Reengenharia de aplicações cliente/servidor para a plataforma Web usando metamodelos

Cláudio Roberto de Lima Martins

Marcelo Soares Pimenta

Ana Maria de Alencar Price

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil
+55 (51) 3316-6159, 3316-6168, fax: +55 (51) 3316-7308
{cmartins, mpimenta, anaprice}@inf.ufrgs.br

Resumo

Um desafio presente hoje é a reengenharia de sistemas legados para operar na Internet. Uma classe desses sistemas de informação foi implementada na fase inicial do desenvolvimento das aplicações cliente/servidor de banco de dados, usando ambientes visuais com interfaces gráficas tipo WIMP. O que caracteriza essas aplicações é a programação baseada em eventos, em uma arquitetura de duas camadas, onde lógica de negócio, acesso a dados e interface de usuário estão juntos no mesmo código da aplicação cliente. O objetivo deste artigo é propor uma metodologia para migrar sistemas legados com as características citadas acima para a plataforma Web. O processo de migração proposto permite representar em metamodelos componentes extraídos do código fonte do sistema legado, que após tratamentos adequados serão utilizados como artefatos para a geração de modelos UML em uma representação XMI.

Palavras-chave: Migração de sistemas, reengenharia de aplicações cliente/servidor, engenharia reversa, metamodelos.

Abstract

A present challenge today is legacy systems reengineering to operate on the Internet. A subclass of those information systems was implemented in the initial phase of the development of the applications database client/server, using visual environment with WIMP graphic interfaces. The main characteristics of those applications is the adoption of event-based programming, in a two layers architecture, where business rules, persistent data access and user interface are jointly considered in the application. The goal of this paper is to propose a methodology to support migration of legacy systems with the characteristics mentioned above for the Web environment. The proposed migration process allows representing by metamodels the extracted components of the legacy system source code, which will be used as engines after appropriate treatment for the generation of models UML in a XMI representation.

Keywords: Systems migration, client/server reengineering, reverse engineering, metamodels.

1 Introdução

Atualmente, muitas organizações estão buscando atualizar seus sistemas de informação legados para a Internet, em face da consolidação e das vantagens advindas dessa plataforma. Uma parte desses sistemas foi desenvolvida quando se iniciou o processo conhecido por "*downsizing*", que defendia a substituição completa dos sistemas no ambiente de *mainframe* (centralizados), para sistemas em um modelo descentralizado utilizando a arquitetura cliente/servidor. Esta decisão acarretou não somente a troca de ambiente operacional, mas uma nova forma de desenvolver os softwares aplicativos.

A maturidade do paradigma orientado a objetos (doravante abreviado OO) induziu a adoção de linguagens implementadas em ambientes visuais, como Visual Basic (VB/Visual Studio, da Microsoft), Visual Object (VO/Clipper, da Computer Associates) e Delphi (Object Pascal, da Borland), que ofereciam recursos OO em forma de componentes pré-definidos. A geração que iniciou o desenvolvimento de aplicações nesse paradigma, aqui chamadas de aplicações WIMP[1], continuava fortemente ligada à cultura procedural e estruturada da geração passada e culminou com uma subutilização do potencial da orientação a objetos. Assim, os recursos OO usados eram aqueles que o ambiente de programação oferecia de pronto: os componentes visuais e os objetos de acesso a dados. O restante era codificado proceduralmente em métodos para tratamento de eventos, sejam de interfaces gráficas (em sua maioria) sejam de tratamento de exceções (erros de execução e manipulação dos objetos de dados), sem aproveitar adequadamente os mecanismos poderosos da programação OO como herança, polimorfismo, encapsulamento, delegação, troca de mensagens, etc. Como conseqüência desta forma "híbrida" de programar (código procedural em um ambiente OO), produziu-se sistemas difíceis de manter, evoluir e adaptar a novas tecnologias e arquiteturas, devido, sobretudo, às seguintes características: lógica de negócio misturada com código de interface de usuário, código SQL embutido na aplicação, divisão funcional centrada em formulários (janelas gráficas) e arquitetura cliente/servidor em duas camadas (modelo conhecido como "cliente-gordo").

Este artigo apresenta uma metodologia para migrar sistemas com as características anteriormente citadas, para um paradigma OO, em uma arquitetura de três camadas, mais especificamente, sistemas ambientados na Web. O processo de reengenharia usado é baseado na recuperação de componentes de alto nível de abstração, em vez de conversão de código em baixo nível de abstração. O mecanismo de migração através de modelos é a solução para a tradução de código, impraticável quando não há similaridade de arquitetura e de linguagens de codificação correspondentes nos sistemas [11, 24].

Os metamodelos gerados são artefatos de representação para os componentes obtidos por engenharia reversa do sistema origem e são armazenados em um repositório. A partir desse repositório será possível construir alguns modelos para o projeto de migração como, por exemplo, o de interface de apresentação do usuário, os casos de uso para descrever as funcionalidades, e o Modelo Conceitual do Sistema (MCS), com as classes de negócio envolvidas. Outro benefício com o uso de metamodelos e repositório é a possibilidade de gerar os modelos em um formato padrão, como o XMI (*XML Metadata Interchange*) [18], que é aceito por ferramentas CASE de modelagem UML [2] que sigam este padrão.

O artigo está organizado como segue. Na seção 2, discute-se alguns trabalhos relacionados à engenharia reversa e formas de representação de software. Na seção 3, é apresentada uma visão geral dos componentes presentes em uma aplicação cliente/servidor baseada em eventos e implementada em um ambiente visual. Na seção 4, descreve-se o processo para a concepção dos metamodelos e discutem-se as diretrizes para construção de modelos de projeto de migração. Comentários finais e trabalhos futuros são apresentados na seção 5.

2 Trabalhos Relacionados

Uma das atividades mais importantes em um processo de reengenharia de software é a engenharia reversa. O objetivo da engenharia reversa de um sistema é identificar informações de projeto, especificações funcionais e, eventualmente, requisitos do código fonte dos programas, criando representações em um nível mais alto de abstração [5]. A engenharia reversa deve produzir, se possível de forma automatizada, documentos que auxiliem a compreensão do sistema de software legado. Isso irá facilitar o reuso, manutenção, teste, redocumentação e, conseqüentemente, o controle de qualidade de software. Para grandes sistemas, modelos e documentos que retratem o entendimento de aspectos estruturais são mais importantes do que o conhecimento de um componente isolado qualquer [29].

[1] WIMP é um acrônimo para "*Window, Icon, Menu, Pointing device* (ou *Pull-down menu*)", um ambiente de interface gráfica para usuário baseada em controles visuais como aqueles fornecidos em Windows e Macintosh, por exemplo.

Alguns trabalhos de pesquisa são encontrados na literatura para a construção de modelos abstratos, a partir do código procedural legado, como em [1, 9, 11, 19, 23, 29, 30]. Essas abordagens se baseiam, na grande maioria, nas estruturas de dados (tipos de dados e variáveis) e nos rotinas (procedimentos e funções) presentes nos códigos fontes utilizando técnicas, como a análise estática e dinâmica para a extração de *conceitos* (classes/objetos) [28].

Entre diversas técnicas e ferramentas que estudam formas de representar um software, temos os visualizadores, analisadores e *browsers* que fornecem uma visão genérica de estruturas de código, produzindo metamodelos simples de diferentes tipos de informação. Exemplos como o sistema Rigi [29], concentra-se na recuperação da estrutura da arquitetura de grandes sistemas, fornecendo ao usuário uma interface flexível de múltiplas visualizações da arquitetura. Outras ferramentas são focadas em linguagens específicas seguindo técnicas como detecção de *clichés*, que comparam estilos e padrões de arquitetura de código, útil no estudo de como representar e usar o conhecimento e experiência de programação [21].

O método Fusion/RE [19] é um exemplo de estratégia de engenharia reversa para mudança de paradigma do software legado procedural no padrão OO. Nesse método, a engenharia reversa é feita em quatro passos, com a finalidade de gerar um modelo de classes totalmente orientado a objetos, chamado de Modelo de Análise do Sistema (MAS). Fusion/RE apresenta um processo para identificação de anomalias de código e métricas para avaliação da qualidade da implementação, úteis na avaliação do esforço de conversão e reengenharia do sistema. Após a engenharia reversa, quatro alternativas podem ser seguidas: uso da documentação produzida para facilitar a manutenção, melhoria da qualidade do sistema atual, desenvolvimento do sistema por reengenharia e conservação da implementação atual, com aperfeiçoamentos futuros usando a orientação a objetos [3].

O projeto *FAMOOS Esprit* (*A Framework based Approach for Mastering Object-Oriented Systems*) [8] trata da reestruturação de software legado OO para métodos e linguagens de programação OO mais atuais. FAMOOS compreende uma metodologia e um conjunto de ferramentas para detectar falhas de projeto em sistemas OO, cujo objetivo é tornar tais sistemas mais eficientes e flexíveis. Entre as diversas preocupações no projeto FAMOOS, destaca-se FAMIX (*FAMOOS Information Exchange Model*), um modelo de representação de software OO independente da linguagem de programação OO. A finalidade do FAMIX é o compartilhamento e troca de informações entre várias ferramentas que utilizam metamodelos de representação OO. Alguns autores [13, 14, 25] destacam o uso de metamodelos genéricos, mantidos em repositórios, pois fornecem um mecanismo mais eficiente para compartilhamento de informação entre diversos tipos de ferramentas CASE. Além disso, facilita o armazenamento, consultas e extração de qualquer informação relacionada à representação do software. Atualmente, FAMIX utiliza os formatos CDIF [4] e, recentemente, XMI [26]. XMI (XML Metadata Interchange) [18] é um novo padrão definido pela OMG para intercâmbio de metamodelos UML entre ferramentas CASE de modelagem OO.

No trabalho aqui proposto, são utilizados os fundamentos básicos de cada enfoque relacionado anteriormente. O processo de reengenharia usado se preocupa com a recuperação de componentes de alto nível de abstração e na reestruturação do código legado para uma abordagem OO. A representação do esquema de implementação (visão de alto nível dos componentes) é realizada usando metamodelos, que é armazenado em um repositório. A partir desse repositório é possível gerar modelos de projeto em um formato padrão, como o XMI (*XML Metadata Interchange*) [18].

3 Visão Geral

A estrutura básica com os principais componentes de uma aplicação WIMP cliente/servidor é vista na Figura 1. O estereótipo "*implClass*", semelhante ao estereótipo "*implementation class*" [17], é uma classe que fornece uma implementação física através de componentes fornecidos pelo ambiente de programação. A seguir, são descritas as funcionalidades de cada componente.

Aplicação – compreende todos os elementos físicos do sistema com pelo menos uma unidade modular. Normalmente, um projeto de aplicação é armazenado em uma estrutura de diretório contendo os arquivos de códigos fonte e executável.

Módulo – representa uma unidade física de código, como um arquivo fonte, que pode referenciar outros módulos (por meio de chamadas do tipo *uses*, *include* ou *import*, por exemplo), além de conter rotinas, *containers* visuais e objetos de dados *DataSet*. Exemplos de módulos são as *units* e *classes*, em Delphi e Java, respectivamente.

DataSet – é todo componente de manipulação de uma tabela, através de métodos apropriados para incluir, excluir e alterar registros de dados, acesso a atributos e outros recursos. Como é um objeto, deve ser declarado a uma variável objeto possuindo algumas operações e propriedades.

Container Visual – componente de interface visual contendo outros componentes visuais (controles visuais), inclusive outros *Containers Visuais*. Em um ambiente WIMP são conhecidos por *widgets containers*, como

formulários, *panels*, *frames* ou caixa de diálogo, por exemplo. Por ser um objeto, deve ser declarado a uma variável ou componente objeto possuindo algumas operações, propriedades e eventos.

Controle Visual – é um componente de apresentação visual de dados, que está acoplado direta ou indiretamente a um *DataSet*. Em um ambiente WIMP, os controles visuais são conhecidos como *widgets* de controle (menus, botões de comando, *text-field*, *text-label* ou rótulos, etc.), isto é, objetos visuais com propriedades, operações e eventos associados.

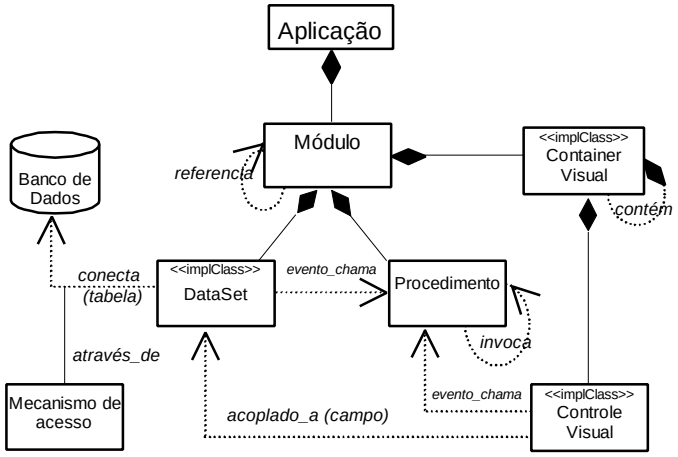

Figura 1. Estrutura básica de uma aplicação WIMP cliente/servidor.

Mecanismo de acesso – interface de acesso e conexão da aplicação com o banco de dados. Os ambientes de desenvolvimento normalmente fornecem bibliotecas de acesso a banco de dados que devem ser instaladas junto com a aplicação cliente. Nessas bibliotecas, o programa cliente estabelece uma conexão assim que é carregado e envia comandos através desta conexão, normalmente, na linguagem SQL. *Middleware* como ODBC (Windows) e BDE (da Borland), são exemplos de mecanismos de acesso.

Procedimento – unidade funcional de código para executar uma rotina, uma função ou implementar um método de um objeto.

4 Processo de Migração

A realização do processo de migração depende do conhecimento da arquitetura do sistema e da identificação das partes que devem ser reimplementadas. Pela quantidade de dados envolvidos, deve-se utilizar técnicas automáticas ou semi-automáticas, como a engenharia reversa para o levantamento da documentação e dos componentes do software a partir dos códigos-fonte e do esquema de banco de dados. Outra técnica menos automática, porém necessária para a compreensão do sistema como um todo, é realizar entrevistas com os especialistas do sistema, recuperando as especificações de requisitos vigentes, manuais e modelos de projeto [3, 19].

Com base nas informações coletadas, deve-se produzir modelos de análise e de projeto de todas as classes e componentes que passarão pelo processo de reengenharia. Isso enfatiza *o que* as classes/componentes fazem, e não *como* se encontram implementadas atualmente.

O processo aqui proposto é definido em três etapas:

1. Elaboração do Modelo Conceitual do Sistema (MCS);
2. Mapeamento das interfaces WIMP para interfaces Web;
3. Construção dos casos de uso de migração.

O Modelo Conceitual do Sistema define todas as classes de negócio envolvidas no sistema, identificadas pela relação entre os objetos de dados e os métodos que manipulam estes dados.

O mapeamento entre interfaces WIMP e Web permite reusar os elementos visuais presentes no sistema legado para a plataforma Web.

A construção dos casos de uso de migração é a etapa responsável pela identificação das funcionalidades e da dinâmica de execução da aplicação.

Nas próximas seções, estas três etapas são descritas com mais detalhes.

4.1 Elaboração do Modelo Conceitual do Sistema (MCS)

Nesta seção, são apresentadas as diretrizes necessárias para a engenharia reversa de um sistema de informação WIMP implementado no paradigma baseado em eventos[2], em linguagens visuais orientas a objetos. Para formar a base da estratégia, alguns conceitos e heurísticas definidas em alguns métodos e trabalhos de pesquisa relacionados ao tema são usados, como em [8, 11, 19].

A estratégia para identificação de *conceitos* (classes com seus atributos e métodos) proposta neste trabalho é baseada na relação entre variáveis objetos de dados (*DataSet's*) e procedimentos (métodos). Para realizar esta estratégia, quatro passos são necessários:

1. definir o perfil da nova aplicação;
2. identificar os objetos de dados (variáveis DataSet);
3. identificar os métodos; e,
4. construir o Modelo Conceitual do Sistema (MCS).

O sucesso para realizar estas atividades dependerá do uso de ferramentas de apoio à extração de fatos, tipo *parsers* e analisadores de código, definidas para a gramática da linguagem visual do sistema origem. Uma descrição detalhada de cada passo é feita, a seguir, com a definição das estruturas de metaclasses do metamodelo a ser mantido em um repositório de migração.

4.1.1 Passo 1: Definição do perfil da nova aplicação

O primeiro passo é definir o grupo de funcionalidades e subsistemas que serão migrados da versão "velha" para a "nova" aplicação, observando um estudo de viabilidade que deve ser elaborado antes de se iniciar o projeto de reengenharia. Devem constar no plano de migração questões como a identificação da audiência (novos usuários), segurança, desempenho, usabilidade e outras relacionadas ao ambiente Web.

O perfil da nova aplicação é construído a partir da identificação dos módulos que serão migrados do sistema legado. Cada módulo é identificado, associando as operações, menus e janelas que o usuário escolhe para a migração. Além das informações do nome do módulo e de sua localização física, algumas métricas de software são levantadas. Métricas auxiliam na avaliação do código fonte em termos de compreensão, complexidade e reusabilidade. São úteis para localizar procedimentos complexos, que podem ser divididos em blocos menores e mais coesos [20]. Entre vários parâmetros de métricas, destacam-se atributos para quantificar linhas de código, comentários e procedimentos declarados no módulo.

4.1.2 Passo 2: Identificação dos objetos DataSet

Objetos *DataSet* provêem o acesso ao banco de dados da aplicação, através de um mecanismo de acesso.

As informações obtidas da conexão com o banco de dados serão úteis para conhecer as particularidades das tabelas de dados e objetos armazenados no banco de dados. A metaclasse *Database* representa essas informações no metamodelo.

Através de engenharia reversa de banco de dados, por consulta às visões do dicionário de dados ou usando uma ferramenta CASE, é possível extrair a estrutura das tabelas, além dos relacionamentos e objetos associados, como índices, *stored-procedures* e *triggers*. A primeira versão do MCS surge das tabelas, dos relacionamentos e dos atributos extraídos do dicionário do banco de dados. Cada tabela pode gerar uma classe, e cada relacionamento uma associação entre as classes envolvidas. Trabalhos como os de Fong & Huang [10] tratam do mapeamento de modelos de banco de dados relacionais para modelos orientados a objetos. Em Gall et al. [11], a transformação de um diagrama entidade-relacionamento (DER) para um modelo estático de classes e objetos é realizada pelas similaridades entre as duas representações: objetos e seus atributos são diretamente derivados das entidades do DER, e as associações "generalizaçao-especialização" e "todo-parte", entre as classes, são mapeadas pelos relacionamentos *is-a* e *part-of* entre as entidades, respectivamente.

[2] Pode-se considerar um paradigma de codificação baseado em eventos um tipo de abordagem procedural, sob o ponto de vista de que a implementação dos métodos para tratar os eventos, constituírem as unidades funcionais em forma de funções ou procedimentos.

Em seguida, são identificadas as formas de manipulação das tabelas, descritas nos *módulos* do sistema. Um módulo pode conter procedimentos que acessam uma ou mais tabelas através dos objetos *DataSet*. Um *DataSet* fornece informações sobre qual tabela de dados é acessada do banco de dados por meio de uma propriedade do objeto ou de um método para execução de comandos SQL.

Nos exemplos da Figura 2, observam-se trechos de código fonte representando um tipo de objeto *DataSet* definido para a variável "***rs***", instanciada a partir de **ADODB.RecordSet** e **TQuery**, classes correspondentes às linguagens VB e Delphi, respectivamente. Em ambos os exemplos, o procedimento é o mesmo: abre-se o objeto *rs*, informando-se o comando SQL para recuperar todos as *tuplas* (registros) da tabela *Customer*, com todas as colunas (campos); depois, preenche-se dois controles do formulário (*txtCustomerID* e *txtCompanyName*) com os valores dos campos correspondentes da tabela. O mesmo exemplo serve para ilustrar o estilo de programação discutido na seção 1, que deve ser evitado, isto é, a mistura no código da aplicação do mecanismo de acesso (SQL) e conexão aos dados com objetos de interface de usuário (controles visuais de formulário).

a) Visual Basic (VB)

```
// --- definição da variável-objeto rs no módulo
Public rs As New ADODB.Recordse t

Private Sub Form_Load()
With rs
.Source = "Select * from Customer"
.ActiveConnection
"dsn=MizMoORCL;uid=SISVEN;pwd=XYZPOO"
.Open
End With
txtCustomerID.Text = rs.Fields("CustomerID") & ""
txtCompanyName = rs.Fields("CompanyName") & ""
End Sub
```

b) Delphi

```
// --- definição da variável-objeto rs na unit
public
    rs: TQuery;
procedure Form_Load;
begin
  rs.SQL.Text:='SELECT * from Customer ';
=   rs.Open;
  txtCustomerID.Text =
      rs.FieldByName("CustomerID").AsString;
  txtCompanyName =
      rs.FieldByName("C ompanyName").AsString;
end;
```

Figura 2. Exemplos de objetos *DataSet* em VB e Delphi.

O metamodelo representando as estruturas relativas ao banco de dados é visto na Figura 3. As metaclasses Tabela e Campo contém as informações sobre as tabelas, os campos destas tabelas e os relacionamentos entre as tabelas. Database é a metaclasse com as informações de conexão ao banco de dados. Na metaclasse DataSet é identificado o objeto (presente no código legado) referente ao acesso a uma tabela do banco de dados.

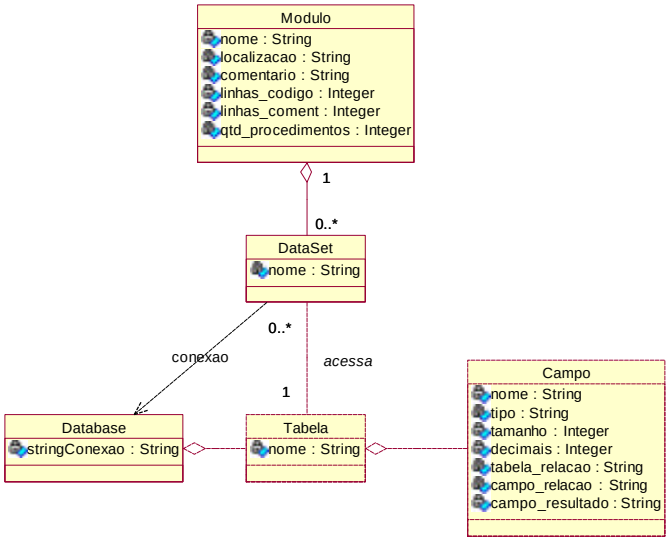

Figura 3. Estrutura de metaclasses com a classe *DataSet*.

4.1.3 Passo 3: Identificação dos métodos

O objetivo deste passo é identificar o relacionamento entre procedimentos (com métodos candidatos) e objetos *DataSet*. Os procedimentos escolhidos são os relevantes para a compreensão das regras de negócio, isto é, os que tratam com os dados de negócio: variáveis e métodos que direta ou indiretamente estejam vinculados a objetos *DataSet*. Exclui-se todos os procedimentos que não pertencem a esta regra, como aqueles que implementam funcionalidades de interface visual, por exemplo.

Satisfeitas as condições de relevância, um procedimento deve ser quebrado em pedaços menores de bloco de código que possam realizar uma única tarefa com forte coesão e baixo acoplamento. Esses blocos de código formarão os métodos que modificam o estado de um objeto *DataSet*. É esperado que um procedimento apresente anomalias do tipo: *acessa mais de um DataSet*, *modifica mais de um campo (atributo)* ou *lê vários atributos*. Neste caso, isola-se o bloco para cada método encontrado que manipule um único *DataSet*, classificando o tipo de manipulação realizada sobre este objeto, conforme os valores definidos, a seguir:

- "**M**", quando o módulo é modificador de um *DataSet*, isto é, quando o código apresenta operações de atualização aos dados (inclusão, alteração e exclusão) de um *DataSet*; e

- "**O**", quando o módulo apenas recupera (observa) valores de um *DataSet*.

Um exemplo de procedimento com anomalias pode ser visto no código da Figura 4, que apresenta um procedimento com a seguinte assinatura: **Calcula_Saldo (CliID: integer, Valor: float)**. A finalidade do procedimento é atualizar o saldo de um cliente com os dados do valor (a debitar do saldo) e o código do cliente passados como parâmetros no procedimento. Após uma análise, pode-se afirmar que três blocos de código podem ser isolados formando os métodos M1, M2 e M3. O método M1 é do tipo "O" (observador do objeto *rs*), enquanto M2 e M3 são do tipo "M" (modificadores do estado de *rs*). O critério de nomeação dos métodos deve respeitar os objetivos do que cada método se propõe a executar, buscando sempre realizar uma única tarefa (regra da alta coesão) [6].

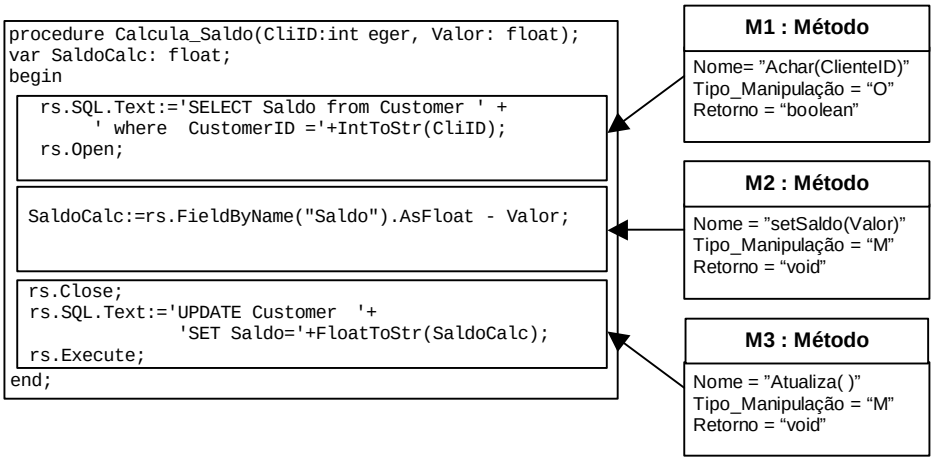

Figura 4. Exemplos de anomalias de código.

O metamodelo (parcial) visto na Figura 5, mostra as estruturas Procedimento e Método. Informações de métricas são adicionadas à metaclasse Procedimento, seguindo os mesmos objetivos apresentados nas métricas de Módulo, com exceção do parâmetro de Pontos de Decisão (**DP** – *Decision Point*). *Pontos de Decisão* (DP) é equivalente à métrica de McCabe [16], conhecida como *complexidade ciclométrica*, uma das métricas mais usadas para estimativa de complexidade [27]. DP é definido por um valor inteiro positivo, que serve como medida para aferir um certo grau de complexidade. A finalidade prática deste valor é que se pode quebrar um procedimento em pedaços menores, assim que o valor de DP ultrapassar um valor limite convencionado para a complexidade, por exemplo, um valor acima de 10. Outra forma de uso, é classificar o procedimento em graus de risco quanto à capacidade de o código receber modificações, dependendo de faixas de complexidade.

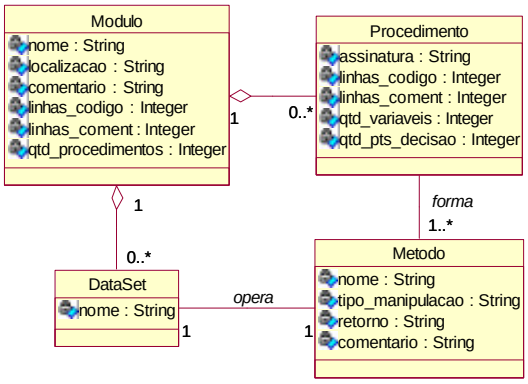

Figura 5. Modelo com as estruturas Procedimento e Método.

4.1.4 Passo 4: Construção do Modelo Conceitual do Sistema

Para a construção do Modelo Conceitual (de classes de negócio) do Sistema, MCS, os componentes obtidos por engenharia reversa e registrados no metamodelo de metaclasses são utilizados. Como pode ser visto na Figura 6, as metaclasses que fornecem as informações para a criação do MCS são: as instâncias de Tabela, Campo e Método que assumirão os papéis de candidatos a classes, atributos e métodos, respectivamente.

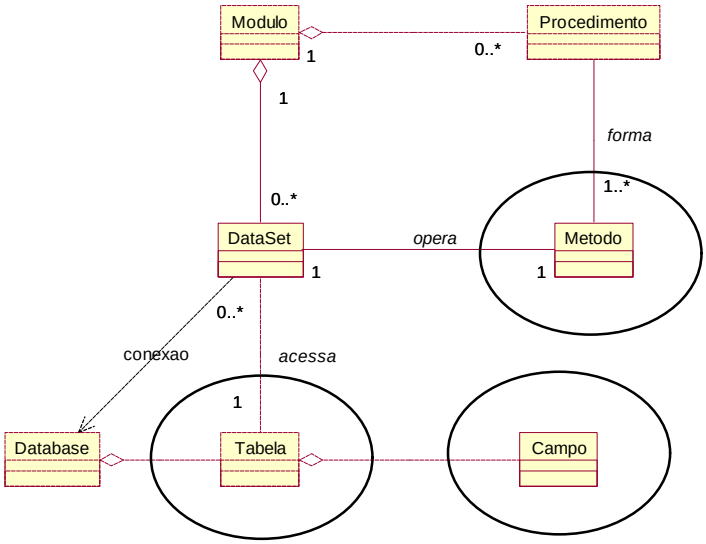

Figura 6. Metaclasses para a criação do MCS.

Definidos os possíveis candidatos de classes, atributos e métodos no MCS, o passo seguinte é depurar o modelo conceitual para evitar inconsistências e torná-lo o mais coerente possível com o paradigma OO. A seguir, as diretrizes mais importantes são apresentadas para alcançar o refinamento desejado, algumas delas baseadas em heurísticas de análise e projeto OO, como em [6, 19].

a) Remover atributos de ligação de relacionamentos. Alguns atributos de classe representam chaves estrangeiras para associações entre as classes, que surgem dos relacionamentos entre as tabelas. Em um modelo OO estes atributos não são necessários quando o objetivo é fazer o relacionamento entre tabelas.

b) Transformar classes em relacionamentos. Algumas tabelas do modelo ER (entidade-relacionamento) são transformadas em relacionamentos no MCS. Como regra, tabelas (classes) que não possuam atributos (apenas atributos de ligação/chave), ou que possuam apenas um atributo fraco, podem ser transformadas em relacionamentos. Considera-se como atributo fraco àquele que não identifica unicamente o objeto da classe, podendo ser repetido em diversos objetos. Tais classes são tipicamente tabelas do modelo ER com duas chaves estrangeiras e nenhum ou apenas um atributo. Outra característica marcante dessas classes é que cada uma das chaves estrangeiras faz a ligação da classe com outra classe por intermédio de um relacionamento com cardinalidade "*muitos para um*". No caso de haver um atributo na classe que está sendo eliminada, o relacionamento deverá carregá-lo.

c) Identificar agregações e especializações de classe. Agregações podem ser criadas sempre que uma classe englobar outras. Especializações podem ser criadas para isolar em uma superclasse os atributos e métodos comuns a diversas subclasses.

d) Converter para nomes mais significativos. Os nomes dos atributos, métodos e classes do MCS devem ser modificados para nomes mais significativos, tornando os conceitos mais claros possíveis, de acordo com o domínio de aplicação.

e) Refinar os métodos. Redefinir e desmembrar os métodos quando o método realizar mais de uma funcionalidade na mesma classe. Mesmo que na fase de identificação de blocos nos procedimentos tenha-se a preocupação com este critério, nada impede que ainda existam métodos com baixa coesão.

f) Eliminar/unir métodos duplicados. É possível que vários métodos estejam duplicados em mais de um módulo e procedimento. Deve ser observado se os nomes e os objetivos são os mesmos, assim como parâmetros de entrada e valor de retorno do método. Se ocorrer de métodos terem a mesma funcionalidade e nomes diferentes, opta-se por um nome mais genérico que caracterize a função que o método executa.

4.2 Mapeamento das interfaces WIMP para interfaces Web

A engenharia reversa de interface tipo WIMP inclui três atividades principais:

a) identificação dos objetos gráficos de telas e as dependências com os atributos das classes de negócio;

b) identificação das informações ou mensagens (do computador ao usuário e do usuário ao computador) envolvidas na execução de cada tarefa; e

c) identificação da seqüência de mensagens necessárias para cumprir a tarefa.

Para realizar essas atividades, é necessário estudar o funcionamento da interface existente, isto é, deve-se observar como o sistema interage com o usuário. Existem problemas como reconhecer uma tela única aparecendo várias vezes com informações diferentes, ou identificar as informações que permitem passar de uma tela para a tela seguinte. Para isto, será necessário analisar não só a forma de interação do sistema, mas também o código fonte.

Os objetos visuais, presentes em janelas gráficas, devem ser mapeados para as correspondentes formas apresentadas na plataforma Web. Como é decisão de projeto de migração adotar a arquitetura *cliente-magro* [7], todos os recursos adotados seguirão aqueles definidos no padrão HTML. Dependendo da estratégia a ser adotada, alguns controles visuais usados no ambiente WIMP podem ser mapeados diretamente para HTML, como os *text-box*, por exemplo. Já outros controles WIMP esta associação não é direta. Neste caso, a adoção de padrões de projetos (*Design Patterns*) de interface [12, 22] é uma abordagem que pode ser utilizada. Um exemplo de mapeamento indireto pode ser demonstrado quando da presença de um componente *Tab* (vários *frames* sobrepostos, como "*folhas de páginas*" de um fichário) dentro de um *container* formulário. Neste caso, pode-se mapear para um padrão de projeto do tipo "*Multi-Página*", em que cada página HTML representa uma "folha" do fichário, e *links* posicionados acima da página fariam o papel das abas de cada folha. Este exemplo pode ser visto na Figura 7.

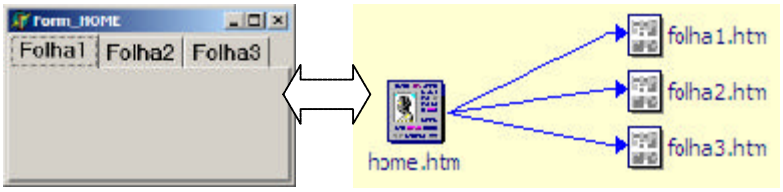

Figura 7. Mapeamento de um controle WIMP em HTML, por meio de um *Design Pattern* de interface de apresentação ("Multi-Página").

No âmbito da abordagem da engenharia reversa, os componentes WIMP, presentes no sistema origem, podem ser extraídos da definição dos *containers*, como os formulários e *frames*. Os componentes relevantes a serem armazenados no repositório são os controles visuais associados aos dados (campos) dos DataSets. Estas associações, ou vínculos de acoplamento de dados são fundamentais para o mapeamento dos atributos das classes conceituais do sistema. Os componentes visuais são representados no esquema do metamodelo por duas metaclasses, *Container* e *Controle*, e outra metaclasse (a *WidGet*) que identifica todos os tipos de controles visuais estabelecidos para o ambiente WIMP, como é observado na Figura 8. Mais detalhes sobre esse tópico encontra-se em [15].

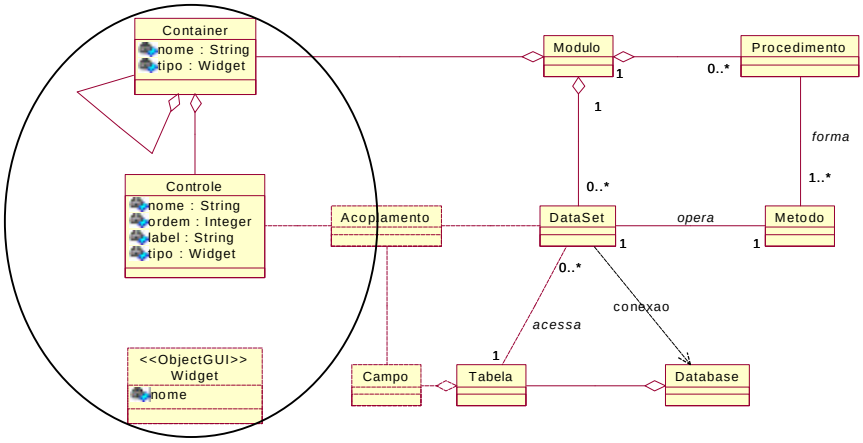

Figura 8. Definição das metaclasses de interface WIMP.

4.3 Construção dos casos de uso de migração

Os aspectos dinâmicos do sistema são obtidos pelo levantamento dos casos de uso e cenários de execução dos módulos. Todavia, aplicar automaticamente a engenharia reversa a um diagrama de caso de uso é uma tarefa difícil [2], pois pode haver perda de informações quando se passa da especificação do comportamento de um elemento para o modo como ele é implementado. Quando a documentação do sistema é precária, a alternativa é realizar uma inspeção, com ajuda de um especialista, das funcionalidades que se pretende colocar na forma de um diagrama de caso de uso.

As diretrizes gerais para realizar a engenharia reversa de um diagrama de caso de uso, segundo os autores da UML [2], são as seguintes:

a) Identificar cada ator que interage com o sistema.

b) Para cada ator, considerar a maneira como esse ator interage com o sistema, como altera o estado do sistema ou seu ambiente, ou responde a algum evento.

c) Traçar o fluxo de eventos do sistema executável relativo a cada ator. Iniciar com os fluxos primários e somente depois considerar os caminhos alternativos.

d) Agrupar os fluxos relacionados, declarando um caso de uso correspondente. Considerar a modelagem de variantes, usando relacionamentos do tipo estendido e considerar a modelagem de fluxo comum pela aplicação de relacionamentos de inclusão.

e) Representar esses atores e casos de uso em um diagrama de caso de uso e estabelecer seus relacionamentos.

No contexto da metodologia proposta, a interação entre um módulo e os atores/agentes através da execução dinâmica da aplicação origem deve ser o ponto de partida para o mapeamento das mensagens trocadas entre ambos. Cada funcionalidade do módulo (caso de uso candidato) deve ser instanciada (executada) para formar cenários reais, representados por Diagramas de Seqüência gerais, considerando os conceitos (componentes do sistema) mais relevantes para o entendimento. No exemplo da Figura 9, observamos um caso de uso para a funcionalidade “Incluir Cliente”, apresentando o Diagrama de Seqüência com as mensagens disparadas por eventos presentes no módulo e interface tipo WIMP.

No exemplo da Figura 9 (b), as mensagens entre os objetos descritas no Diagrama de Seqüência poderiam ser obtidas através de um *trace* de execução, que fica registrado em um arquivo de “log”. Essa estratégia pode ser realizada por meio de funções especializadas que são inseridas no próprio código, por exemplo. Outra forma de fazê-lo é executando no modo de depuração (*debugging*), recurso geralmente disponibilizado por ferramentas do ambiente de programação.

a) Caso de uso: **Incluir Cliente**

Ator Atendente: **Vendedor**

1. Atendente abre janela de cadastro de cliente
2. Atendente clica no botão “Incluir”
3. Atendente informa os dados do cliente: código, nome, endereço, etc.
4. Atendente clica no botão “Salvar”
5. Atendente clica no botão “Sair”

b) Diagrama de Seqüência

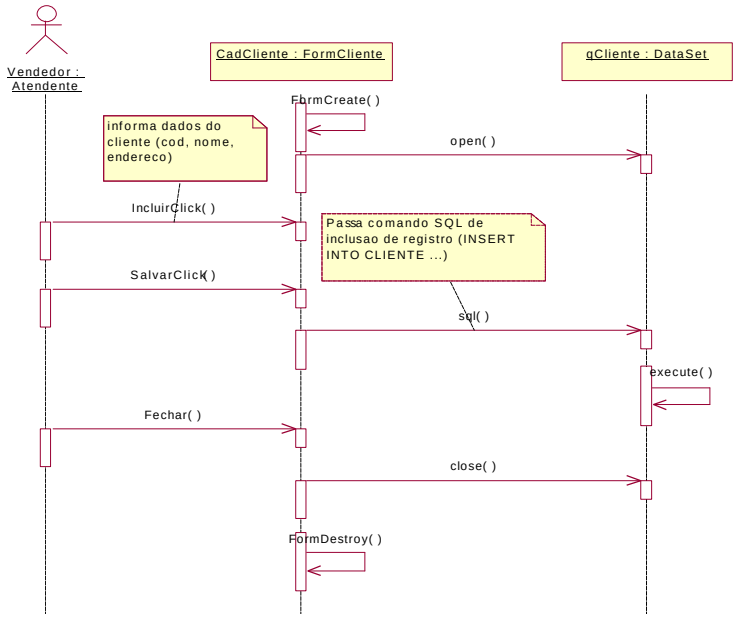

Figura 9. Exemplo de um caso de uso (a) e seu Diagrama de Seqüência (b).

O mapeamento entre as mensagens disparadas por eventos e o MCS, é o seguinte: cada mensagem representa um procedimento definido no módulo. Entretanto, a identificação de qual método é acionado na metaclasse do módulo é dificultada, pois este é um conceito obtido da quebra do procedimento em pedaços menores (os possíveis candidatos a métodos das classes). Uma solução é a reestruturação do código fonte original, reescrevendo os procedimentos com anomalias, *quebrando-os* em blocos de procedimentos menores, que corresponderão aos futuros métodos.

5 Conclusão

Este artigo apresentou uma proposta de metodologia para migração de aplicações cliente/servidor, baseado em uma abordagem de metamodelos de representação. Metamodelos são úteis para representar componentes de projeto em um alto nível de abstração, importantes para a compreensão de aplicações legadas, onde a documentação quase sempre é inexistente ou desatualizada.

O processo de migração é descrito para uma categoria de aplicações cliente/servidor, com interfaces WIMP, codificados com base em procedimentos/eventos que manipulam objetos de acesso em banco de dados relacionais. Ao final do processo, uma estrutura de metaclasses constitui metamodelos de representação do sistema, capaz de fornecer artefatos para a construção de modelos do projeto de migração. Entre os modelos temos o MCS, Modelo Conceitual do Sistema, com as classes de domínio da aplicação. Outrossim, são demonstradas formas de recuperação e mapeamento de artefatos de interface do usuário e das funcionalidades do sistema, em forma de casos de uso e diagramas de seqüência.

Embora a abordagem não esteja totalmente validada, a utilização da metodologia em mais estudos de casos reais permitirá aprimorar e detalhar melhor alguns passos do processo. Além disso, a construção de uma ferramenta está sendo finalizada e visa demonstrar a viabilidade da abordagem de metamodelos armazenados em repositório, com a possibilidade prática de gerar automaticamente modelos UML no formato XMI. Pretende-se divulgar mais resultados em futuras publicações.

Referencias

[1] Benedusi, P. Improving reverse engineering models with test-case related knowledge. *Information and Software Tecnology*, V. 38, p. 711-718, 1996.

[2] Booch, G., Jacobson, I., e Rumbaugh, R. *Unified Modeling Language*. Rational Software Corporation. Janeiro 1997. Version 1.0.

[3] Braga, R., e Masiero, P. *Padrões de Software a partir de Engenharia Reversa de Sistemas Legados*. Dissertação de Mestrado – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo. São Carlos. 1998.

[4] CDIF Technical Committee. CDIF framework for modeling and extensibility. *Technical Report EIA/IS-107*, Electronic Industries Association, Janeiro 1994. Disponível em http://www.cdif.org/. (10/06/2002).

[5] Chikofsky, E., e Cross II, J. Reverse engineering and design recovery : A taxonomy. *IEEE Software,Vol. 7*, No. 1, p 13-17. 1990.

[6] Coad, P., e Yourdon, E. *Object-Oriented Analysis*. Prentice Hall, Englewood Cliffs, NJ, 2a. edição, 1991.

[7] Conallem, J. *Building Web Applications with UML*. Addison Wesley. 2000.

[8] Ducasse, S., e Demeyer, S. *The FAMOOS Object-Oriented Reengineering Handbook*. University of Berne, Outubro 1999. Disponível em http://www.iam.unibe.ch/~famoos/handbook. (10/06/2002).

[9] Finnigan, P., Holt, R., Kalas, I. , Kerr, S., Kontogiannis, K., Mueller, H., Mylopoulos, J., Perelgut, S., Stanley, M., e Wong, K. The software bookshelf. *IBM Systems Journal, 36*(4), p.564-593, Novembro 1997.

[10] Fong, J., e Huang, S. *Information Systems Reengineering*. Spring-Verlag. Singapura. 1997.

[11] Gall, H., Klosch, R., e Mittermeir, R. Object-Oriented Re-Architecturing. Em *Proceedings of the Fifth European Software Engineering Conference* (ESEC '95), p. 499-519. Notas em Computer Science, SpringerVerlag, setembro 1995.

[12] Garrido, A., Rossi, G., e Schwabe, D. Pattern Systems for Hypermedia. *In Proceedings of PloP'97*, Pattern Language of Programming, 1997.

[13] Jarzabek, S., e Wang, G. Model-based Design of Reverse Engineering Tools. *Journal of Software Maintenance: Research and Practice*, n. 10, 1998, John Wiley & Sons, p. 353-380.

[14] Koschke, R., Girard, J-F., e Würthner, M. An Intermediate Representation for Reverse Engineering Analyses.

Proceedings of the Working Conference on Reverse Engineering - WCRE'98, 1998.

[15] Martins, C., Pimenta, M., e Price, A. Mapeamento de Interfaces WIMP para Interfaces Web. *IHC 2002 - 5th Symposium on Human Factors in Computer Systems*. Fortaleza, Brasil. Outubro, 2002, p. 380-383.

[16] McCabe, T. A Complexity Measure. *IEEE Transactions on Software Engineering, SE-2, n. 4*, p. 308-320. Dezembro, 1976.

[17] Object Management Group. Unified Modeling Language (UML) Specification. *Document formal/01-09-67 (Unified Modeling Language, v1.4)*, Object Management Group, Junho 2002. Disponível em http://www.omg.org/uml/ (10/06/2002).

[18] Object Management Group. XML Metadata Interchange (XMI). *Technical Report ad/98-10-05*, Object Management Group, Fevereiro 1998. Disponível em http://www.omg.org/uml/ (10/06/2002).

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

[20] Penteado, R., Braga, R., e Masiero,P. Improving the Quality of Legacy Code by Reverse Engineering. *In Proceedings of 4th International Conference on Information Systems, Analysis and Synthesis, ISAS'98*. Orlando, Florida, Julho 1998, p. 364-370.

[21] Rich, C., e Wills, L. Recognizing a Program's Design: A Graph-Parsing Approach. *IEEE Software*, p. 82-89, Janeiro 1990.

[22] Rossi, G., Schwabe, D., e Garrido, A. Design Reuse in Hypermedia Applications Development. *In Proceedings of Hypertext '97*, 1997.

[23] Sneed, H. Object-Oriented COBOL Recycling. *Working Conference on Reverse Engineering (WCRE)*, 3a., Monterey-CA, EUA, 1996. Anais. IEEE, p. 169-178. 1996.

[24] Terekhov, A., e Verhoef, C. The Realities of Language Conversions. *IEEE Software, 2000*. Disponível em http://adam.wins.uva.nl/~x/cnv/cnv.html (10/06/2002).

[25] Tichelaar, S., Ducasse, S., Demeyer, S., e Nierstrasz, O. A Meta-model for Language-Independent Refactoring. *Proceedings ISPSE'2000*, IEEE, 2000.

[26] Tichelaar, S., Ducasse, S., e Demeyer, S. FAMIX and XMI. *Proceedings of WCRE 2000*. Novembro 2000. Disponível em http://www.iam.unibe.ch/~tichel/publications.html (10/06/2002).

[27] Watson, A.,e McCabe, T. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. 1996. Disponível em http://hissa.ncsl.nist.gov/HHRFdata/Artifacts/ITLdoc/235/mccabe.html. (10/06/2002).

[28] Wiggerts, T. , Bosma, H., e Fielt, E. Scenarios for the identification of objects in legacy systems. *4th Working Conference on Reverse Engineering*, p. 24–32. IEEE Computer Society, 1997.

[29] Wong, K., Tilley, S., Muller, H., e Storey, M. 1995. Structural redocumentation: A case study. *IEEE Software*, 12-1, p. 46-54. Janeiro 1995.

[30] Zou, Y., e Kontogiannis, K. A Framework for Migrating Procedural Code to Object-Oriented Platforms. *In Proceedings of 8th Asia-Pacific Software Engineering Conference*, Macau SAR, China, Dezembro 2001.