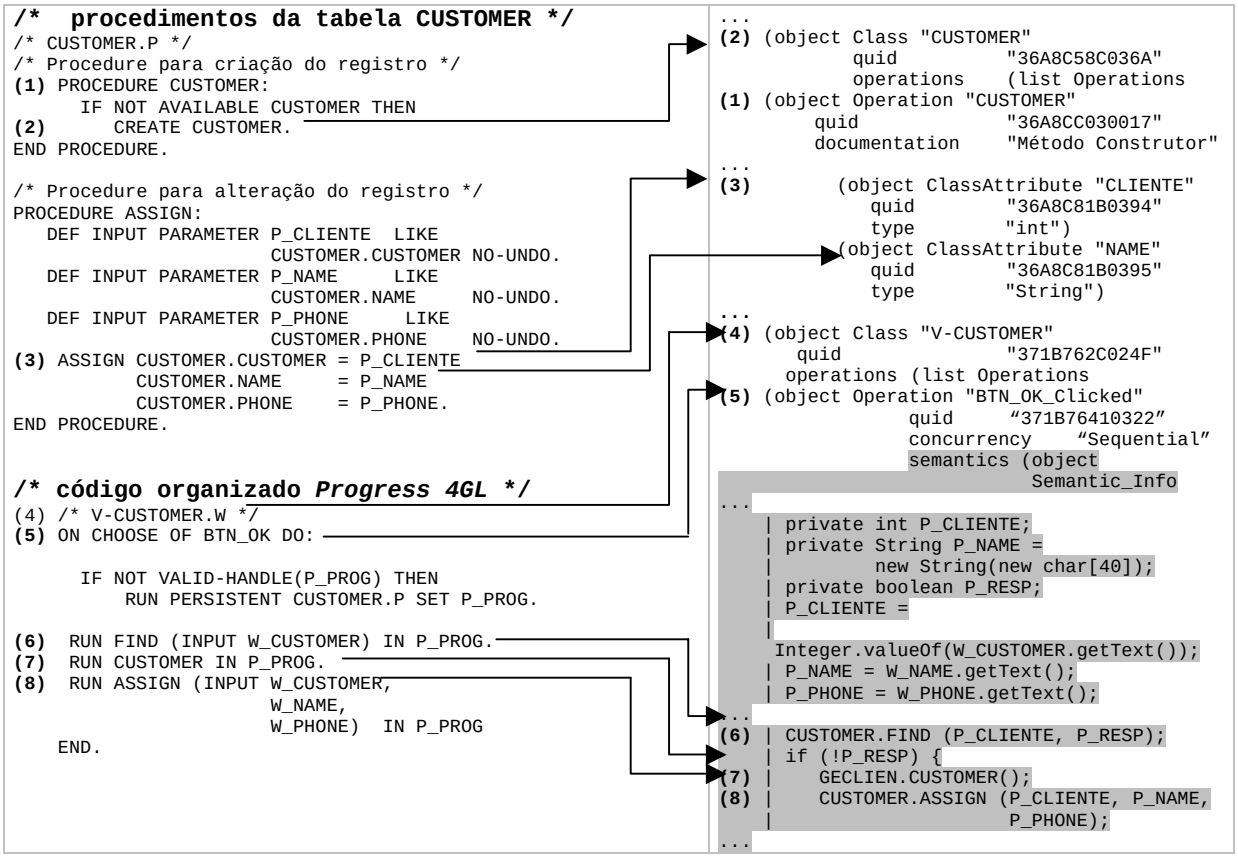

Figura 5 – Transformação do código organizado para especificações UML

Os Diagramas de classes do sistema são reimplementados em Java pelo transformador *CatalysisToJava* do domínio *Catalysis*. Na reimplementação, o sistema de transformação Draco-PUC, trata os métodos das classes, cujos comportamentos já estão especificados em Java, como uma linguagem embutida na linguagem UML. O código Java gerado a partir das especificações UML é integrado com o código Java dos métodos para obter a implementação sistema.

As especificações na linguagem de modelagem UML, geradas neste passo, permitem recuperar o projeto orientado a objetos do sistema na ferramenta CASE *Rational Rose*, conforme apresenta o próximo passo da estratégia.

4.3 Reprojetar Sistema

Neste passo, o engenheiro de software usa a ferramenta *Rational Rose* para importar as

especificações em UML e recuperar o Projeto Orientado a Objetos do sistema atual. A figura 6 mostra à esquerda parte o projeto recuperado na ferramenta CASE *Rational Rose* e a direita os correspondentes códigos Java dos métodos em cada classe do projeto recuperado.

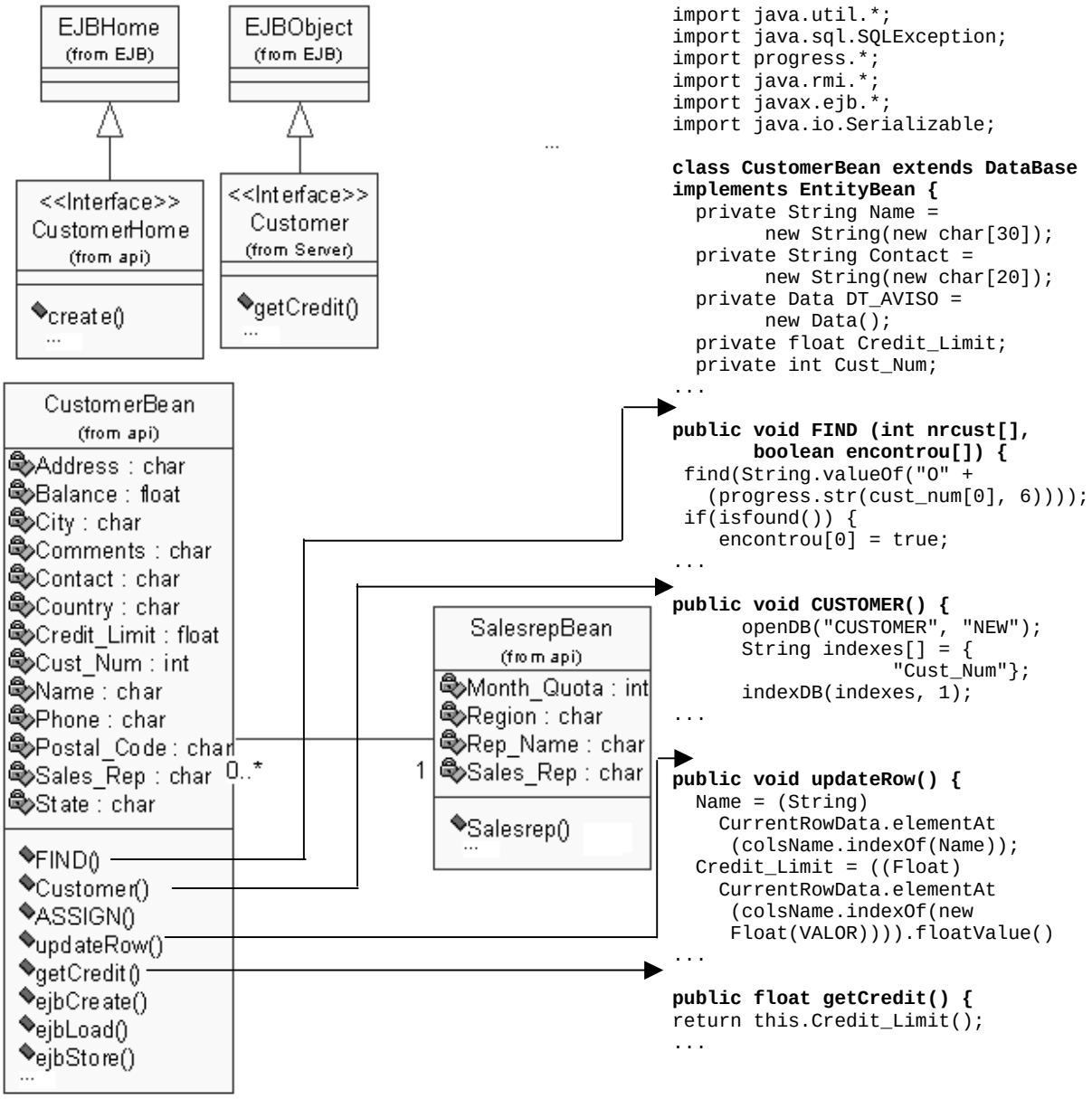

Figura 6 – Projeto orientado a objetos recuperado pela ferramenta CASE e código Java dos métodos

Com o projeto do sistema recuperado em UML, torna-se possível compreendê-lo e reprojetá-lo para atender novos requisitos e distribuir seus componentes em uma arquitetura Cliente/Servidor. Pode-se também editar e alterar a semântica dos métodos, ”portados” diretamente para Java.

O sistema reprojetado é novamente reimplementado pelo passo anterior, **Reimplementar Sistema**, obtendo-se a implementação final do sistema orientado a objetos distribuído. O código gerado pode ser executado por diferentes CPUs e 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 sistema.

Uma vez que 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. Conclusão

Este artigo apresentou uma estratégia para reengenharia de sistemas legados orientados a procedimentos, escritos em *Progress 4GL*, para sistemas orientados a objetos distribuídos, sem perda da funcionalidade e com a possibilidade de reprojeto do sistema e distribuição dos seus componentes. A estratégia objetiva reconstruir sistemas legados escritos em linguagem procedural, para serem executados em plataformas mais modernas de hardware e software.

As seguintes idéias comprovam a originalidade desta estratégia:

- Sistemas legados descritos em *Progress 4GL* são reimplementados, segundo os princípios da orientação a objetos, em uma linguagem de 4ª Geração, incluindo comandos específicos para consulta e manutenção de informações em um banco de dados relacional;
- O Domínio *Progress 4GL*, com 3 bibliotecas *ProgressToProgress*, *ProgressToUML* e *ProgressToJava*, foi construído no Sistema de Transformação Draco-PUC;
- Uma biblioteca com classes que tratam as entradas e saídas do Banco de Dados *Progress* em Java foi construída para facilitar o processo de transformação;
- Diferentes técnicas, destacando-se o método Fusion/RE, uma ferramenta CASE, *Enterprise JavaBeans* (EJB) e um Sistema de Transformação foram integrados para suportar a Reengenharia;
- Uma estratégia testada em vários sistemas legados que cobre todo o ciclo da reengenharia de um software não orientado a objetos: recuperação do Modelo de Análise Atual (MASA), elaboração do Modelo de Análise do Sistema (MAS), reprojeto do sistema em uma ferramenta CASE e reimplementação automática numa linguagem orientada a objetos.

Dada a capacidade do Sistema de Transformação Draco-PUC de trabalhar com uma rede de domínios com diferentes linguagens, pode-se aplicar a estratégia de reengenharia em sistemas legados escritos em outras linguagens, diferentes de *Progress 4GL*. Também as linguagens alvo da reimplementação podem ser diferentes de UML e Java.

Outra contribuição desta pesquisa é que a estratégia torna sistemático o uso de Sistemas de Transformação na Reengenharia de Software.

Bibliografia

Abr99] 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.

[Bax97] BAXTER I., PIDGEON, C.W. Software Change Through Design Maintenance. **International Conference on Software Maintenance – ICSM'97.** In Proceedings. Bari, Italy. October 1st –3rd, 1997.

[Boy89] BOYLE, J. **Abstract Programming and Program Transformation – An**

Approach to Reusing Programs, in Software Reusability. Vol.1, pp. 361-413. Ed Ted Biggerstaff. ACM Press, 1989.

[Cor93] CORDY, J., CARMICHAEL, I., **The TXL Programming Language Syntax and Informal Semantics.** Technical Report. Vol.7.Queen's University at Kingston – Canada. June, 1993. (or http://www.queis.queensu.ca/STLab/TXL)

[Faq98] FAQs – RescueWare. http://www.relativity.com/products/faqs/index.html

[Fuk99] Fukuda, Ana Paula; Jesus, Elisangela S.; Prado, Antonio F. *Refinamento Automático de Sistemas Legados para Sistemas Orientados a Objetos Distrinuídos.* XXV Centro Latino-americano de Estudos em Informática (CLEI'99), Assunção, Paraguai, Agosto de 1999.

[Lei91] 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.

[Lei94] 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, 1994.

[Nei84] 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.

[Pen96] PENTEADO, R.D. *Um Método para Engenharia Reversa Orientada a Objetos.* São Carlos, 1996. Tese de Doutorado. Universidade de São Paulo. 251p.

[Pra92] PRADO, A.F. **Estratégia de Engenharia de Software Orientada a Domínios.** Rio de Janeiro, 1992. Tese de Doutorado. Pontífica Universidade Católica. 333p.

[Pra98] 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.

[Rat98] RATIONAL SOFTWARE CORPORATRION, *Rational Rose 98 Rose Extensibility User's Guide*, 1998.

[Rea92] REASONING SYSTEMS INCORPORATED. **Refine User's Guide, Reasoning Systems Incorporated.** Palo Alto, 1992.

[San93] SANT'ANNA, M. Lavoisier: **Uma Abordagem Prática do Paradigma Transformacional.** Monografia de Graduação. Rio de Janeiro. PUC-Rio - Pontífica Universidade Católica do Rio de Janeiro. 1993. 100p.

[Sun97] Sun Microsystems. *Tutoriais Java*, URL: http://www.javasun.com, 1997.

[Uml97] FOWLER, M., SCOTT, K. **UML Distilled – Applying the Standard Object Modeling Language.** Addison-Wesley, 1997.

[Uml98] URL: http://www.rational.com/uml/references, 1998.

[Val99] Valeskey, T., *Enterprise JavaBeans, - Developing Component-Based Distributed Applications*. ADDISON-WESLEY, 1999.

[Wid93] WILE D. **POPART: Producer of Papers and Related Tools System Builders Manual.** Technical Report. USC/Information Sciences Institute. November, 1993.