Reengenharia Orientada a Objetos de Sistemas COBOL para Ambiente Web Thin-Client

Valter Vieira de Camargo* (1,2,3)
valter@icmc.usp.br

Rosângela Ap. D. Penteado (1)
rosangel@dc.ufscar.br

[1] Universidade Federal de São Carlos UFSCar Departamento de Computação Cep 13565-905 – São Carlos – SP- Brasil	[2] Fundação Educacional de Fernandópolis – FEF Fernandópolis – SP – Brasil	[3] Universidade de São Paulo – USP Instituto de Ciências Matemáticas e de Computação (ICMC) São Carlos – SP - Brasil

Abstract. *This paper shows an object-oriented reengineering process of procedimental legacy systems for Web-based environments. So, the architecture of these systems must be restructured for the client/server model. It can be made separating the application logic and database components, which are executed in the server, from user interface component, which is executed in the client. A middleware must be defined which is responsible by communication between these components. The reengineering proposed here uses the Fusion/RE method for the reverse engineering process. This method was specialized for COBOL language domain and guidelines are presented to generate object oriented analysis model specified in UML. HTML language is used for the implementation of new system's interfaces and, for the middleware is used an API of Java language, the servlets. Persistence Layer design pattern is used to separate the application logic component from database component.*

Key Words: Reengineering, Web, Internet, Reverse Engineering, COBOL, Fusion/RE, Object Orientation, UML.

Resumo. *Este artigo apresenta um processo de reengenharia orientada a objetos de sistemas legados procedimentais para ambientes baseados na Web. A arquitetura de tais sistemas deve ser reestruturada para o modelo cliente/servidor. Isso pode ser feito separando os componentes lógica da aplicação e banco de dados, que são executados no servidor, do componente interface do usuário, que é executado no cliente. Também deve haver a definição de um middleware, responsável pela comunicação entre os componentes citados. A reengenharia realizada utiliza o método Fusion/RE para a realização da engenharia reversa. Esse método foi especializado para o domínio da linguagem COBOL e, diretrizes para sua aplicação, gerando modelos de análise orientados a objetos, especificados em UML, são apresentadas. A linguagem de marcação HTML é utilizada para a implementação das novas interfaces do sistema e, para middleware, é utilizada uma API da linguagem Java, os servlets. O padrão de projeto Persistence Layer também é utilizado para separar o componente lógica da aplicação do componente banco de dados.*

Palavras chave: Reengenharia, Web, Internet, Engenharia Reversa, COBOL, Fusion/RE, Orientação a Objetos, UML.

* Trabalho realizado com o apoio financeiro da CAPES

1 - INTRODUÇÃO

A reengenharia de sistemas legados para ambientes cliente/servidor necessita que tais sistemas sejam separados em três componentes: a) interface do usuário; b) lógica da aplicação; e c) banco de dados. O componente interface do usuário contém o código responsável pela implementação das interfaces com o usuário. Já, o componente lógica da aplicação contém o código que implementa as regras de negócio e, o componente banco de dados contém o código que implementa as manipulações com o banco de dados. Dependendo do estilo de arquitetura cliente/servidor escolhido, *fat-client* ou *thin-client*, o componente lógica da aplicação pode ser colocado no cliente, no servidor ou pode haver balanceamento entre ambos. Neste trabalho, esse componente é colocado no servidor, pois se busca uma arquitetura com estilo *thin-client*, em que a maior parte do processamento ocorre no servidor.

O processo de reengenharia descrito utiliza: a) o método de engenharia reversa Fusion/RE para a extração do componente lógica da aplicação, b) o padrão de projeto *Persistence Layer* para separar o componente banco de dados do componente lógica de aplicação, c) *servlets* da linguagem Java para tratar as requisições/respostas dos clientes ligados à rede. O objetivo é fornecer diretrizes que auxiliem na reengenharia orientada a objetos de sistemas legados implementados em COBOL para ambientes baseados na Web, por exemplo: *Internet*, *Intranet* e *Extranet*. O sistema alvo tem arquitetura cliente/servidor estilo *thin-client*, é implementado em Java e utiliza banco de dados relacional *SyBase*.

Na Seção 2 são apresentados os trabalhos relacionados e, na Seção 3, o processo completo de reengenharia é tratado. Na Seção 4 é apresentado o estudo de caso realizado e, a Seção 5, trata das considerações finais.

2 - TRABALHOS RELACIONADOS

Os trabalhos relacionados são divididos em duas seções, sendo que a primeira, Seção 2.1, trata de engenharia reversa/reengenharia, porém, sem considerar aspectos de ambientes Web. Já, a segunda, Seção 2.2, trata especificamente de reengenharia/migração de sistemas para ambientes Web.

2.1 – Engenharia Reversa e Reengenharia

Uma abordagem de reengenharia orientada a objetos para sistemas implementados em COBOL é proposta por Vilanova [15]. Essa abordagem alimenta repositórios de software por meio da extração de componentes, mais especificamente, de projetos orientados a objetos do código legado procedimental. A autora considera os itens de grupo de um FD COBOL como candidatos a classes e o trabalho realizado aqui segue a mesma linha de raciocínio.

Cimitile et al apresenta um método para recuperar objetos de código legado procedimental com alto grau de abstração [7]. Os objetos são obtidos por meio dos arquivos de dados e, as operações, através de rotinas e parágrafos dos programas COBOL, assim como é realizado por este trabalho.

Um processo de reengenharia incremental é apresentado por Wiggerts [16]. Objetos são identificados no código procedimental através da aplicação de um dos três cenários propostos: a) dirigido à função, b) dirigido aos dados ou c) dirigido aos objetos. Após a identificação dos objetos, o código legado que originou cada objeto, é descartado e são adicionadas invocações apropriadas de métodos para que o novo objeto tenha interação com o restante da aplicação que continua em linguagem procedimental. Esse tipo de reengenharia resulta em uma aplicação híbrida.

Cagnin et al [3] realizaram reengenharia parcial do ambiente *Statsim* utilizando o método de engenharia reversa orientado a objetos Fusion/RE [12]. A linguagem Java e o banco de dados relacional *Sybase* também foram utilizados, e, os conflitos de comunicação entre objetos e tabelas do banco relacional foram tratados com a implementação dos padrões *Persistence Layer* e CRUD [19]. O padrão *Persistence Layer* trata as incompatibilidades entre objetos e banco relacional por meio de uma camada de persistência responsável pelas conversões necessárias. A classe *Broker* implementada por Cagnin foi adaptada e reusada pelo trabalho apresentado por este artigo [3].

2.2 – Reengenharia/Migração para Web

MORPH é uma técnica que foi originalmente desenvolvida para realizar a reengenharia de interfaces baseadas em caractere para interfaces gráficas. Em [10], os autores a estenderam para realizar a reengenharia de interface para Web, trocando a iniciativa de diálogo, de preemptivo ao sistema, para preemptivo ao usuário. MORPH realiza a reengenharia e também essa reestruturação em três passos: a) Detecção; b) Transformação e c) Geração e Reestruturação.

Sistemas legados, freqüentemente, apresentam subrotinas que contém tanto código de interface do usuário, quanto código de acesso a banco de dados e, sendo assim, Canfora et al, apresenta um algoritmo para realizar a separação [6]. O algoritmo utiliza uma representação gráfica obtida por meio da análise do fluxo de controle dos programas. A separação desses componentes foi realizada neste trabalho com o auxílio do padrão de projeto *Persistence* Layer [19].

Um projeto para migrar sistemas legados COBOL para Web também é apresentado por Aversano et al [1]. O componente interface do usuário foi submetido a reengenharia, utilizando a técnica MORPH, com *Active Server*

Pages (*ASP*) e *VB Script*, enquanto que o componente servidor foi empacotado (*wrapped*) com bibliotecas do *Microsoft Object* COBOL. A autora diz que, embora a técnica de migração utilizada seja rápida, o empacotamento da aplicação legada é um ponto negativo do processo, devido à má qualidade dos programas legados. Dessa forma, não se pode esperar benefícios quanto à manutenibilidade e evolução do sistema.

Cimitile et al propõe estratégias de migração incremental de sistemas legados para uma nova arquitetura cliente/servidor [8]. A estratégia consiste de seis fases, sendo as três primeiras relacionadas à engenharia reversa e as últimas à reengenharia, empacotamento e substituição incremental dos componentes. O artigo utiliza o algoritmo proposto em [6] e objetiva identificar os componentes e, gradualmente, substituí-los por novos, até que o sistema todo utilize os conceitos da orientação a objetos.

Patil et al, apresenta um processo para realizar a reengenharia de sistemas procedimentais visando ambientes baseados em rede [11]. As primeiras etapas do processo realizam a engenharia reversa do sistema legado e produzem um modelo de classes especificado em UML [2] [18].

Sneed, apresenta uma técnica para reengenharia de interface como passo preliminar para o empacotamento da aplicação legada [14]. O objetivo é encapsular velhos componentes de software em arquiteturas orientadas a objetos para que possam se comunicar, por meio das interfaces dos pacotes, com sistemas que utilizam tecnologias recentes.

Tabela 1 – Diretrizes para a Engenharia Reversa de COBOL

ENGENHARIA REVERSA O.O. DE PROGRAMAS COBOL USANDO FUSION/RE			
FASE	**P A S S O**		
	Nº	*NOME*	*DESCRIÇÃO DO PASSO*
I Elaboração Do MASA	I.1	Identificação dos Principais FD's	Identifica-se os módulos do sistema legado que incluem dados nos arquivos.
	I.2	Mapeamento dos Principais FD's para pseudo-classes	Tranforma-se cada FD identificado no Passo 1 em uma Pseudo-Classe.
	I.3	Identificação de Itens elementares e de Grupo	Identificar os itens elementares (pseudo-atributos) e itens de grupo (agregações) nos FDs do passo anterior. Três casos surgem quando da redefinição (REDEFINES) de Itens de Grupo
		I.3.1 – Redefinição de item elementar em item de grupo, (caso 1).	Será criada uma classe parte com mesma semântica de um atributo da classe todo.
		I.3.2 – Redefinição de item de grupo em outro (caso2)	Classes partes utilizam atributos em comum da classe todo.
		I.3.3 – Redefinição de item de grupo geral em outro (caso 3).	Um item de grupo de âmbito geral, geralmente nível 01, é redefinido em outro, gerando dois registros distintos.
	I.4	Identificação de Relacionamentos de Associação	Analisa-se os FD´s em busca de chaves de outros FD´s. Caso seja encontrada, cria-se um relacionamento
	I.5	Identificação dos pseudo-métodos e suas Anomalias	Cada parágrafo transforma-se em um pseudo-método,
II Elaboração do MAS	II.1	Eliminação das Pseudo-Classes que Representam Datas	Elimina-se as pseudo-classes que representam datas
	II.2	Identificação de Relacionamentos de Generalização/Especialização	Analisa-se as agregações do MASA que se originaram dos 3 casos da utilização do REDEFINES. Para cada caso um tratamento é sugerido.
		II.2.1 – Caso 1 - (Item I.3.1)	Elimina-se o atributo que tem a mesma função da classe Parte.
		II.2.2 – Caso 2 – (Item I.3.2)	Relacionamentos de agregação são substituídos pelos de generalização /especialização
		II.2.3 – Caso 3 – (Item I.3.3)	Elimina-se a classe Parte que é dependente da implementação.
	II.3	Conversão de Nomes	Classes, atributos e métodos tem seus nomes alterados para mnemônicos mais significativos.
	II.4	Identificação dos Métodos	Transforma-se cada pseudo-método em um ou mais métodos.
	II.5	Refinamentos Adicionais	Alterações que podem facilitar a leitura e a compreensão do modelo são feitas.

3 – O PROCESSO DE REENGENHARIA

Foi realizado um estudo da reengenharia orientada a objetos de sistemas COBOL para ambientes cliente/servidor estilo *thin-client* [4] Desse estudo, pôde-se abstrair um processo para a realização de tal reengenharia. Esse processo constitui-se de diretrizes para a engenharia reversa e reengenharia e pode ser seguido por engenheiros de software que tem interesse no tipo de sistema alvo obtido. O processo é dividido em duas etapas: a engenharia reversa e a reengenharia, e são descritas nas Seções 3.1 e 3.2, respectivamente.

3.1 – Engenharia Reversa com Fusion/RE

O método Fusion/RE [12]foi considerado para realizar a engenharia reversa. Ele já foi aplicado a sistemas implementados em C [3] e Clipper [13] e, neste trabalho, foi utilizado para sistemas implementados em COBOL. Devido às características próprias de cada linguagem, há necessidade de especializar o método para cada um dos domínios. Assim, foram criadas diretrizes específicas que orientam a aplicação desse método, a fim de realizar a engenharia reversa de sistemas COBOL.

A aplicação dessas diretrizes resulta em dois modelos de análise, o MASA (Modelo de Analise do Sistema Atual) e o MAS (Modelo de Análise do Sistema). O MASA é um modelo de pseudo-classes, construído a partir do código fonte procedimental, inferindo decisões de projeto e de implementação que foram adotadas quando do desenvolvimento do sistema legado. O segundo, MAS, é um modelo de classes totalmente orientado a objetos abstraído do MASA.

A Tabela 1 apresenta as diretrizes que foram criadas. As diretrizes de I.1 a I.5 são responsáveis pela geração do MASA. A diretriz I.3, que é responsável pela identificação de itens elementares e de grupo, destina-se a analisar esses itens e gerar candidatos a atributos (pseudo-atributos) ou a classes (pseudo-classes). Nesse passo notou-se a existência de três tipos de utilização do comando `REDEFINES` que influenciam a forma de obtenção das classes [4][5]. O processo completo de engenharia reversa pode ser obtido em [5] e [4].

3.2 - Reengenharia para a Web

As próximas subseções tratam da reestruturação que deve haver quando da reengenharia de sistemas legados *stand-alones* para ambientes cliente/servidor, como é o caso da Web.

3.2.1 – Identificação e Reimplementação do Componente Interface do Usuário

O componente interface do usuário deve ser migrado para o cliente e deve ser preemptivo ao usuário e não ao sistema. A mudança da iniciativa de diálogo consiste em substituir as antigas interfaces, baseadas em *caracter*, por novas interfaces gráficas [10]. Neste trabalho as interfaces do sistema legado são identificadas com o auxílio de grafos de fluxo de controle e, posteriormente, implementadas em HTML. Essa identificação consiste em procurar, em nós do grafo, declarações COBOL que são responsáveis por entrada de dados no sistema, como por exemplo `ACCEPT`. Geralmente esse comando armazena em estruturas de dados, da `Working Storage Section`, os parâmetros da interface que foram digitados pelo usuário. Tendo identificado essas estruturas de dados, deve-se realizar uma análise dinâmica e estática da interface a fim de reimplementá-la em HTML. Alguns mapeamentos são sugeridos para o desenvolvimento da interface HTML. Por exemplo, declarações `USING` devem ser mapeadas para caixas de texto em HTML e declarações VALUE para rótulos.

3.2.2 – Mudança no Meio de Armazenamento

Como o sistema submetido à reengenharia é implementado em COBOL e realiza a persistência dos dados em formato específico dessa linguagem, foi necessário alterar o meio de armazenamento. Sendo assim, o projeto de banco de dados relacional, implementado em *SyBase*, foi elaborado com base nas classes persistentes da aplicação. Isto é, para cada classe cria-se uma tabela correspondente, para cada atributo cria-se uma coluna, para relacionamentos de associação, chaves estrangeiras e para relacionamentos de herança, novas tabelas ou novas colunas da tabela.

3.2.3 – Separação dos Componentes Lógica de Aplicação do Componente Banco de Dados

A terceira forma de implementação do padrão de projeto *Persistence Layer*, proposta por Yoder [19], é utilizada neste trabalho, sendo que a classe *Broker* é reutilizada e adaptada de Cagnin [3]. A utilização desse padrão separa o componente banco de dados dos outros dois: lógica da aplicação e interface com o usuário. Isso é ideal para o tipo de reengenharia realizada aqui, cujo objetivo é a separação desses componentes.

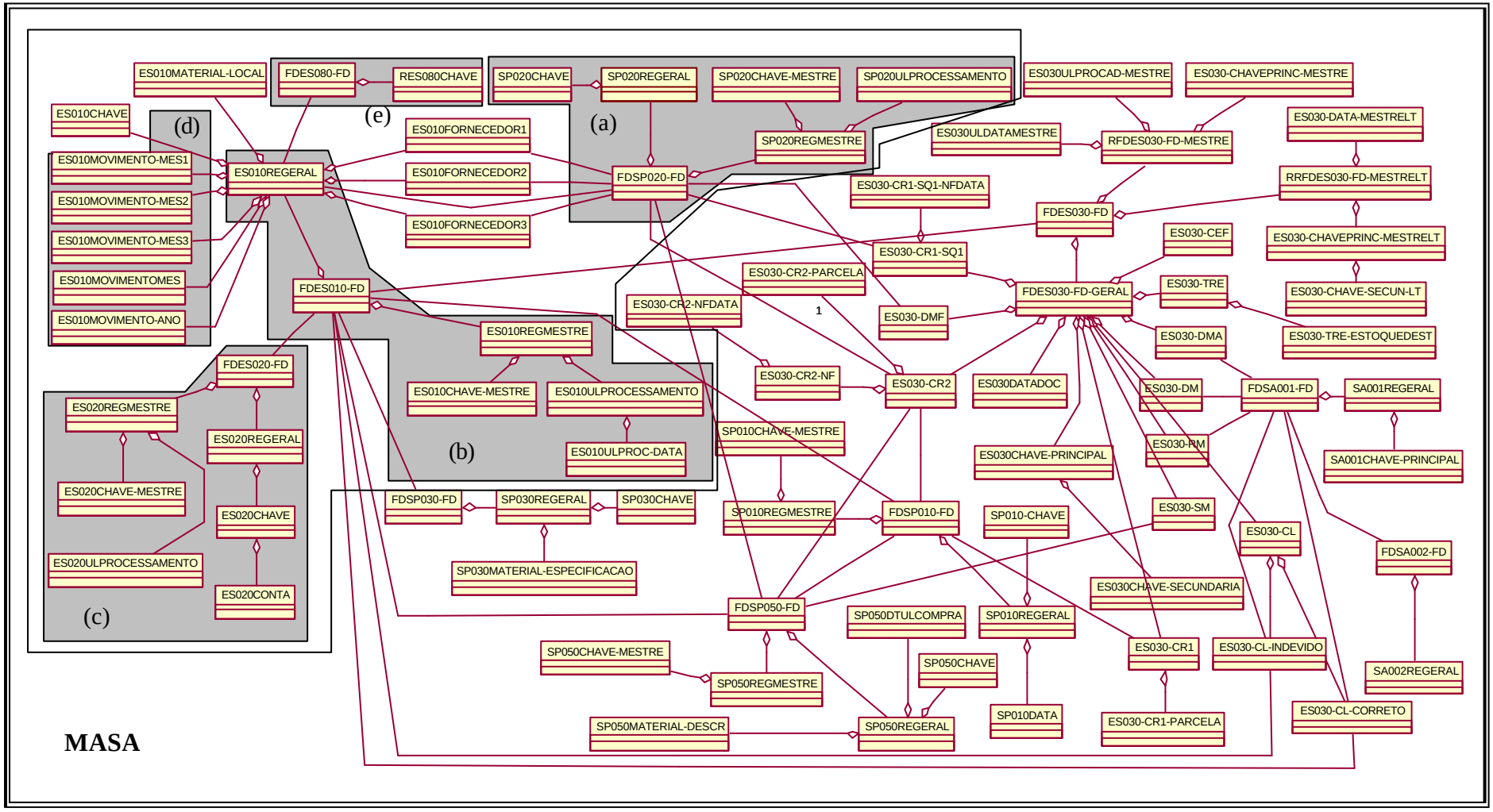

Figura 1 – Modelo de Análise da Solução Atual (MASA)

3.2.4 – Desenvolvimento do Middleware

Após a criação das interfaces HTML deve-se definir como seus parâmetros são enviados para a aplicação. É necessário definir um *middleware* que trate as requisições/respostas dos clientes ligados à rede. Optou-se pelos *servlets*, da linguagem Java, por vantagens que apresentam e por serem adequados a aplicações cliente/servidor com estilo *thin-client* [9]. Devem ser criados vários *servlets*, um para cada operação de classe invocada por meio da interface. Por exemplo, para a operação de inclusão de um novo material, deve-se criar um *servlet* denominado `srvCadastraMaterial`. Além disso, os *servlets* devem manter o fluxo de controle similar ao do sistema legado. Isso é feito gerando um grafo de fluxo de controle para o trecho de código legado que possui correspondência funcional com o *servlet* a ser criado. A implementação desse *servlet* é baseada nesse grafo.

4 - ESTUDO DE CASO

Utilizou-se como estudo de caso um sistema de controle de estoque, implementado em COBOL Microfocus 85. Ele possui 498 módulos, sendo que sessenta e nove são programas e o restante *CopyFiles*[1]. Possui setenta e quatro (74) Kloc e é bem estruturado. Sua funcionalidade inclui o cadastro de materiais, fornecedores, contas, previsão de compras e algumas movimentações, como por exemplo, requisição de material, solicitação de material, comunicado de recebimento, devolução de materiais ao fornecedor, etc. O processo de reengenharia foi realizado em dezesseis módulos do sistema, que foram considerados por apresentarem funcionalidade mais interessante. Durante o processo identificou-se dois tipos de módulos, os construtores e os observadores. Os construtores são aqueles que realizam persistência de dados e que originam classes no modelo de análise. Já, os observadores apenas consultam os dados, não sendo responsáveis pela geração de novas classes. Apesar de que o processo utilizou apenas dezesseis módulos, essa parte é a principal do sistema, pois é onde se concentram todos os módulos construtores As próximas seções comentam o processo realizado.

4.1 – Engenharia Reversa Orientada a Objetos

O entendimento do sistema foi conseguido através do uso da ferramenta de compreensão de programas *Legacy Aid* [17]. O MASA (Figura 1) é obtido quando da aplicação da primeira fase das diretrizes, já, o MAS (Figura 2), que corresponde à parte destacada com linha em negrito na Figura 1, foi elaborado aplicando-se as diretrizes da segunda fase e todos os refinamentos adicionais [4][5]. Cada conjunto de pseudo-classes rotulado com determinada letra no MASA foi transformado nas classes rotuladas com a mesma letra no MAS, por meio da aplicação das diretrizes citadas. Comparando essas duas figuras pode-se notar que houve redução acentuada do número de classes. Essa redução ocorre devido à mudança de paradigma e é uma das vantagens da engenharia reversa orientada a objetos.

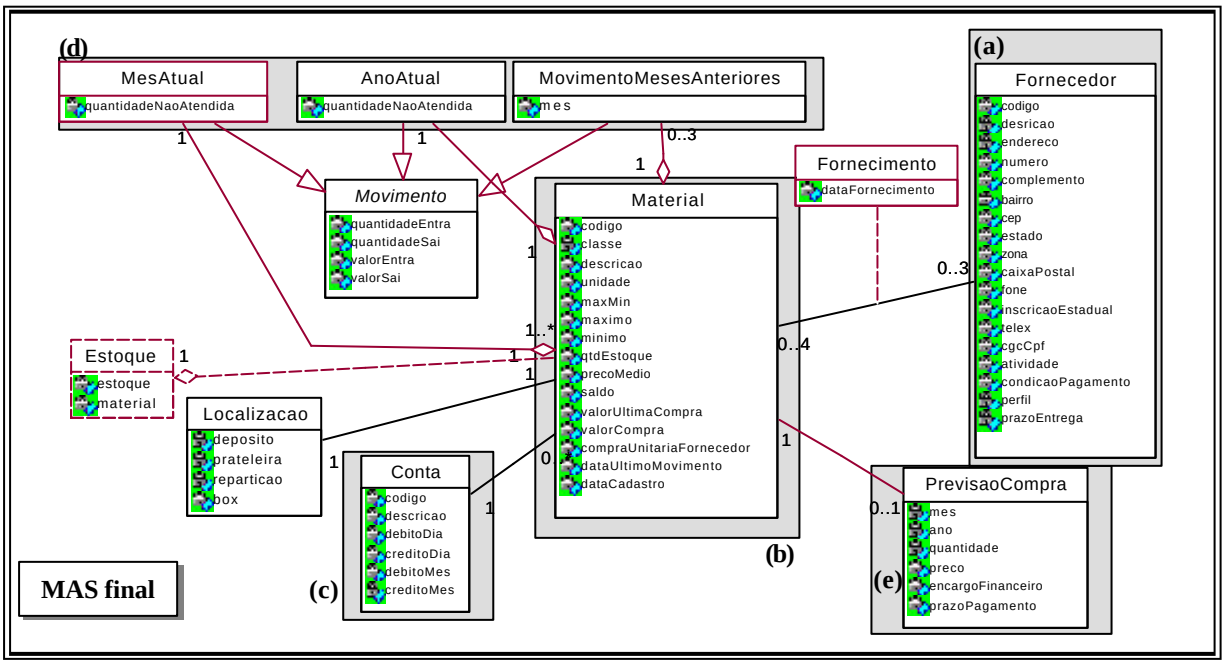

Figura 2 – Modelo de Análise da Solução Atual (MAS)

[1] Arquivos que são copiados para dentro dos módulos principais em tempo de execução.

4.2 – Engenharia Avante Orientada a Objetos para a Web

4.2.1 – Identificação e Reimplementação do Componente Interface do Usuário

Como citado na seção 3.2.1, deve-se utilizar grafos de fluxo de controle para identificar o componente interface do usuário. A ferramenta *Legacy Aid* foi utilizada para gerar tais grafos e fazer as identificações. A Figura 3 apresenta uma parte de um grafo de fluxo de controle gerado pela ferramenta. O grafo mostrado à esquerda é o nível mais alto de abstração, porém, expandindo-se a declaração `PERFORM 10-00-INCLUSAO,` que transfere o controle para o parágrafo `10-00-INCLUSAO,` obtém-se um nível de abstração mais baixo, representado pelo grafo à direita. Nesse grafo encontra-se uma declaração `ACCEPT TELA-10` que obtém dados da interface do usuário. `TELA-10` é uma estrutura de dados definida na `WORKING-STORAGE` que contém variáveis responsáveis por armazenar os dados digitados pelo usuário. Identificar essa estrutura de dados no código legado é fundamental para o projeto de uma nova interface.

Por meio de análise estática identifica-se na `WORKING-STORAGE SECTION` a estrutura de dados correspondente à `TELA-10`, mostrada parcialmente na parte superior da Figura 4. `REG10-MATERIAL`, `REG10-MATDESCRICAO`, `REG10-MATUNIDADE`, `REG10-CONTA`, dentre outras, são variáveis responsáveis por armazenar os dados digitados pelo usuário e que devem ser parâmetros na nova interface.

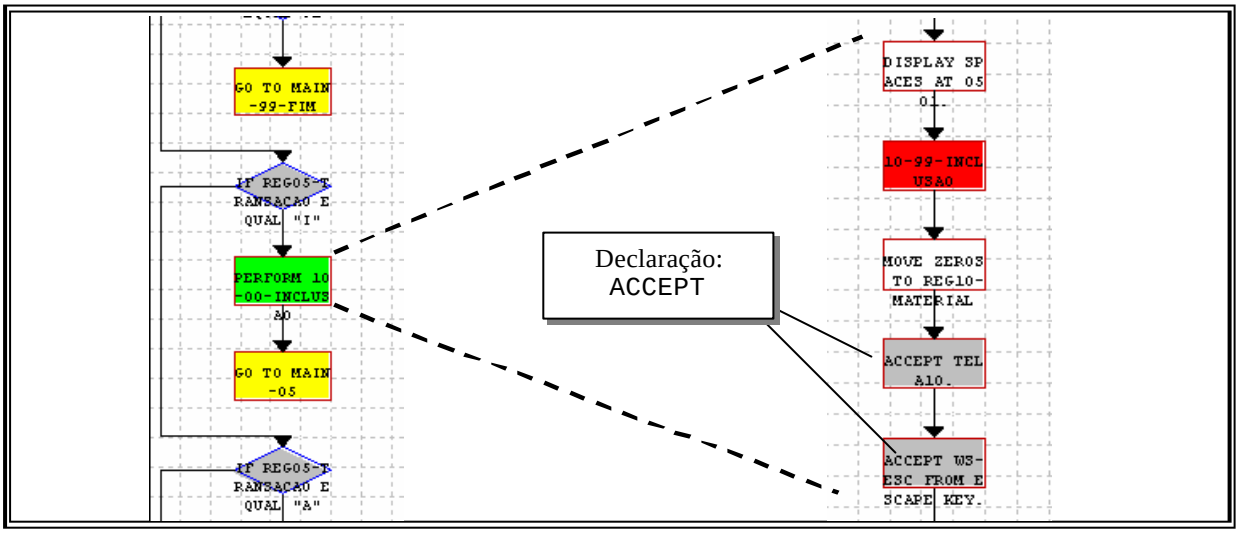

Figura 3 – Grafo de Fluxo de Controle

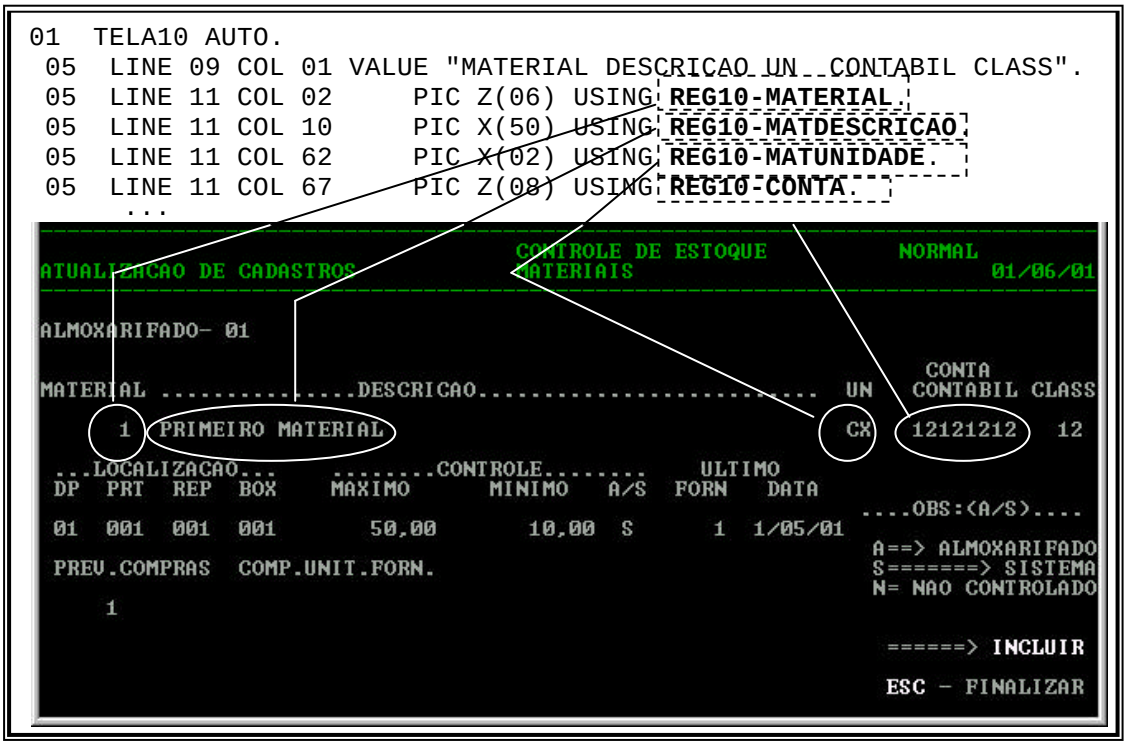

Figura 4 - Relação Estrutura de Dados/Interface

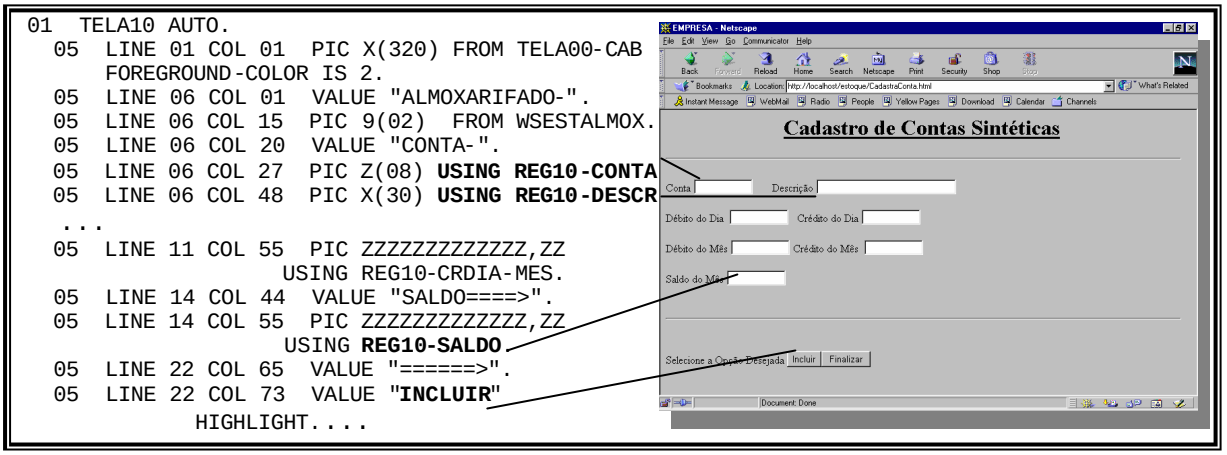

Figura 5 – Interface HTML do Sistema Alvo

Por meio de análise dinâmica devem-se identificar as características da interface do sistema legado, para que a nova interface seja projetada similarmente. A parte inferior da Figura 4 apresenta a interface do sistema, referente à estrutura de dados mostrada na parte superior. As áreas destacadas com elipses correspondem às variáveis em negrito da estrutura.

As novas interfaces foram desenvolvidas com a linguagem de marcação HTML, devido à sua portabilidade em relação aos *browsers* encontrados no mercado. Além disso, a utilização exclusiva dessa linguagem para a construção da interface é ideal para aplicações cliente/servidor do estilo *thin-client*, que é o objetivo deste trabalho.

Alguns mapeamentos são sugeridos para o desenvolvimento de interfaces HTML a partir de interfaces implementadas em COBOL. Para declarações `USING`, como mostra a estrutura de dados da Figura 5 deve-se criar caixas de texto. Já os rótulos dessas caixas de texto podem ser identificados pela declaração `VALUE`, na mesma estrutura.

A Figura 5 apresenta a correspondência que deve haver entre os mapeamentos sugeridos. A interface apresentada por essa figura, bem como a estrutura de dados, corresponde ao cadastro de contas sintéticas.

4.2.2 – Mudança no Meio de Armazenamento

O projeto do banco de dados é realizado com base nas classes persistentes de aplicação, isto é, para cada classe cria-se uma tabela correspondente, para cada atributo cria-se uma coluna, para relacionamentos de associação, chaves estrangeiras e para relacionamentos de generalização/especialização, novas tabelas ou novas colunas da tabela. No caso desse último tipo de relacionamento, este trabalho fez como Cagnin [3], criou tabelas somente para subclasses e adicionou a elas os atributos da superclasse. Um exemplo de como deve ser elaborado o projeto de banco de dados é apresentado na Figura 6. A classe abstrata `Movimento`, que é uma classe de aplicação, foi criada durante a engenharia reversa somente para acomodar atributos comuns às classes que a especializam, sendo assim, não foi mapeada para uma tabela do banco relacional.

As classes persistentes de aplicação `MesAtual`, `AnoAtual` e `MovimentoMesesAnteriores` são classes Parte da classe Todo `Material`, e, possuem tempo de vida coincidente com essa. Sendo assim, foram mapeadas para tabelas do banco relacional, porém, são entidades fracas. As classes `Material`, `Fornecedor` e `Fornece` também são classes persistentes e foram mapeadas para tabelas do banco de dados relacional.

4.2.3 – Separação dos Componentes Lógica de Aplicação e Banco de Dados

A partir do MAS desenvolve-se o projeto do sistema alvo com base nas tecnologias que serão utilizadas na nova implementação. Neste trabalho, a linguagem Java e o banco de dados relacional *Sybase* foram considerados. Além disso, a terceira forma de implementação do padrão de projeto *Persistence Layer* é utilizada. A Figura 7 mostra o modelo de projeto e como as classes do *Persistence Layer* devem ser relacionadas com as classes de aplicação do MAS para que a separação dos componentes banco de dados e lógica da aplicação seja realizada. Todas as classes persistentes da aplicação devem implementar o comportamento da interface `PersistentObject` e associar-se à classe `TableManager`, embora somente algumas sejam exibidas nessa figura. Nota-se que a utilização desse Padrão separa o sistema em duas camadas: camada de aplicação e de persistência. Isso é ideal para o tipo de reengenharia realizada aqui, cujo objetivo é a separação do componente lógica da aplicação do componente banco de dados.

Após a elaboração do modelo de projeto, as classes da camada de persistência e as da aplicação são implementadas utilizando a linguagem orientada a objetos Java [9]. Como no estudo de caso há interesse no estilo *thin-client*, deve-se colocar tanto o componente da lógica da aplicação, (camada de aplicação), quanto o componente banco de dados, (camada de persistência), no servidor. Isso faz com que o lado do cliente necessite de pouca configuração e suporte.

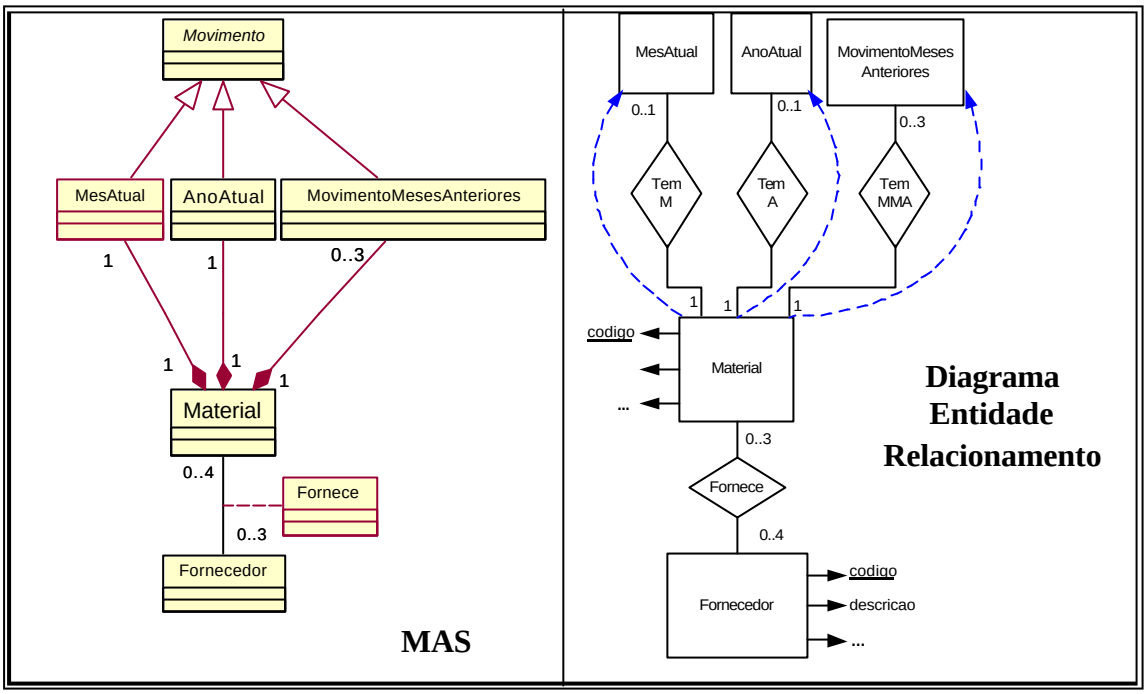

Figura 6 – Projeto de Banco de Dados Relacional

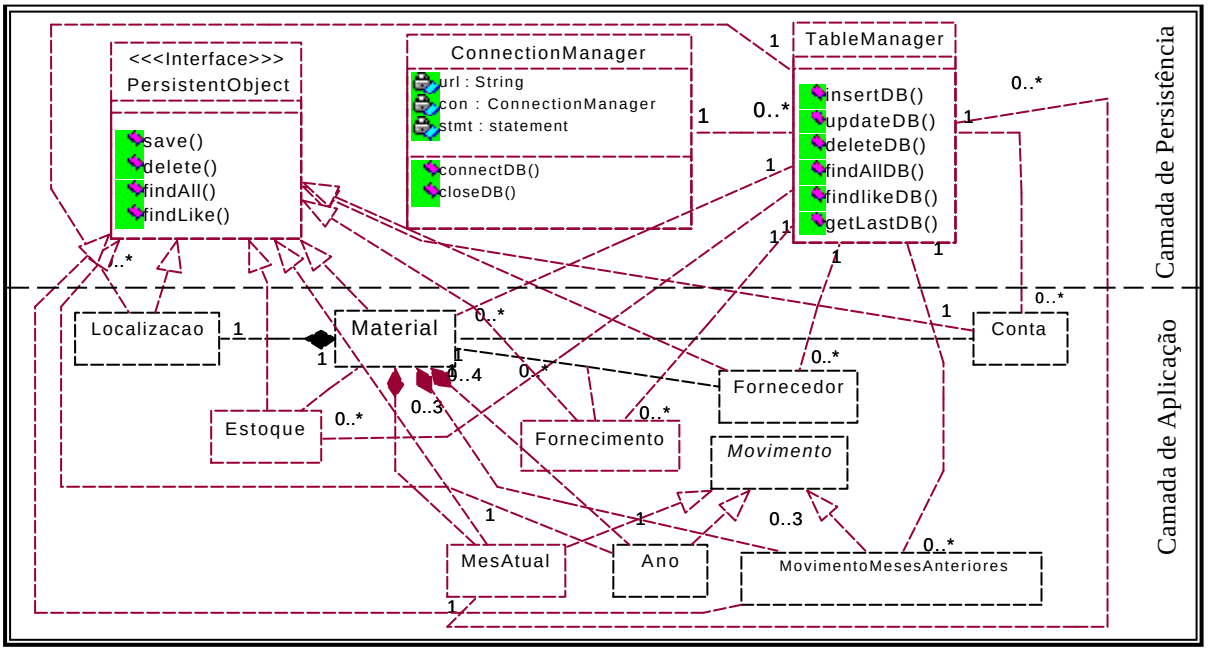

Figura 7 – Modelo de Projeto (Separação dos Componentes)

4.2.4 – Desenvolvimento do Middleware

Em um sistema com arquitetura cliente/servidor, as interfaces não podem invocar diretamente os métodos da camada de aplicação, já que essa se localiza em outro computador. Sendo assim, o mecanismo escolhido para realizar essa comunicação foi o *servlet* da linguagem Java, por ser adequado para aplicações cliente/servidor do estilo *thin-client* [9].

A criação dos *servlets* é um ponto importante, pois devem agir como mediadores entre a interface e a aplicação, e manter o mesmo fluxo de controle do sistema legado. Devem ser criados vários *servlets*, um para cada operação de classe que é invocada por meio da interface. Isto é, para a operação de inclusão de um novo material, deve-se criar um *servlet* denominado, por exemplo, `srvCadastraMaterial`, para a operação de exclusão, um *servlet* denominado `srvRetiraMaterial`, e assim por diante. Além disso, a criação deles deve manter o fluxo de controle similar ao do sistema legado, como apresentado pela Figura 8. Nessa Figura, as setas indicam a

correspondência entre o código legado COBOL, o nós do grafo e o código Java. Dessa forma, o fluxo de controle do sistema alvo se torna similar ao do legado.

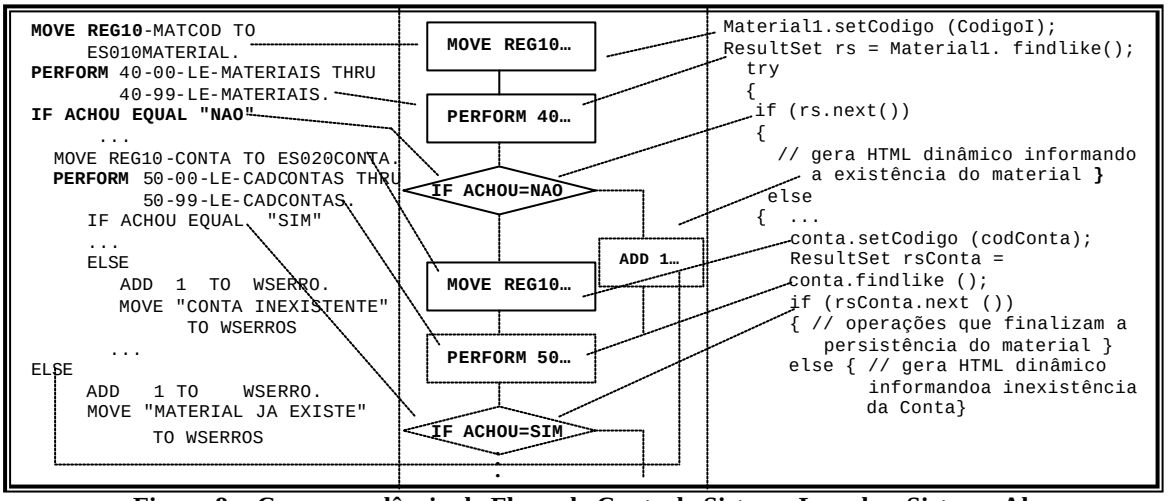

Figura 8 – Correspondência de Fluxo de Controle Sistema Legado - Sistema Alvo

5 – CONSIDERAÇÕES FINAIS

As diretrizes criadas para a realização da engenharia reversa de sistemas COBOL geram dois modelos de análise que podem ser utilizados tanto para tarefas de manutenção e evolução, quanto para o processo de reengenharia. Também foram criadas diretrizes para a reengenharia, que se concentram em identificar/reimplementar as interfaces do sistema legado, bem como, criar *servlets* com base em grafos de fluxo de controle para cada operação invocada por meio da interface. O objetivo é disponibilizar o sistema alvo na Web devido a vantagens como: a) a funcionalidade do sistema pode ser utilizada pelos clientes ligados à rede, sem elevar custos com instalação em cada estação de trabalho; b) padronização das interfaces, que são executadas em um *Web Browser*; c) a integração com tecnologias como CORBA, EJB, é facilitada.

A utilização do padrão *Persistence Layer* é diferencial em relação a trabalhos existentes [1][6][7][10]. A classe *Broker* [3], foi reutilizada e adaptada completamente, aumentando a manutenibilidade, reusabilidade e amenizando os esforços durante o desenvolvimento.

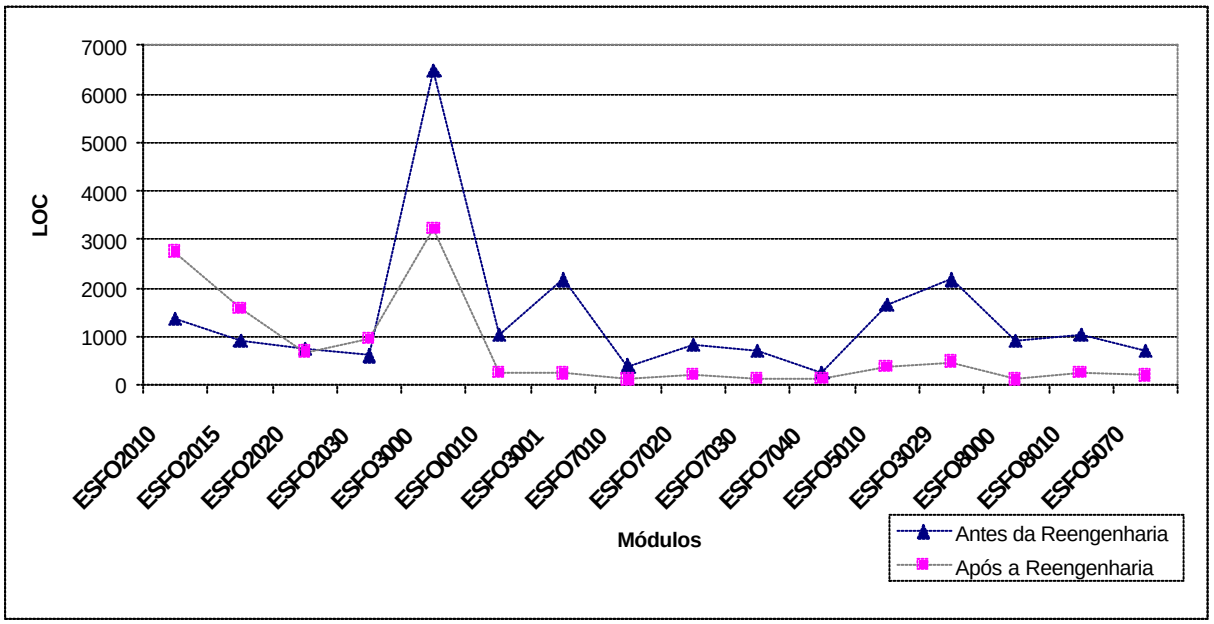

Figura 9 – Kloc Antes e Após a Reengenharia

Uma comparação entre o sistema legado e o alvo foi realizada antes e após a reengenharia com o objetivo de avaliar o produto resultante desse processo. Alguns módulos do sistema legado foram considerados construtores e, outros, observadores. Os construtores geram classes, interfaces e *servlets*, enquanto que os observadores não geram classes. A reengenharia dos construtores mostrou que a soma das linhas de código das classes, interfaces e *servlets* aumentam em comparação às do módulo legado. Isso acontece porque os `FD´s` desses módulos geram

classes e, cada uma terá no mínimo, métodos de atribuição (*setXxxx*) e leitura (*getXxxx*) para todos os seus atributos. Além disso, a quantidade de itens de grupo no FD também interfere na quantidade de classes geradas, já que, cada um gera uma classe no modelo de análise. Já, na reengenharia dos módulos observadores, há diminuição drástica da quantidade de linhas de código em relação ao módulo legado. Isso acontece porque esses módulos não geram novas classes, apenas utilizam as já existentes. Em alguns casos houve redução de aproximadamente noventa por cento (90%).

Realizando-se uma média da porcentagem de linhas de código que foi reduzida após a reengenharia, obtém-se o valor de quarenta e sete por cento (47%). Esse valor indica que houve uma redução de quase metade da quantidade de linhas de código em relação à mesma parte do legado. Porém, como nem todos os módulos observadores foram submetidos à reengenharia, estima-se que esse valor tende a diminuir à medida que eles são submetidos a esse processo. A Figura 9 apresenta um gráfico de comparação entre a quantidade de linhas de código dos módulos antes e após a reengenharia. Nota-se que para os módulos construtores a quantidade de linhas de código aumentou, exceto para o módulo ESFO2020 que deu origem à apenas uma classe, e para o ESFO3000, que não foi completamente considerado no processo. Para os módulos observadores a tendência é a diminuição drástica das linhas de código.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Aversano, L.; Canfora, G.; Cimitile, A.; De Lucia, A. Migrating Legacy Systems to the Web: na Experience Report. In Proceedings CSMR 2001.

[2] Booch, G.; Rumbaugh, J.; Jacobson, I. UML – Guia do Usuário. Editora Campus, 1ª edição, 2000.

[3] Cagnin, M. I.; Penteado, R. A. D.; Germano, F. R. S.; Masiero, P. C. Reengenharia com Uso de Padrões de Projeto. XIII Simpósio Brasileiro de Engenharia de Software, pág. 273-288, Outubro, 1999.

[4] Camargo, V.V. Reengenharia Orientada a Objetos de Sistemas COBOL com a utilização de Padrões de Projeto e Servlets. Dissertação de Mestrado, Departamento de Computação – UFSCar, Abril – 2001.

[5] Camargo, V.V.; Penteado, R.D. Diretrizes para a Realização da Engenharia Reversa de Sistemas COBOL utilizando Fusion/RE. In Proceedings do CLEI 2001, (CD-ROM).

[6] Canfora, G.; Cimitile, A.; De Lucia, A.; Di Lucca, G.A. Decomposing Legacy Programs: A First Step Towards Migrating to Client-Server Platforms. In Proceedings of 6th IEEE International Workshop on Programa Comprehension, Ischia, Italy, June 1998, IEEE Computer Society Press, pp. 136-144.

[7] Cimitile, A.; De Carlini, U.; De Lucia, A. Incremental Migration Strategies: Data Flow Analysis for Wrapping. In Proceedings of Working Conference on Reverse Engineering, pages 59-68, Outubro 1998.

[8] Cimitile, A.; De Lucia A.; Di Lucca, G.A.; Fasolino, A.R. Identifying Objects in Legacy Systems Using Design Metrics. Elsevier Science 1999.

[9] Deitel, H.M.; Deitel, P.J. Java – Como Programar. Bookman, 3ª Edição, 2000.

[10] Moore, M.M.; Moshkina, L. Migrating Legacy User Interfaces to the Internet: Shifting Dialogue Initiative. In Proceedings on 7th Working Conference on Reverse Engineering, 2000.

[11] Patil, P.; Zou, Y.; Kontogiannis, K.; Mylopoulos, J. Migration of Procedural Systems to Network-Centric Platforms. CASCON'99, Toronto, Canadá, Novembro 1999.

[12] Penteado, R. A. D. Um Método para Engenharia Reversa Orientada a Objetos. Tese de Doutorado – Instituto de Física de São Carlos, Universidade de São Paulo. São Carlos, 1996.

[13] Penteado, R.; Masiero, P.C.; Prado, A.F.; Braga, R.T.V. Reengineering of Legacy Systems Based on Transformation Using the Object – Oriented Paradigm. In 5th Working Conference on Reverse Engineering, Honolulu, Hawai – USA, pg. 144-153, 1998.

[14] Sneed, H. M. – Program Interface Reengineering for Wrapping. In: Fourth Working Conference on Reverse Engineering, Amsterdam, The Netherlands. Proceedings, 1997.

[15] Vilanova, L.O. Reengenharia para Reutilização de Software: Uma abordagem para Recuperação de Projetos Orientados a Objetos. Dissertação de Mestrado - COPPE/UFRJ, 1997.

[16] Wiggerts, T.; Bosma, H.; Fielt, E. Scenarios for the Identification of Objects in Legacy Systems. IEEE software p. 24-32, 1997.

[17] www.casemaker.com. Último acesso em 13/07/2002.

[18] www.rational.com. Último acesso em 13/07/2002.

[19] Yoder, J. W.; Johnson, R. E.; Wilson, Q. D. – Connection Business Objects to Relational Databases. In: Conference on the Pattern Languages of Programs, 5, Monticello-IL, EUA, 1998.