UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA APLICAÇÃO DO PADRÃO DATA ACCESS OBJECT (DAO) EM SISTEMAS DESENVOLVIDOS COM A LINGUAGUEM DE PROGRAMAÇÃO DELPHI

MARCELO SARDAGNA

BLUMENAU
2007

2007/2-25

MARCELO SARDAGNA

FERRAMENTA PARA APLICAÇÃO DO PADRÃO DATA ACCESS OBJECT (DAO) EM SISTEMAS DESENVOLVIDOS COM A LINGUAGUEM DE PROGRAMAÇÃO DELPHI

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Adilson Vahldick – Orientador

BLUMENAU
2007

2007/2-25

FERRAMENTA PARA APLICAÇÃO DO PADRÃO DATA ACCESS OBJECT (DAO) EM SISTEMAS DESENVOLVIDOS COM A LINGUAGUEM DE PROGRAMAÇÃO DELPHI

Por

MARCELO SARDAGNA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Adilson Vahldick, Especialista – Orientador, FURB
Membro:	Prof. Wilson Pedro Carli, Mestre – FURB
Membro:	Prof.ª Joyce Martins, Mestre – FURB

Blumenau, 07 de dezembro de 2007

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste. Agradeço a minha família pelo apoio.

AGRADECIMENTOS

A Deus, por ter me concedido mais essa graça.

À minha família, e meus amigos que mesmo longe, sempre estiveram sempre presente me apoiando.

Ao meu orientador, Adilson Vahldick, por ter me apoiado, ajudado e acreditado na conclusão deste trabalho.

A Cooperativa de Produção e Abastecimento do Vale do Itajaí (Cooper) pelo incentivo e compreensão deste momento.

Aos meus colegas de trabalho que me apoiaram e incentivaram a chegar até o final.

A minha noiva Mikaele Justen pela compreensão e apoio nesse momento difícil, mas especial em minha vida.

Em especial aos meus pais José e Vanilda Sardagna que não mediram esforços para que eu pudesse alcançar mais este objetivo lutando e incentivando desde o inicio da faculdade.

A sorte favorece a mente mais bem preparada.

Albert Einstein

RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta que procura expressões SQL em projetos em Delphi, cria classes para agrupar essas expressões e substitui essas expressões por chamadas a métodos da classe criada. A procura é feita nos arquivos de código fonte (PAS) e nos formulários (DFM). A classe gerada segue o padrão *Data Access Object* (DAO) onde ela concentra todo o acesso ao banco de dados. Usou-se a ferramenta GALS para geração das classes de análise léxica, sintática e semântica para verificar se existem expressões SQL iguais. A ferramenta foi desenvolvida em Borland Developer Studio 2006 e consta neste trabalho um estudo de caso apresentando a utilização da mesma.

Palavras-chave:*Data access object* . Mapeamento de dados. SQL. *Design pattern*.

ABSTRACT

This work presents the development of a tool that serchs SQL expressions in Delphi project, creates classes to group these expressions and substitutes in these expressions for calls of the methods from this classes. The search is made in the archives of source code (PAS) and in forms (DFM). The generated classes follow the Data Access Object (DAO) pattern what concentrates the access to the data base. GALS was used for generation of the lexical, syntatic and semantics analysis classes. The tool was developed in Borland Developer Studio 2006 and this work there is a case study presenting the use of it.

Key-words: Data access object. Data mapping. SQL. Design pattern.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADO - *ActiveX Data Objects*

B2B - *Business to Business*

BDE - *Borland Database Engine*

BNF – *Backus-Naur Form*

DAO – *Data Access Object*

DFM – *Delphi ForM*

DPR – *Delphi PRoject*

IBX – *InterBase eXpress*

LDAP - *Lightweight Directory Access Protocol*

MER – Modelo Entidade-Relacionamento

PAS – *Delphi Source File*

RF – Requisito Funcional

RNF – Requisito Não Funcional

SQL - *Structured Query Language*

UML – *Unified Modeling Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente as empresas que desenvolvem software têm diversos pontos para preocuparem-se. Entre eles estão o conhecimento sobre o assunto pela equipe de trabalho, em qual plataforma o software rodará e também a utilização de padrões de projeto. Pode-se obter muitos benefícios na utilização de padrões como facilidade com manutenções, agilidade na análise do código, reutilização de código, adaptações e melhorias. Padrões de projetos também são conhecidos como *Design Patterns*. Segundo Gamma (2000, p. 18), "os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem sucedidas. Expressar técnicas testadas e aprovadas as torna mais acessíveis para os desenvolvedores de novos sistemas.".

Um dos problemas quando se desenvolve um sistema com banco de dados é onde colocar os comandos *Struct Query Language* (SQL). Dependendo de como o sistema for desenvolvido, os comandos SQL podem ficar espalhados por todo o código fonte, podendo assim ter vários problemas, como por exemplo, dois comandos SQL idênticos para a mesma finalidade ou pior, dois comandos SQL iguais que estão em lugares diferentes utilizados por dois formulários diferentes.

Nota-se que o problema está na distribuição das expressões SQL em toda a aplicação. Naturalmente uma solução é o agrupamento dessas expressões em uma unidade, módulo ou classe. Assim haverá um único ponto para manutenção. Essa solução é o que propõe o padrão de projeto *Data Access Object* (DAO) que segundo, Aéce (2005), é um padrão de projeto que abstrai e encapsula os mecanismos de acesso aos dados.

Neste trabalho foi desenvolvida uma ferramenta para a aplicação do padrão DAO em programas escritos na linguagem de programação Delphi. Os arquivos de entrada são os arquivos fonte da linguagem de programação Delphi onde é feita uma análise para encontrar as expressões SQL.

Quanto ao funcionamento da ferramenta, inicialmente o usuário seleciona o banco de dados do projeto a ser analisado e para cada tabela do banco de dados é gerada uma classe DAO com os métodos `Inserir`, `Alterar`, `Excluir` e `SelecionarTodos`. Logo após o mesmo seleciona a pasta onde é salva as novas classes DAO. Na seqüência o usuário escolhe o projeto a ser analisado indicando o arquivo de projeto Delphi (com extensão `dpr`). Em seguida, a ferramenta busca as expressões SQL das `units` desse projeto. Quando encontrada é então retirada da referida `unit`, e criado um método na classe DAO com essa expressão.

Após a criação do método, é feita a substituição da expressão SQL encontrada, pela chamada do método da classe DAO.

Depois de feita a análise das `units`, é analisado o arquivo fonte *Delphi ForM* (DFM). Primeiramente é necessário identificar qual o componente de acesso à base de dados que está sendo usado. Tendo descoberto o componente de acesso, é necessário listar no código fonte onde este componente está sendo utilizado e retirar a expressão SQL do componente para criar um método na classe DAO, para que em seguida seja colocada uma chamada a esse método criado onde ele é utilizado.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta que analise o código fonte na linguagem de programação Delphi e substitua as chamadas às expressões SQL por chamadas a métodos de classes DAO.

Os objetivos específicos do trabalho são:

a) analisar os arquivos *Delphi Source File* (PAS) e DFM;
b) criar uma classe DAO, com métodos relativos às expressões SQL existentes nos arquivos analisados;
c) substituir as expressões que estão no código fonte e fazer manutenção nos DFM;
d) analisar projetos com componentes de acesso `TSQLQuery`.

1.2 ESTRUTURA DO TRABALHO

O trabalho está estruturado em capítulos explicados em seguida. O segundo capítulo apresenta a fundamentação teórica do trabalho, destacando a importância de aplicar padrões em projetos de software, seguindo com uma explicação sobre o padrão DAO, análise léxica, sintática e semântica, componentes Delphi de acesso a banco de dados. Por fim, são apresentados dois trabalhos correlatos.

O terceiro capítulo apresenta os requisitos, especificação, implementação e resultados obtidos no desenvolvimento da ferramenta. São apresentados modelos de casos de uso,

diagrama de atividades, diagramas de classes, diagramas de seqüência, um exemplo de código de testes da classe DAO e a especificação da *Backus-Naur Form* (BNF) utilizada. Também são apresentadas explicações sobre o desenvolvimento da aplicação apresentada.

O quarto capítulo apresenta conclusões, limitações e sugestões para futuros trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir são explanados os principais assuntos relacionados ao desenvolvimento da ferramenta: padrão de projeto DAO; analisadores léxico, sintático e semântico; componentes de acesso a banco de dados na linguagem de programação Delphi e os trabalhos correlatos.

2.1 APLICAÇÃO DE PADRÕES

Com os padrões de projetos podem-se escolher alternativas para tornar um sistema reutilizável (GAMMA, 2000, p. 18). Segundo Fowler (2006, p. 31), a não adoção de padrões de projetos é um dos motivos pelos quais muitas ferramentas têm sido um fracasso total.

Para Shalloway e Trott (2004, p. 96), na descrição de um padrão estão envolvidos quatro itens: o nome, o propósito, como se pode realizá-lo e as restrições e os motivos que se deve considerar para realizá-lo. Conforme Shalloway e Trott (2004, p. 99), as razões mais comuns para estudar padrões devem-se ao fato que eles permitem:

a) reutilização das soluções: obtém-se um início direcionado e evita-se percalços. Beneficia-se de aprender a partir de soluções de outros, pois não se tem que reinventar soluções para problemas recorrentes;
b) estabelecimento uma terminologia comum: a comunicação e o trabalho em equipe requerem um vocabulário comum dos problemas. Os padrões fornecem um ponto de referência comum durante a fase de análise e elaboração de um projeto de software;
c) perspectiva dos problemas: os padrões fornecem uma perspectiva de mais alto nível acerca dos problemas e do processo de projeto e orientação a objetos, sem ter que lidar com os detalhes cedo demais.

Para Valente (2003, p. 3), a importância de se utilizar padrões de software pode ser citada em vários itens:

a) resolvem problemas reais, sendo soluções já comprovadas e testadas em diversos casos de um ou vários domínios;
b) capturam o conhecimento dos modeladores mais experientes, aumentando a reutilização de idéias;

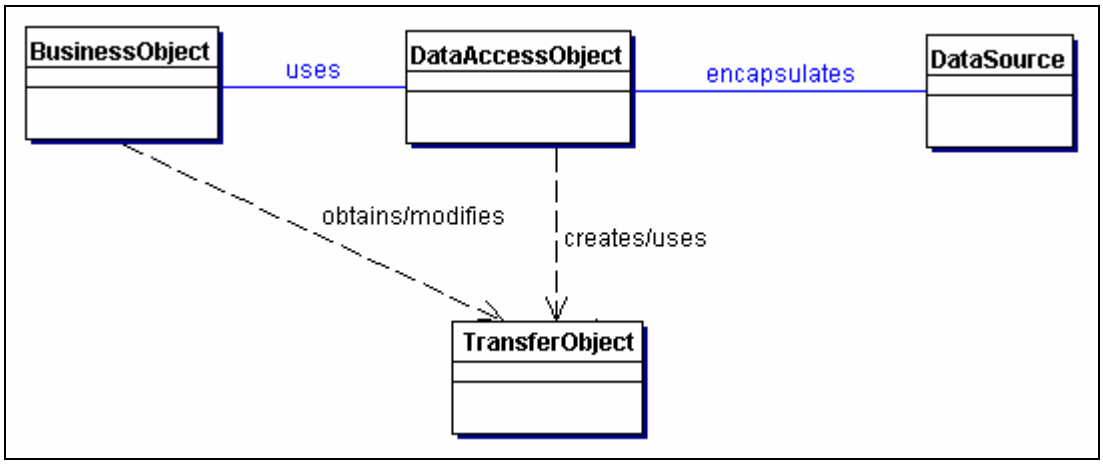
c) ajudam modeladores iniciantes a compreender e resolver problemas complexos;
d) podem ser utilizados para registrar e encorajar a reutilização das melhores práticas;
e) capturam partes essenciais do projeto de forma compacta;
f) aumentam o nível de abstração para o desenvolvimento de soluções para os problemas;
g) formam um vocabulário comum para discussão e resolução de problemas;
h) servem para melhorar a comunicação entre membros de uma equipe;
i) além da solução, indicam as condições necessárias para aplicá-la, o raciocínio lógico utilizado para concebê-la, as restrições e custos associados.

Em muitos casos, é impossível a aplicação de um padrão conforme ele é escrito. Segundo Fowler (2006, p 31), sempre que implanta-se padrões é necessário adaptar-se conforme o projeto. Observa-se a mesma solução diversas vezes, mas ela nunca é a mesma.

2.2 DATA ACCESS OBJECT

Para Aéce (2005), "DAO, é um padrão de projeto que é utilizado para abstrair o acesso a dados da camada de negócios (Business Layer), que por sua vez, acessa a Base de Dados através de Data Access Objects, tornando assim transparente o acesso". A aplicação do padrão DAO é importante, pois além de abstrair e encapsular as expressões SQL, a camada de negócio consequentemente não sabe nada a respeito do banco de dados. Segundo Aéce (2005), isso leva a uma produtividade muito grande e facilidades em manutenções futuras.

Segundo Sun (2007), o DAO esconde completamente os detalhes da execução da origem dos dados de seus clientes, conforme mostra a Figura 1.



Fonte: Sun (2007).

Figura 1 - Estrutura do padrão DAO

Conforme Sun (2007), no padrão DAO existem quatro classes:

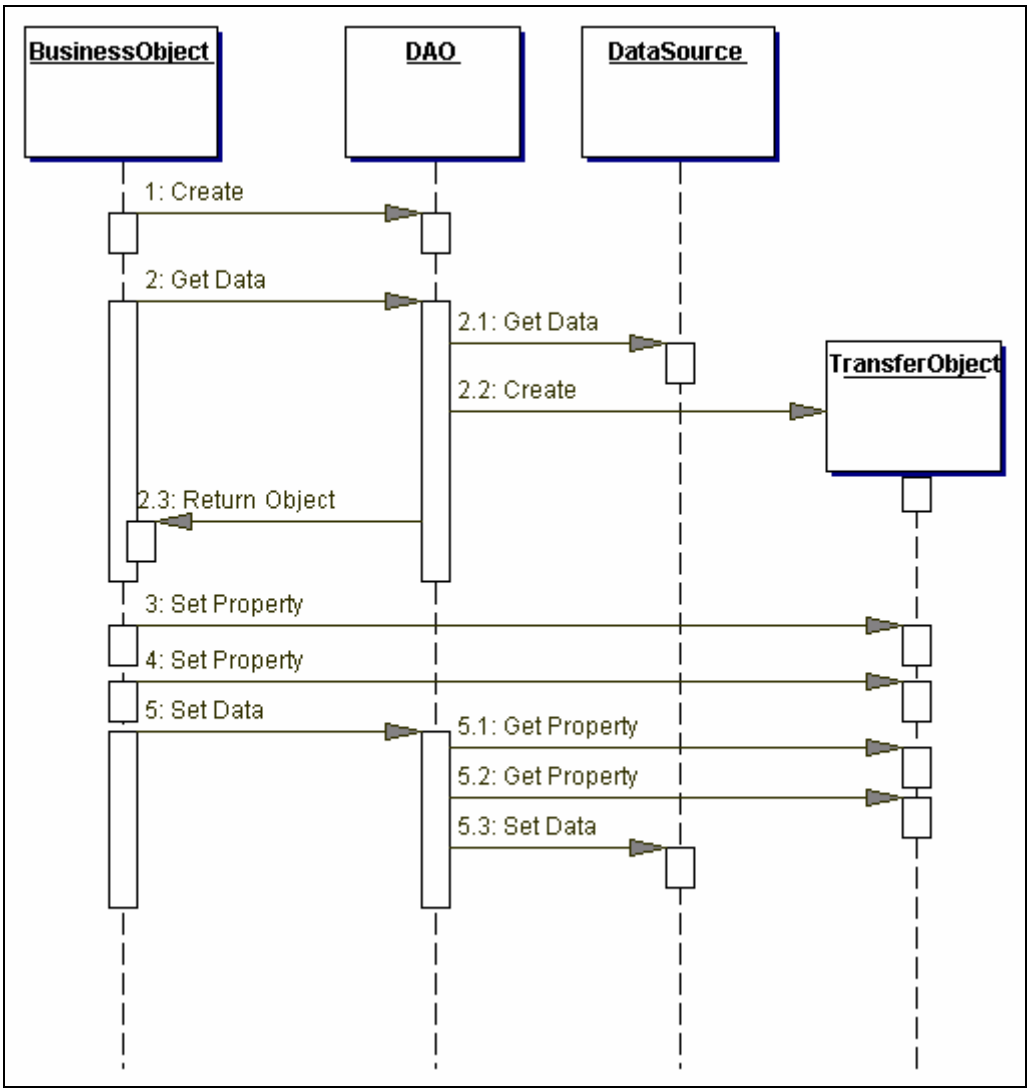
a) `BusinessObject` : representa o cliente dos dados. É o objeto que requer o acesso à origem dos dados de obter e armazenar dados. Um exemplo pode ser uma tela de cadastro que solicita a busca de uma informação ou o armazenamento de um dado na base a qual está conectada;

b) `DataAccessObject` : é o objeto principal deste padrão. Ele abstrai a execução do acesso dos dados para que o `BusinessObject` libere o acesso transparente à origem dos dados;

c) `DataSource` : representa a origem dos dados. Uma origem dos dados pode ser uma base de dados, um repositório de *eXtensible Markup Language* (XML), outro sistema (*legacy/mainframe*), um serviço (*business to business* - B2B), ou algum tipo do repositório (*Lightweight Directory Access Protocol* - LDAP);

d) `TransferObject` : representa um objeto de transferência usado como um portador de dados. O `DataAccessObject` pode usar esse objeto como retorno dos dados ao cliente. O `DataAccessObject` pode também receber os dados do cliente nesse objeto para atualizar os dados na origem dos dados.

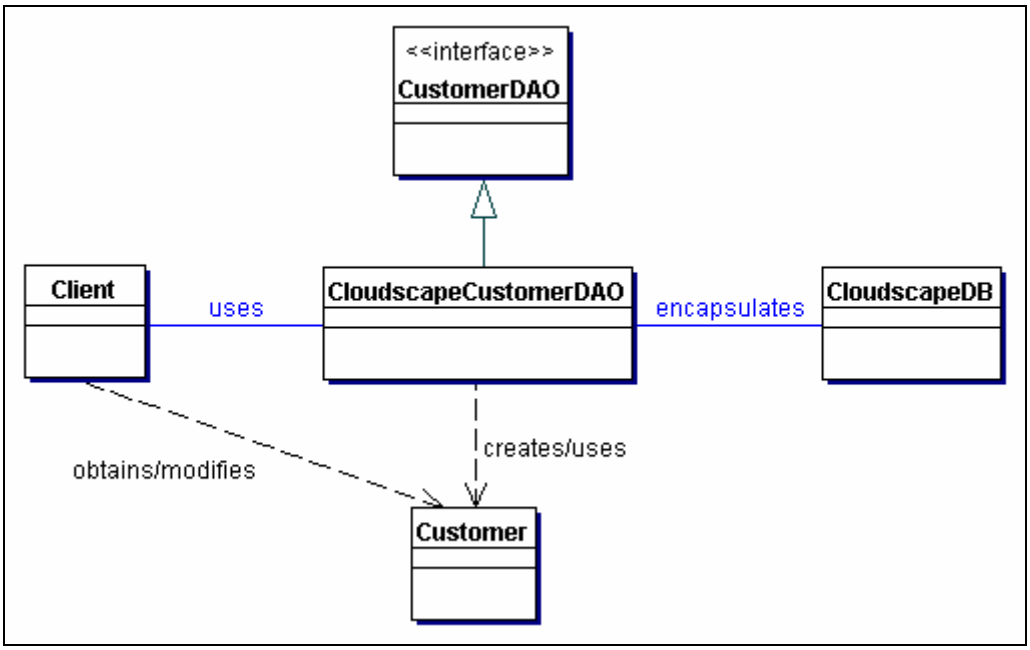
O padrão DAO tem por objetivo abstrair os dados da camada de aplicação. A Figura 2 mostra um diagrama de seqüência entre os vários participantes do padrão.



Fonte: Sun (2007).

Figura 2 - Diagrama de seqüência que mostra a execução do padrão DAO

Um diagrama de classes exemplo para um objeto persistente que represente a informação do cliente é mostrado na Figura 3. O `CloudscapeCustomerDAO` cria um objeto de transferência do cliente quando o método `findCustomer()` é invocado.



Fonte: Sun (2007).

Figura 3 - Implementação do padrão DAO

2.3 ANÁLISE LÉXICA SINTÁTICA E SEMÂNTICA

Segundo Price e Toscani (2001, p. 17), o principal objetivo da análise léxica é "fazer a leitura do programa fonte, caractere a caractere, e traduzi-lo para uma seqüência de *símbolos léxicos* também chamados de *token*". O analisador léxico identifica os *tokens* lendo caractere a caractere, verificando se os caracteres lidos pertencem ao alfabeto da linguagem. Segundo Souza et al (2002), uma das funções do analisador léxico é separar estas marcas, enviando uma cadeia de *tokens* ao analisador sintático. Ainda segundo Price e Toscani (2001, p. 24), "os analisadores léxicos são, usualmente, especificados através de notações para a descrição de linguagens regulares tais como autômatos finitos, expressões regulares ou gramáticas regulares".

O analisador sintático por sua vez tem como objetivo verificar se a estrutura gramatical do programa está correta. Para realizar esta verificação, o analisador sintático agrupa os *tokens* enviados pelo analisador léxico em estruturas sintáticas, produzindo a árvore de derivação. Louden (2004, p. 95), afirma que a análise sintática determina a estrutura de um comando ou programa. Essa estrutura é dada pela BNF da linguagem, que define as suas regras gramaticais.

Segundo Louden (2004, p. 259), “essa fase recebe o nome de análise semântica, pois requer a computação de informações que estão além da capacidade das gramáticas livres de contexto e dos algoritmos padrão de analise sintática“. Ainda segundo Louden (2004, p. 259), “a análise semântica requer a construção de uma tabela de símbolos para acompanhar o significado dos nomes estabelecidos nas declarações e efetuar inferência e verificação de tipos em expressões e declarações de forma a determinar sua correção pelas regras de tipos da linguagem”.

2.4 GERAÇÃO DE CÓDIGO

Para Herrington (2003, p. 15), a geração de código é benéfica quando se gera uma grande quantidade de fonte podendo ainda reutilizar os mesmos. Com isso a atenção fica na camada de negócios.

Para Herrington (2003, p. 93) as etapas a seguir são necessárias para o desenvolvimento de um gerador de código:

a) Construir o código de teste: para iniciar o desenvolvimento de um gerador é necessário ter um código de testes inicial para saber o que o deve-se gerar. O código gerado é comparado com o código de testes inicial para verificar se é compatível.
b) projetar o gerador: nesta etapa é necessário definir as informações de entrada como a mesma deve ser processada para enfim apresentar a saída. A análise dos requisitos sobre o código gerado nesta fase são muito importantes.
c) desenvolver o analisador de entrada: nesta etapa é desenvolvido o processamento de entrada, para que seja efetuada a extração e o armazenamento dos dados de entrada para gerar a saída.
d) desenvolver os *templates* do código de testes: criar os *templates* para a geração do código de saída;
e) verificação do código de saída: executar o código de saída e comparar com a execução do código de teste, verificando a equivalência.

2.5 COMPONENTES DELPHI DE ACESSO A BANCO DE DADOS

Atualmente pode-se acessar um banco de dados de diversas maneiras no Delphi. Pode-se acessar com os componentes nativos do Delphi como o `Borland Database Engine` (BDE), o `ActiveX Data Objects` (ADO), o *driver* nativo para acesso a bancos de dados InterBase (IBX) e o `DBExpress` a partir da versão 7. Também se pode acessar uma base de dados através de componentes desenvolvidos pelas próprias empresas de software. Uma das maneiras de como pode ser especificada uma expressão SQL é apresentada no Quadro 1.

```
procedure TFrmCadChequesRecebidos.PesquisaBanco(CodBanco: Integer);
begin
  With QueryPesquisaBanco do
    begin
      If Active then
        Close;
      SQL.Clear;
      SQL.Add('SELECT * FROM BANCO');
      SQL.Add('WHERE BA_COD = :BA_COD');
      ParamByName('BA_COD').AsInteger := CodBanco;
      Open;

      edNomBanco.Text := FieldByName('BA_NOME').AsString;
    end;
end;
```

Quadro 1 - Exemplo de código fonte Delphi com componentes de acesso `TSQLQuery`

Conforme ilustrado no Quadro 1, o desenvolvedor está adicionando a expressão SQL em tempo de execução. O componente utilizado no Quadro 1 é o `TSQLQuery` e está adicionado no formulário. Para os componentes nativos do Delphi a funcionalidade segue o mesmo princípio podendo ser adicionados os comandos em tempo de execução ou ainda ser adicionado diretamente nos componentes.

O `TSQLQuery` é um componente que pertence ao pacote `DBExpress`. A partir da versão 7.0 esse passou a ser um pacote nativo do Delphi. Conforme Borland Software Corporation (2005), o mesmo deve ser utilizado para executar comandos SQL. Este componente pode retornar os resultados de um comando *Select*, ou executar comandos de *Insert*, *Delete* e *Update*. Este componente tem diversas propriedades e métodos. A seguir são apresentadas algumas das propriedades e métodos:

a) `SQL`: após estar conectado à base de dados, deve-se utilizar a propriedade `SQL` para especificar o comando. Esta mesma propriedade pode ser utilizada em tempo de execução;

b) `Open`: caso o comando especificado seja uma seleção, deve-se executar este

método para que o componente retorne os resultados do banco de dados a que está conectado;

c) `ExecSQL`: caso a expressão SQL seja um comando para inserir, alterar ou excluir, o método chamado deve ser o `ExecSQL`. Este método insere um registro no banco de dados a que está conectado;

d) `Add`: o método `add` pode ser utilizado para atribuir expressões SQL em tempo de execução, pois o comando pode ser adicionado em partes ou depender do usuário informar algum dado de entrada conforme Quadro 2;

```
QueryAux.SQL.Add('Select * From Fornecedor');
QueryAux.SQL.Add('Where Fo_Cod = :Fo_Cod');
QueryAux.ParamByName('FO_COD').AsInteger := edCodFornec.AsInteger;
QueryAux.Open;
```

Quadro 2 - Exemplo de como adicionar comandos SQL em tempo de execução

e) `Text`: a propriedade `text` tem a mesma funcionalidade do método `Add`, podendo ser atribuídos comandos SQL em tempo de execução;

f) `ParamByName`: após especificar o comando a ser executado, poderão ser passados parâmetros ao comando especificado. Quando é especificado um comando que contenha parâmetros, o `dataset` automaticamente analisa gramaticalmente o comando e adiciona na lista de parâmetros. Assim para atribuir valores para o comando especificado, deve-se chamar o método `ParamByName` e o nome do parâmetro e então atribuir o valor ao parâmetro.

A Borland Software Corporation (2005), ainda cita que o comando que será fornecido para esta propriedade deve ser válido para o usuário do banco de dados a que o objeto de `TSQLQuery` é conectado.

2.6 TRABALHOS CORRELATOS

Algumas ferramentas desempenham papel semelhante ao proposto no presente trabalho, cada qual com suas peculiaridades. Dentre elas foram selecionadas a ferramenta desenvolvida por Grott (2003) e a ferramenta descrita em Ferreira (2003).

Grott (2003) implementou um *framework* para cálculo de impostos em projetos de automação comercial utilizando a linguagem Java 2 SDK, verificando a melhor forma de solucionar um problema recorrente através de *Design Patterns* para proporcionar um maior

índice de reutilização da solução no desenvolvimento de sistemas. As características desta ferramenta são:

a) exemplificar o uso de *patterns* e *frameworks*;
b) criar um *framework* com os *patterns* selecionados;
c) desenvolver um material que ajudará os arquitetos de sistemas a decidirem qual a melhor solução a ser tomada no desenvolvimento de um projeto.

Segundo Ferreira (2003), a ferramenta desenvolvida é capaz de documentar externamente projetos implementado na linguagem de programação Delphi. As características desta ferramenta são:

a) gerar uma biblioteca contendo todas as rotinas do sistema (*procedures* e *functions*);
b) identificar, através das análises léxica e sintática do código fonte, toda a manipulação feita com as tabelas do banco de dados (*insert, update, delete* e *select*);
c) implementar um mecanismo para que o desenvolvedor possa consultar e emitir relatórios de toda documentação gerada.

3 DESENVOLVIMENTO DA FERRAMENTA

Neste capítulo são apresentados os requisitos, a especificação e a implementação da ferramenta. Também são mostrados os cenários, os diagramas de atividades os diagramas de classe, e um exemplo de código fonte da classe DAO bem como a especificação da BNF. Para finalizar são apresentados os resultados obtidos e uma comparação com os trabalhos correlatos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Abaixo estão apresentados os principais requisitos funcionais (RF) e não-funcionais (RNF) da ferramenta:

a) a partir da análise do código fonte escrito na linguagem de programação Delphi extrair as expressões SQL e as tabelas envolvidas criando métodos nas classes DAO (Requisito Funcional - RF);
b) a partir do banco de dados do projeto gerar uma classe DAO para cada tabela do encontrada (RF);
c) substituir as expressões SQL por métodos das classes geradas (RF);
d) fazer a análise léxica, sintática e semântica das expressões SQL para armazenar dados importantes dos comandos para verificar se não existe comandos SQL repetidos (RF);
e) para a análise léxica, sintática e semântica será utilizado o analisador GALS (RNF);
f) ser implementado na linguagem de programação Borland Developer Studio 2006 (RNF);
g) ser compatível com Windows 2000 e Windows XP (RNF);
h) gerar classes para banco de dados InterBase;
i) ser compatível com funções, implementações e componentes das versões do Delphi 5.0, Delphi 6.0 e Delphi 7.0 (RNF);

3.2 ESPECIFICAÇÃO

Para a especificação da ferramenta foram feitos os diagramas pertencentes à modelagem UML, tendo como ferramenta de auxílio a aplicação Enterprise Architect. Apresenta-se nesta seção o modelo de casos de uso, o diagrama de atividades, os diagramas de classes, um exemplo de código fonte da classe DAO e a especificação da BNF utilizada.

3.2.1 Modelo de caso de uso

Para Bezerra (2002, p. 46), "um caso de uso é a especificação de uma seqüência de iterações entre um sistema e os agentes externos que utilizam esse sistema". Segundo Bezerra (2002, p. 46), "Um caso de uso deve definir o uso de uma parte da funcionalidade de um sistema, *sem revelar a estrutura e o comportamento internos desse sistema*". Os casos de usos devem ser escritos na forma narrativa das iterações que acontecem entre os elementos externos e o sistema.

Para Bezerra (2002, p. 49), os casos de usos podem ser definidos de várias maneiras. Uma delas pode ser através dos cenários. "Um *cenário* é a descrição de uma das maneiras pelas quais um caso de uso pode ser realizado". Um cenário pode ser chamado de instância de um caso de uso. Conforme Bezerra (2002, p. 50), uma coleção de cenários para um caso de uso pode ser útil na fase de testes para verificar erros na implementação do sistema, esclarecimento e no entendimento dos casos de uso dos quais eles são instanciados.

No Quadro 3 é apresentado o cenário para o único caso de uso da ferramenta.

UC01 – Aplicar Padrão DAO

Sumário: O usuário deseja aplicar o padrão DAO em um projeto de Delphi.

Ator primário: Usuário.

Precondições: Banco de dados selecionado.

Fluxo principal:

1. O usuário informa o banco de dados do projeto a ser analisado. O banco de dados deve ser o InterBase..
2. O usuário informa o login e senha do banco de dados no qual irá se conectar.
3. O usuário informa a pasta onde serão salvas as novas classes DAO.
4. A ferramenta cria uma classe DAO para cada tabela no banco de dados com os métodos Inserir, Alterar, Excluir e SelecionarTodos.
5. A ferramenta armazena os comandos SQL que foram especificados nas classes DAO.
6. A ferramenta habilita o form para que seja informado o projeto a ser aplicado o padrão.
7. O usuário informa o projeto a ser aplicado o padrão.
8. O usuário informa a pasta a serem salvos os novos arquivos de código fontes.
9. O usuário solicita à ferramenta para iniciar o processo.
10. A ferramenta analisa os arquivos DFM e PAS e retira os comandos SQL encontrados.
11. A ferramenta armazena os comandos SQL retirados dos arquivos fontes.
12. A ferramenta apresenta os comandos SQL encontrados nos fontes DFM e PAS.
13. O usuário solicita a ferramenta para proceder com a aplicação do padrão.
14. A ferramenta aplica o padrão e salva os novos arquivos fontes no local escolhido.

Fluxo alternativo: Alteração do nome dos métodos sugeridos.

a) Depois do passo 10 há possibilidade de alterar o nome dos métodos a serem criados.

Fluxo de exceção: erro léxico, sintático ou semântico.

a) No passo 9 se houver erro na análise léxica, sintática ou semântica dos comandos SQL, os mesmos são apresentados em tela, sendo que a análise continua até não mais encontra-los.

Pós-condição: Padrão aplicado com sucesso.

Quadro 3 – Descrição do caso de uso Aplicar Padrão DAO

3.2.2 Modelo de atividades

Segundo Bezerra (2002, p. 228), o diagrama de atividades é considerado um tipo especial de diagrama de estados, sendo orientado pelo fluxo de controle. Para apresentar o fluxo do processo é usado o diagrama de atividades.

O diagrama de atividades da ferramenta é demonstrado na Figura 4.

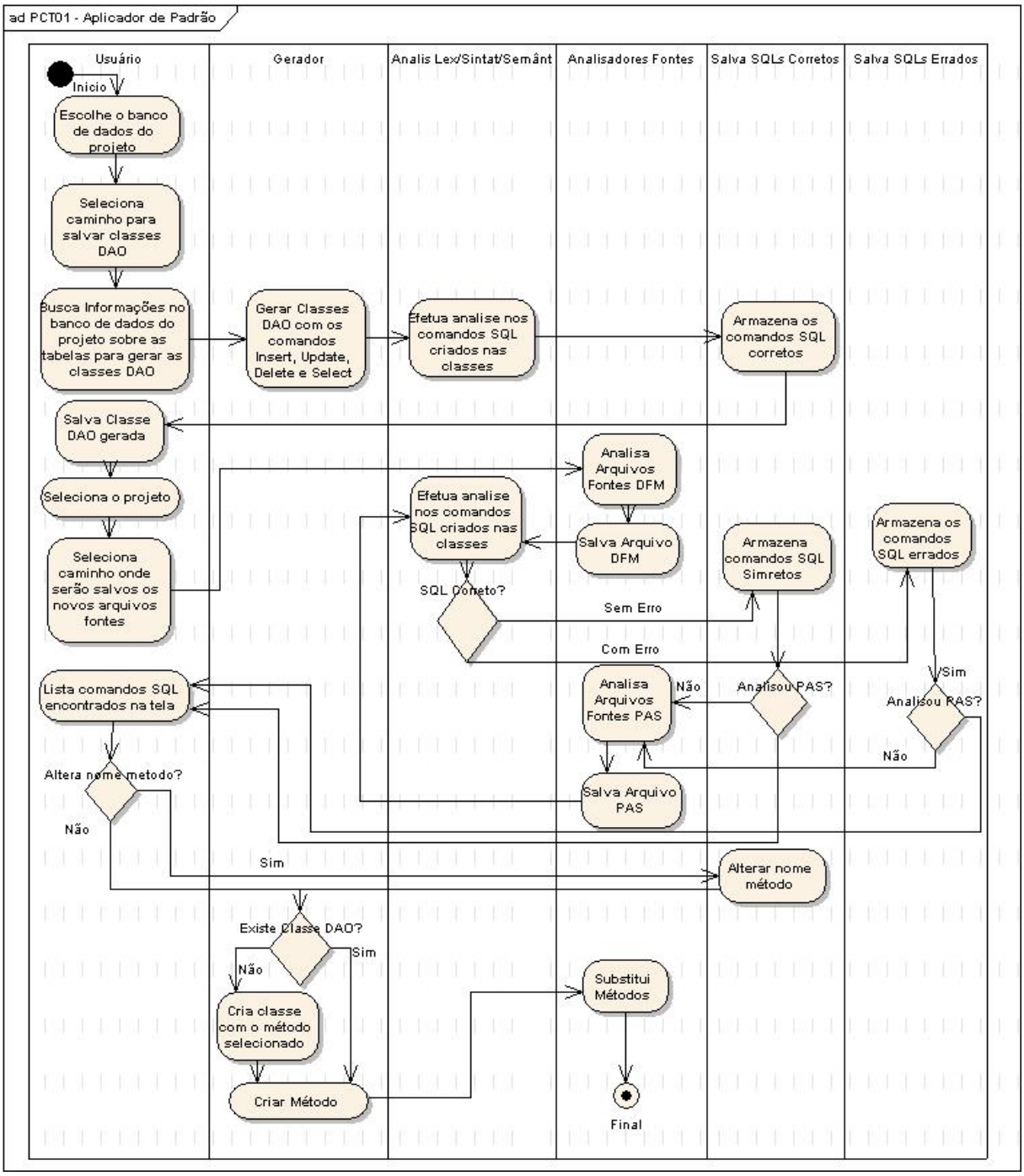


Figura 4 – Diagrama de atividades da ferramenta

Primeiramente, o usuário deve selecionar o caminho do banco de dados do projeto. Após a seleção do banco, o usuário deve indicar a pasta em que as classes DAO a serem geradas deverão ser salvas. A partir das tabelas serão geradas as classes DAO, uma para cada tabela, contendo os métodos *insert*, *update*, *delete* e *select*. Esses mesmos comandos criados nas classes DAO são analisados semanticamente para que seja possível dividir as expressões SQL em partes importantes para verificar se existem expressões SQL iguais. Depois dessa análise as expressões são adicionadas em um objeto para uso posterior.

Em seguida o usuário seleciona o projeto a ser analisado e o caminho onde irão ser

salvos os novos arquivos fontes. Assim que selecionado é iniciada a análise do código fonte dos arquivos com extensão DFM e PAS. Nesta análise são buscados todos os comandos SQL nos arquivos fontes e para cada comando encontrado é feita uma análise do mesmo. Se o comando estiver correto o mesmo é armazenado numa lista de comandos corretos para uso posterior. Se o comando for classificado como incorreto, o mesmo é armazenado em outra lista.

Feito isso, a ferramenta apresenta as classes DAO e métodos a serem gerados, junto com as expressões SQL que originaram os métodos. O usuário pode alterar o nome dos métodos que serão criados. Ao final a ferramenta faz a substituição dos comandos SQL pelos métodos das classes DAO.

3.2.3 Diagrama de classes

Para Bezerra (2002, p. 95), "o *aspecto estrutural estático* de uma cooperação permite compreender como o sistema está estruturado internamente para que as funcionalidades externamente visíveis sejam produzidas". Segundo Bezerra (2002, p. 95), "também é dito *estrutural* porque a estrutura das classes de objetos e as relações entre elas são apresentadas".

A Figura 5, apresenta o diagrama com as classes envolvidas no processo responsável por representar uma classe DAO. Por sua vez, ela gerencia especializações da classe `TMetodo`. Essas classes têm a responsabilidade de manter dados sobre os comandos SQL para que adiante sejam utilizadas para fazer a substituição dos comandos pelos métodos criados.

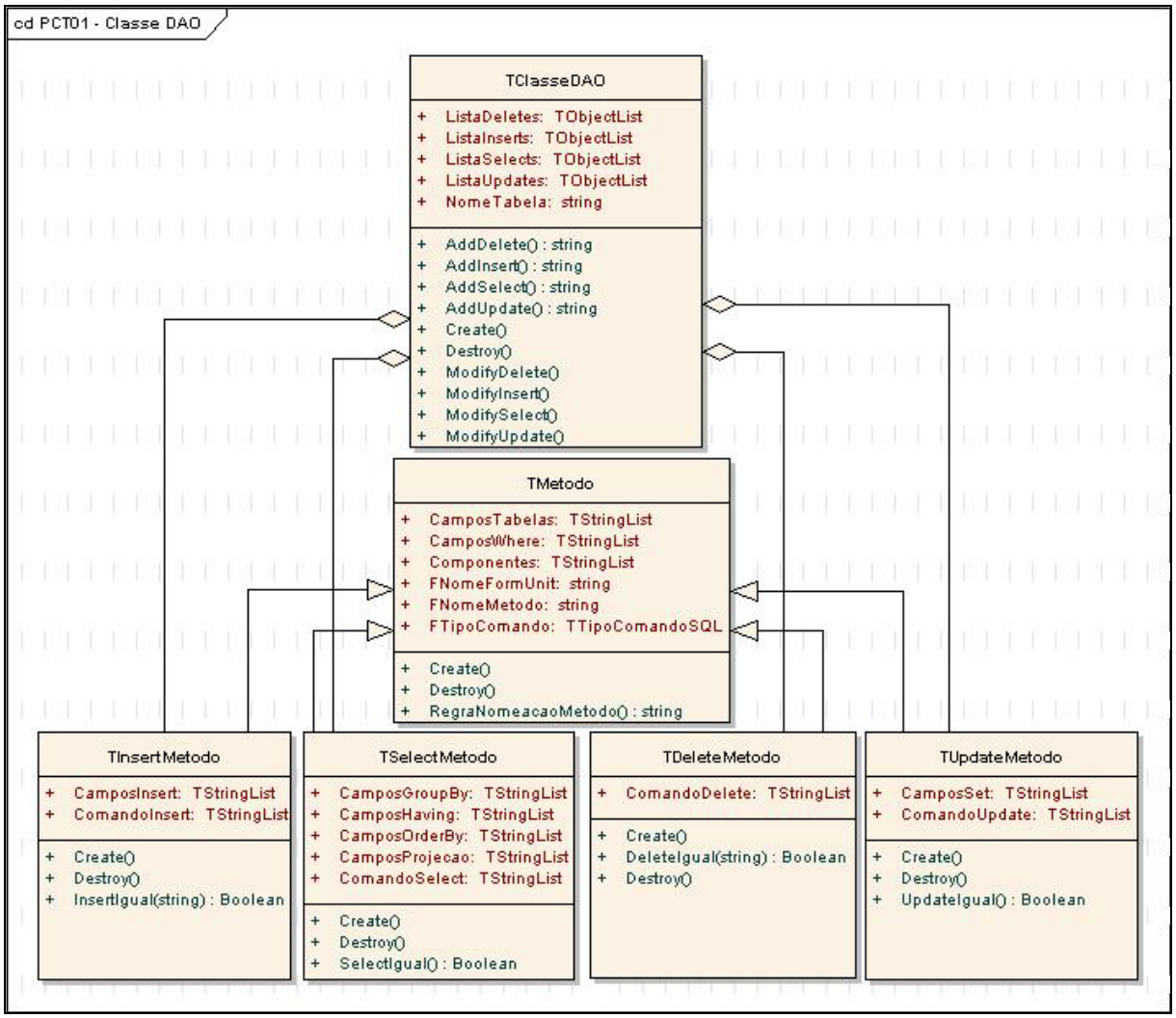


Figura 5 – Diagrama de classes para representar uma classe DAO

A Figura 6, apresenta o diagrama com as classes envolvidas no processo responsável por representar uma classe DAO com comandos SQL incorretos. Por sua vez, ela gerencia especializações da classe `TMetodoError`. Essas classes têm a responsabilidade de manter dados dos comandos SQL errados, a tabela envolvida no comando, o nome do arquivo fonte no qual está contido e o tipo do comando para que adiante seja utilizado para apresentar os comandos incorretos aos usuários. A classe `TClasseDAOError` foi criada com o intuito inicial de armazenar comando que por ventura possam estar errados e para que em futuras extensões da ferramenta possa ser possível fazer esses comandos se tornarem corretos.

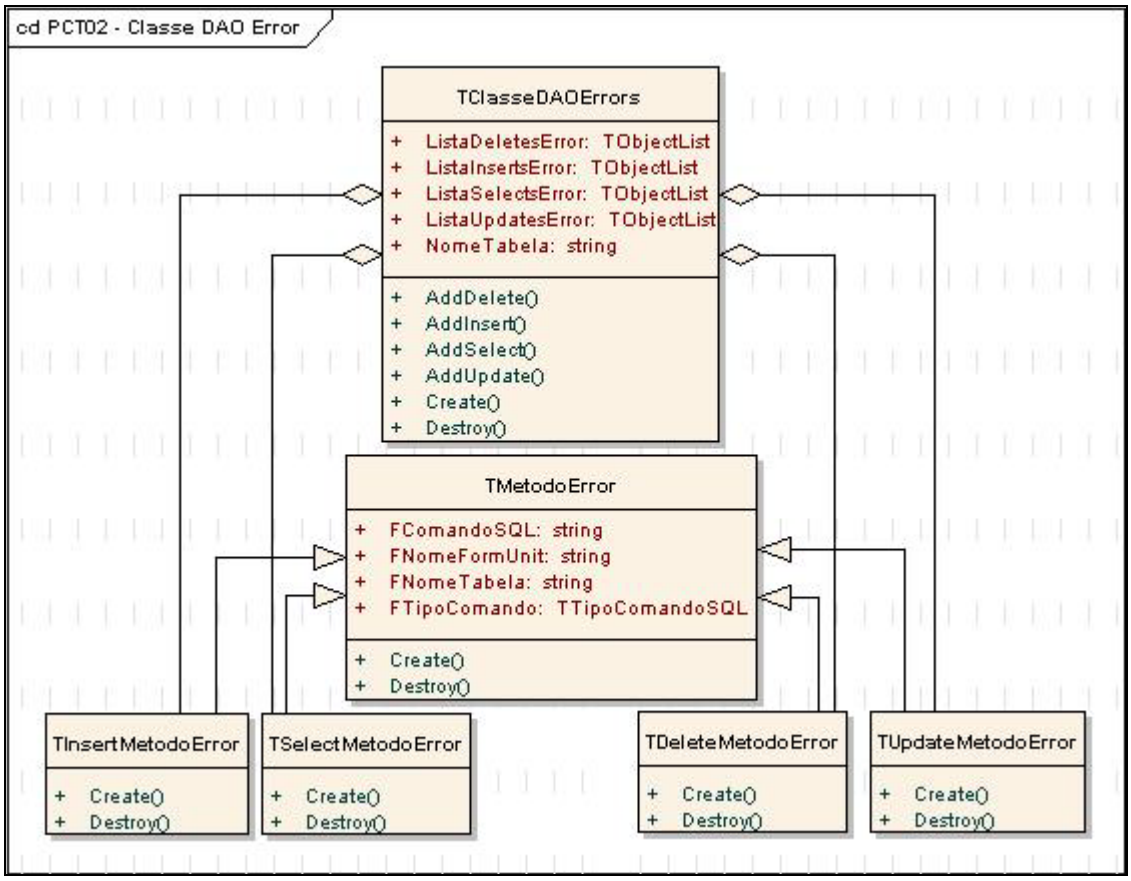


Figura 6 - Diagrama de classes para representar uma classe DAO de erro

A Figura 7, apresenta o diagrama da classe `TAnalisador` que tem por finalidade analisar os arquivos fontes DFM e PAS e extrair os comandos SQL neles encontrados. Já a classe `TCreateClasse` tem por objetivo criar as classes e os métodos DAO.

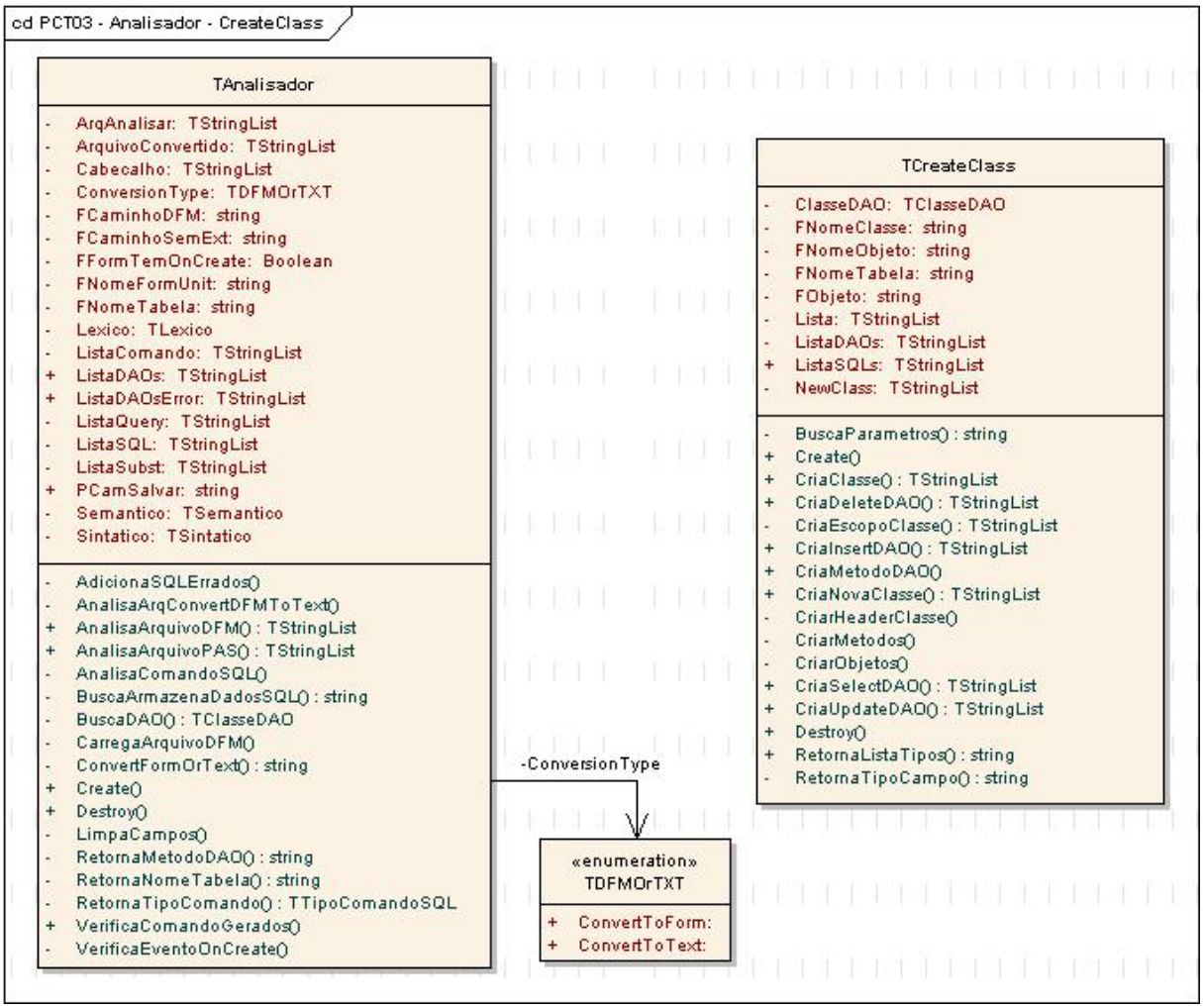


Figura 7 – Diagrama das classes `TAnalisador` e `TCreateClass`

3.2.4 Geração de código

Com a finalidade de direcionar a implementação de geração de código das classes DAO a mesma é baseada na estrutura das tabelas e no conteúdo do DFM e PAS. Na Quadro 4 é ilustrado uma tabela do banco de dados e na Figura 8 um trecho de código fonte do arquivo DFM e PAS que foram base para gerar a classe.

```
SET SQL DIALECT 3;

SET NAMES NONE;

/****                              Tables                                  ****/

CREATE TABLE FORNECEDOR (
    FO_COD          INTEGER NOT NULL,
    FO_DSC          VARCHAR(100) NOT NULL,
    FO_DSC_RED      VARCHAR(30),
    FO_CNPJ         VARCHAR(20) NOT NULL,
    FO_INSCR_EST    VARCHAR(15) NOT NULL,
    FO_RUA          VARCHAR(50),
    FO_CEP          VARCHAR(30),
    FO_CIDADE       VARCHAR(50),
    FO_BAIRRO       VARCHAR(30),
    FO_UF           VARCHAR(2)
);

/****                            Primary Keys                              ****/

ALTER TABLE FORNECEDOR ADD CONSTRAINT PK_FORNECEDOR PRIMARY KEY (FO_COD);
```

Quadro 4 - Exemplo de uma tabela em um banco de dados

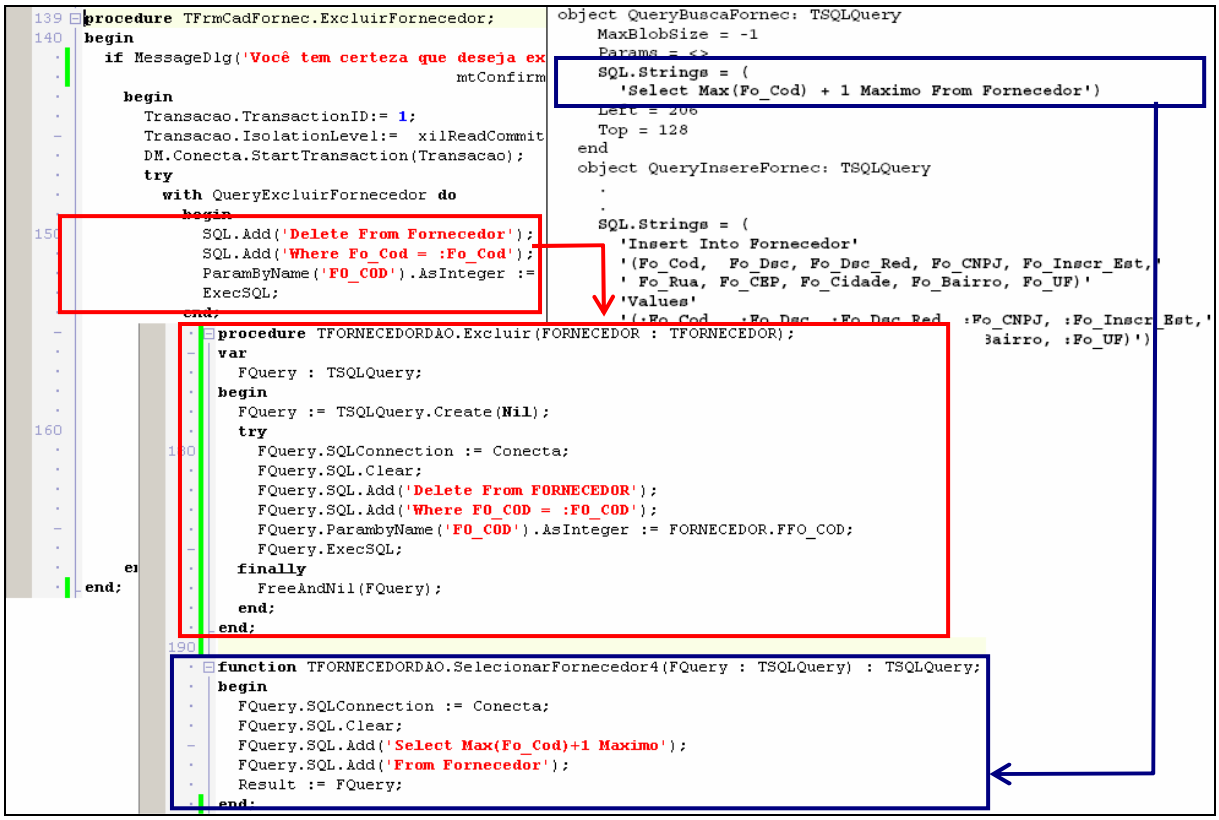


Figura 8 – Comandos SQL extraídos dos arquivos fontes PAS e DFM

Cada tabela no banco de dados é mapeada para uma classe DAO. Para cada tabela buscam-se os tipos das colunas e as chaves primárias. Os tipos das colunas são utilizados para definir os tipos dos atributos das classes DAO. O tipo `String` será usado quando a ferramenta não conseguir traduzir algum tipo. Cada classe DAO é acompanhada de uma classe de modelo. Por exemplo, a classe `TFornecedorDAO` do Quadro 6 tem relação com a classe `TFornecedor` como pode ser visto nas linhas 6 a 25, para que seja utilizado para inserir os

valores para serem inseridos, alterados e excluídos, sendo que este objeto de modelo é passado como parâmetro nos métodos DAO.

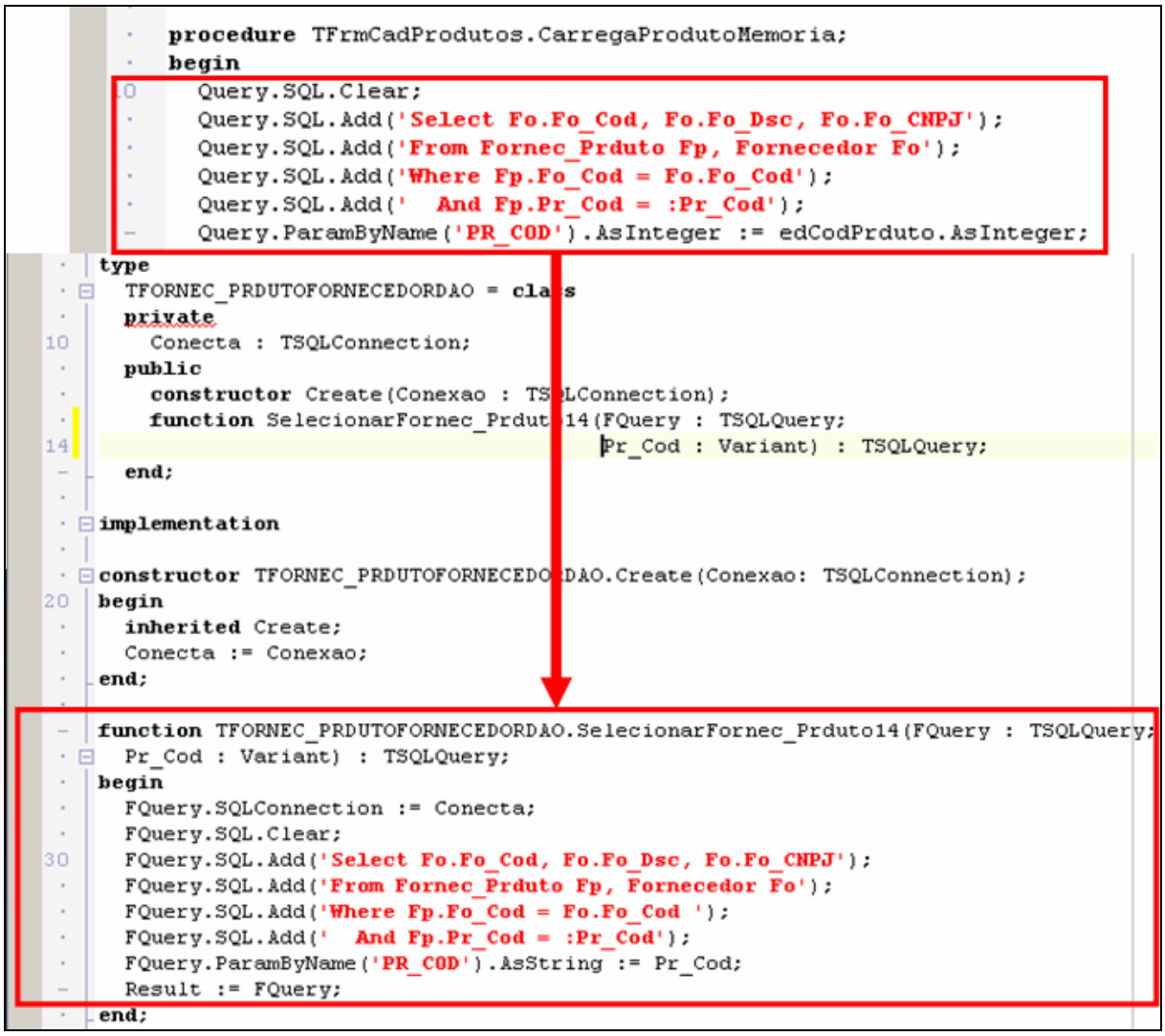
No Quadro 6 entre as linhas 45 a 69 há um exemplo de um método `Inserir` que foi gerado pela ferramenta com base na tabela do banco no qual é passado uma classe de modelo como parâmetro, com os valores a serem atribuídos ao comando *insert*.

O método `Selecionar` pode alternar muito, pois os campos de restrições diferem de um comando para outro. Por esse motivo os campos de restrições das expressões SQL são passados diretamente como parâmetro não tendo uma classe de modelo para atribuir os valores ao comando.

Nas situações em que é encontrado um `select` com mais de uma tabela, a classe DAO é nomeada de acordo com as tabelas envolvidas no comando e a classe de modelo não é criada, pois a classe DAO terá apenas comandos de seleção. No Quadro 5 é demonstrado um exemplo de um comando select com uma tabela somente e na Figura 9 é apresentado um código fonte de exemplo de um comando `select` coma mais de uma tabela.

```
function TFORNECEDORDAO.SelecionarFornecedor2(FQuery : TSQLQuery;
  Fo_Cod : Variant) : TSQLQuery;
begin
  FQuery.SQLConnection := Conecta;
  FQuery.SQL.Clear;
  FQuery.SQL.Add('Select *');
  FQuery.SQL.Add('From Fornecedor');
  FQuery.SQL.Add('Where Fo_Cod= :Fo_Cod');
  FQuery.ParamByName('FO_COD').AsString := Fo_Cod;
  Result := FQuery;
end;
```

Quadro 5 - Método selecionar com somente uma tabela



```
procedure TFrmCadProdutos.CarregaProdutoMemoria;
begin
  Query.SQL.Clear;
  Query.SQL.Add('Select Fo.Fo_Cod, Fo.Fo_Dsc, Fo.Fo_CNPJ');
  Query.SQL.Add('From Fornec_Prduto Fp, Fornecedor Fo');
  Query.SQL.Add('Where Fp.Fo_Cod = Fo.Fo_Cod');
  Query.SQL.Add('  And Fp.Pr_Cod = :Pr_Cod');
  Query.ParamByName('PR_COD').AsInteger := edCodPrduto.AsInteger;
type
  TFORNEC_PRDUTOFORNECEDORDAO = class
  private
    Conecta : TSQLConnection;
  public
    constructor Create(Conexao : TSQLConnection);
    function SelecionarFornec_Prduto14(FQuery : TSQLQuery;
                                       Pr_Cod : Variant) : TSQLQuery;
  end;

implementation

constructor TFORNEC_PRDUTOFORNECEDORDAO.Create(Conexao: TSQLConnection);
begin
  inherited Create;
  Conecta := Conexao;
end;

function TFORNEC_PRDUTOFORNECEDORDAO.SelecionarFornec_Prduto14(FQuery : TSQLQuery;
  Pr_Cod : Variant) : TSQLQuery;
begin
  FQuery.SQLConnection := Conecta;
  FQuery.SQL.Clear;
  FQuery.SQL.Add('Select Fo.Fo_Cod, Fo.Fo_Dsc, Fo.Fo_CNPJ');
  FQuery.SQL.Add('From Fornec_Prduto Fp, Fornecedor Fo');
  FQuery.SQL.Add('Where Fp.Fo_Cod = Fo.Fo_Cod ');
  FQuery.SQL.Add('  And Fp.Pr_Cod = :Pr_Cod');
  FQuery.ParamByName('PR_COD').AsString := Pr_Cod;
  Result := FQuery;
end;
```

Figura 9 - Exemplo de um select com mais de uma tabela

```
unit FORNECEDORDAO;
interface
uses
 Windows, SysUtils, Variants, Classes, Controls, FMTBcd, DB, SqlExpr;
type
 TFORNECEDOR = class
 private
  FFO_COD : Integer;
   FFO_DSC : String;
   FFO_DSC_RED : String;
   FFO_CNPJ : String;
   FFO_INSCR_EST : String;
   FFO_RUA : String;
   FFO_CEP : String;
   FFO_CIDADE : String;
 public
   property FO_COD : Integer Read FFO_COD Write FFO_COD ;
   property FO_DSC : String Read FFO_DSC Write FFO_DSC ;
   property FO_DSC_RED : String Read FFO_DSC_RED Write FFO_DSC_RED ;
   property FO_CNPJ : String Read FFO_CNPJ Write FFO_CNPJ ;
   property FO_INSCR_EST : String Read FFO_INSCR_EST Write FFO_INSCR_EST ;
   property FO_RUA : String Read FFO_RUA Write FFO_RUA ;
   property FO_CEP : String Read FFO_CEP Write FFO_CEP ;
   property FO_CIDADE : String Read FFO_CIDADE Write FFO_CIDADE ;
   property FO_BAIRRO : String Read FFO_BAIRRO Write FFO_BAIRRO ;
   property FO_UF : String Read FFO_UF Write FFO_UF ;
 end;
 TFORNECEDORDAO = class
 private
   Conecta : TSQLConnection;
 public
   constructor Create(Conexao : TSQLConnection);
   procedure Inserir(FORNECEDOR : TFORNECEDOR);
   procedure Alterar(FORNECEDOR : TFORNECEDOR);
   procedure Excluir(FORNECEDOR : TFORNECEDOR);
   function ListarTodos(FQuery : TSQLQuery) : TSQLQuery;
   procedure AlterarFornecedor1(Fornecedor : TFORNECEDOR);
   function SelecionarFornecedor2(Fo_Cod : String) : TSQLQuery;
   function SelecionarFornecedor4() : TSQLQuery;
 end;
implementation
constructor TFORNECEDORDAO.Create(Conexao: TSQLConnection);
begin
  inherited Create;
 Conecta := Conexao;
end;
procedure TFORNECEDORDAO.Inserir(FORNECEDOR : TFORNECEDOR);
var
  FQuery : TSQLQuery;
begin
 FQuery := TSQLQuery.Create(Nil);
 try
   FQuery.SQLConnection := Conecta;
   FQuery.SQL.Clear;
   FQuery.SQL.Add('Insert Into FORNECEDOR');
   FQuery.SQL.Add('(FO_COD, FO_DSC, FO_DSC_RED, FO_CNPJ, FO_INSCR_EST, FO_RUA, FO_CEP, FO_CIDADE, FO_BAIRRO, FO_UF');
   FQuery.SQL.Add('Values');
   FQuery.SQL.Add('(:FO_COD,:FO_DSC,:FO_DSC_RED, :FO_CNPJ, :FO_INSCR_EST,:FO_RUA,:FO_CEP,:FO_CIDADE,:FO_BAIRRO,:FO_UF ');
   FQuery.ParambyName('FO_COD').AsInteger := FORNECEDOR.FFO_COD;
   FQuery.ParambyName('FO_DSC').AsString := FORNECEDOR.FFO_DSC;
   FQuery.ParambyName('FO_DSC_RED').AsString := FORNECEDOR.FFO_DSC_RED;
   FQuery.ParambyName('FO_CNPJ').AsString := FORNECEDOR.FFO_CNPJ;
   FQuery.ParambyName('FO_INSCR_EST').AsString := FORNECEDOR.FFO_INSCR_EST;
   FQuery.ParambyName('FO_RUA').AsString := FORNECEDOR.FFO_RUA;
   FQuery.ParambyName('FO_CEP').AsString := FORNECEDOR.FFO_CEP;
   FQuery.ParambyName('FO_CIDADE').AsString := FORNECEDOR.FFO_CIDADE;
   FQuery.ParambyName('FO_BAIRRO').AsString := FORNECEDOR.FFO_BAIRRO;
   FQuery.ParambyName('FO_UF').AsString := FORNECEDOR.FFO_UF;
   FQuery.ExecSQL;
 finally
   FreeAndNil(FQuery);
 end;
end;
72.....
73.....
function TFORNECEDORDAO.SelecionarFornecedor4() : TSQLQuery;
var
 FQuery : TSQLQuery;
begin
 FQuery := TSQLQuery.Create(Nil);
 try
   FQuery.SQLConnection := Conecta;
   FQuery.SQL.Clear;
   FQuery.SQL.Add('Select Max(Fo_Cod)+1');
   FQuery.SQL.Add('From Fornecedor');
   Result := FQuery;
 finally
   FreeAndNil(FQuery);
 end;
end;
end.
```

Quadro 6 – Código fonte gerado a partir da tabela e do DFM

Os métodos `Alterar` e `Excluir` dependem de chaves primárias, pois são nelas que a ferramenta baseia-se para gerar o comando. Se todos os campos de uma tabela fizerem parte da chave primária o método `Alterar` será criado, mas não haverá o comando `update`

especificado. Caso existam tabelas do banco que não tenham nenhuma coluna definida como chave primária ou todos os campos fizerem parte da mesma, o método é criado, mas não terá nenhum comando SQL no mesmo. No Quadro 7 são apresentados os métodos `Alterar` e `Excluir` de uma classe DAO gerada pela ferramenta.

```
.....
.....
procedure TFORNECEDORDAO.Alterar(FORNECEDOR : TFORNECEDOR);
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Update FORNECEDOR');
    FQuery.SQL.Add('Set FO_DSC = :FO_DSC,');
    FQuery.SQL.Add('    FO_DSC_RED = :FO_DSC_RED,');
    FQuery.SQL.Add('    FO_CNPJ = :FO_CNPJ,');
    FQuery.SQL.Add('    FO_INSCR_EST = :FO_INSCR_EST,');
    FQuery.SQL.Add('    FO_RUA = :FO_RUA,');
    FQuery.SQL.Add('    FO_CEP = :FO_CEP,');
    FQuery.SQL.Add('    FO_CIDADE = :FO_CIDADE,');
    FQuery.SQL.Add('    FO_BAIRRO = :FO_BAIRRO,');
    FQuery.SQL.Add('    FO_UF = :FO_UF');
    FQuery.SQL.Add(' Where FO_COD = :FO_COD');
    FQuery.ParambyName('FO_COD').AsInteger := FORNECEDOR.FFO_COD;
    FQuery.ParambyName('FO_DSC').AsString := FORNECEDOR.FFO_DSC;
    FQuery.ParambyName('FO_DSC_RED').AsString := FORNECEDOR.FFO_DSC_RED;
    FQuery.ParambyName('FO_CNPJ').AsString := FORNECEDOR.FFO_CNPJ;
    FQuery.ParambyName('FO_INSCR_EST').AsString := FORNECEDOR.FFO_INSCR_EST;
    FQuery.ParambyName('FO_RUA').AsString := FORNECEDOR.FFO_RUA;
    FQuery.ParambyName('FO_CEP').AsString := FORNECEDOR.FFO_CEP;
    FQuery.ParambyName('FO_CIDADE').AsString := FORNECEDOR.FFO_CIDADE;
    FQuery.ParambyName('FO_BAIRRO').AsString := FORNECEDOR.FFO_BAIRRO;
    FQuery.ParambyName('FO_UF').AsString := FORNECEDOR.FFO_UF;
    FQuery.ExecSQL;
  finally
    FreeAndNil(FQuery);
  end;
end;

procedure TFORNECEDORDAO.Excluir(FORNECEDOR : TFORNECEDOR);
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Delete From FORNECEDOR');
    FQuery.SQL.Add('Where FO_COD = :FO_COD');
    FQuery.ParambyName('FO_COD').AsInteger := FORNECEDOR.FFO_COD;
    FQuery.ExecSQL;
  finally
    FreeAndNil(FQuery);
  end;
end;
......
......
```

Quadro 7 – Métodos `Alterar` e `Excluir` de uma classe DAO

3.2.5 Especificação da BNF utilizada

A BNF utilizada no desenvolvimento da aplicação é adaptada da mesma utilizada no trabalho final da disciplina de Banco de Dados II do curso de bacharelado em Ciências da Computação. Dos comandos analisados, a BNF não faz a verificação das cláusulas de *subselect*, *union* e *having*.

Para a especificação da BNF foi utilizada a ferramenta GALS (Gerador de Analisadores Léxicos e Sintáticos), que após a especificação da mesma é possível gerar classes de analisadores léxicos, sintáticos e semânticos. Já para a análise das expressões SQL o padrão é o ANSI 92.

No Quadro 8 são apresentadas algumas palavras reservadas e símbolos especiais da gramática.

```
UPDATE                : {U}{P}{D}{A}{T}{E}
SET                   : {S}{E}{T}
INSERT                : {I}{N}{S}{E}{R}{T}
INTO                  : {I}{N}{T}{O}
VALUES                : {V}{A}{L}{U}{E}{S}
DELETE                : {D}{E}{L}{E}{T}{E}
FROM                  : {F}{R}{O}{M}
SELECT                : {S}{E}{L}{E}{C}{T}
DISTINCT              : {D}{I}{S}{T}{I}{N}{C}{T}
AS                    : {A}{S}
JOIN                  : {J}{O}{I}{N}
NATURAL               : {N}{A}{T}{U}{R}{A}{L}
ON                    : {O}{N}
WHERE                 : {W}{H}{E}{R}{E}
ORDER                 : {O}{R}{D}{E}{R}
GROUP                 : {G}{R}{O}{U}{P}
BY                    : {B}{Y}
TIMESTAMP             : {T}{I}{M}{E}{S}{T}{A}{M}{P}
UPPER                 : {U}{P}{P}{E}{R}
MONTH                 : {M}{O}{N}{T}{H}
YEAR                  : {Y}{E}{A}{R}
COUNT                 : {C}{O}{U}{N}{T}
SUM                   : {S}{U}{M}
MAX                   : {M}{A}{X}
MIN                   : {M}{I}{N}
AVG                   : {A}{V}{G}
NULL                  : {N}{U}{L}{L}
ASC                   : {A}{S}{C}
NOT                   : {N}{O}{T}
AND                   : {A}{N}{D}
OR                    : {O}{R}
LIKE                  : {L}{I}{K}{E}
IS                    : {I}{S}
IN                    : {I}{N}
CHAR                  : {C}{H}{A}{R}
CHARACTER             : {C}{H}{A}{R}{A}{C}{T}{E}{R}
INTEGER               : {I}{N}{T}{E}{G}{E}{R}
VARCHAR2              : {V}{A}{R}{C}{H}{A}{R}2
VARCHAR               : {V}{A}{R}{C}{H}{A}{R}
INT                   : {I}{N}{T}
NUMBER                : {N}{U}{M}{B}{E}{R}
NUMERIC               : {N}{U}{M}{E}{R}{I}{C}
SMALLINT              : {S}{M}{A}{L}{L}{I}{N}{T}
MATCH                 : {M}{A}{T}{C}{H}
COLUMN                : {C}{O}{L}{U}{M}{N}

"("
")"
"*"
"+"
","
"-"
"."
"/"
":"
";"
"<"
"="
">"
"<>"
"<="
">="
"'"
```

Quadro 8 – Palavras reservadas e símbolos especiais da gramática

Para que se encontrem os comandos SQL iguais, uma das formas de conseguir êxito é especificar uma gramática. No Quadro 9 é apresentado parte da gramática do comando *select*.

```
<SQLTest> ::=  <Comando>  <PontoVirgula>;
<Comando> ::= <SelectSQL> | <UpdateSQL> | <InsertSQL> | <DeleteSQL>;
<SelectSQL> ::=  <SelectStmt> <UnionSelect> ;
<UnionSelect> ::= UNION <AllOpcional> <SelectStmt> <UnionSelect> | î ;=
<AllOpcional> ::= ALL | î ;
<SubSelectSQL> ::= <SelectSQL> ;
<SelectStmt> ::= <SelectClause> <FromClause> <WhereOpcional> <GroupByClauseOpcional>
<HavingClauseOpcional> <OrderByClauseOpcional>;
<OrderByClauseOpcional> ::= <OrderByClause> | î ;
<HavingClauseOpcional> ::= <HavingClause> | î ;
<GroupByClauseOpcional> ::= <GroupByClause> | î ;
<SelectClause> ::= SELECT #114 <DistinctAllOpcional> <SelectFieldList>;
<DistinctAllOpcional> ::= DISTINCT | ALL | î ;
<FromClause> ::= FROM <FromTableList>;
<FromTableList> ::= <QualifiedTable> <QualifiedSeparator>;
<QualifiedTable> ::= ident #109 <AsAliasOpcional> #127;
<AsAliasOpcional> ::= AS #109 <Alias> #109 | ident #109 | î;
<Alias> ::= ident;
<QualifiedSeparator> ::= "," <FromTableList> | î;
<WhereClause> ::= WHERE #106 #128 <SearchCondition> #129;
<HavingClause> ::= HAVING #130 <SearchCondition> #131;
<OrderByClause> ::= ORDER BY #132 <OrderByFldList> #133;
<GroupByClause> ::= GROUP BY #134 <FieldList> #135;
<SelectFieldList> ::= <SelectField> <SeparatorSelectField>;
<SeparatorSelectField> ::= <ItemSeparator> <SelectField> <SeparatorSelectField> | î ;
<SelectField> ::= <Expression> #125 <AsAlias> | "*" #126 #125;
<AsAlias> ::= AS <Alias> |  î ;
<IdentAsterisco> ::= ident| "*" ;
<Expression> ::= <SimpleExpression> <RelationSimpleExp>;
<SimpleExpression> ::= <NotOpcional> <Term> <RepeteOperator>;
<RepeteOperator> ::= <Operator> <NotOpcional> <Term> <RepeteOperator> | î;
<RelationSimpleExp> ::= <Relation> #104 <SimpleExpression> <RelationSimpleExp> | î;
<NotOpcional> ::= <NotOperator> #103 | î;
<Term> ::= <MenosOpcional> <K> <identOpcional> #155;
<K> ::= <FieldTest> | #140 <ColumnFunction> | <FunctionExpr> | #150 <OpenExpClose> #151;
<OpenExpClose> ::= <OpenParens> <ExpSubSel> <CloseParens>;
<ItemSeparator> ::= "," ;
```

Quadro 9 – Parte da especificação da gramática do comando SQL *Select*

Na análise dos comandos SQL são aceitos quatro comandos: *select*, *insert*, *update* e *delete*.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas para o desenvolvimento da ferramenta e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

A ferramenta foi implementada na linguagem de programação Delphi. Foi utilizado o ambiente de desenvolvimento Borland Developer Studio 2006. Para a geração das classes que fazem a análise léxica, sintática e semântica foi utilizada a ferramenta GALS.

3.3.2 Implementação da ferramenta

Esta seção apresenta a implementação da ferramenta, mostrando trechos de código fonte e explicação sobre os mesmos. Está dividida em análise dos arquivos DFM, análise dos arquivos PAS e substituição de métodos.

3.3.2.1 Análise dos arquivos DFM

Inicialmente é apresentada na Figura 10 um diagrama de seqüência para o processo de análise dos arquivos fonte DFM.

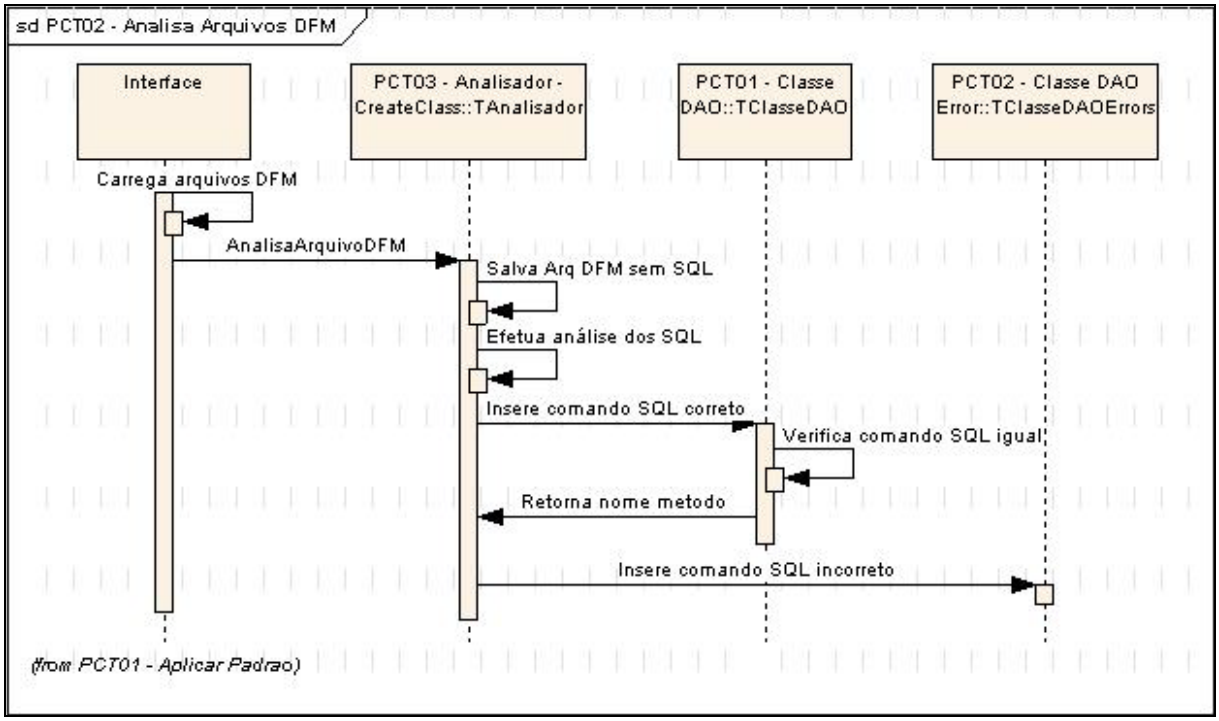


Figura 10 - Diagrama de seqüência da análise dos arquivos fontes DFM

No Quadro 10 é apresentado parte do código fonte responsável pela análise do arquivo DFM. Já no Quadro 11 é apresentada a função que insere um comando SQL na lista para ser substituído.

Para poder executar a análise do DFM é preciso primeiro selecionar um projeto. A análise dos arquivos DFM é dividida nas seguintes fases:

a) o arquivo DPR é carregado para memória para que em seguida seja feita a busca de todos os arquivos DFM;
b) tendo os arquivos DFM listados, a ferramenta invoca o método `AnalisaArquivoDFM`

(método da classe `TAnalisador`). Este método verifica se o arquivo DFM está em formato binário ou texto e carrega o arquivo para memória;

c) após carregar o arquivo DFM, a ferramenta verifica linha a linha onde existe um componente do tipo `TSQLQuery` conforme visto nas linhas de 4 a 10 do Quadro 10;

d) no código fonte DFM onde existir uma declaração deste componente a ferramenta sabe que existe um comando SQL e retira o comando e armazena em uma lista de comandos conforme pode ser visto entre as linhas 14 e 32 do Quadro 10;

e) ao final deste processo e com a lista contendo todos os comandos deste formulário a aplicação salva o arquivo analisado sem os comandos nele encontrados conforme linha 47 do Quadro 10;

f) a ferramenta com base nas classes geradas a partir da BNF faz a análise semântica de todos os comandos SQL encontrados no formulário, para que sejam armazenados dados importantes para a verificação de comandos SQL iguais, como os campos de projeção de um *select*, os campos da cláusula `where` ou os campos da cláusula `set` do comando *update*;

g) se o comando estiver correto o mesmo é adicionado na lista de objetos de acordo com seu tipo, *insert*, *update*, *delete* e *select,* e os métodos `ClasseDAO.AddInsert`, `ClasseDAO.AddUpdate`, `ClasseDAO.AddDelete`, `ClasseDAO.AddSelect` são invocados pela classe `TAnalisador`. No Quadro 11 é apresentado a função que insere os comandos de *insert* na lista;

h) se o comando for igual, apenas é adicionado em qual componente o comando está inserido e retorna o nome do método a ser criado conforme visto entre às linhas 7 e 22 do Quadro 11;

i) se o comando ainda não existir ele é armazenado e a ferramenta retorna o nome do método que será criado na classe DAO através do método de nomeação definido para cada tipo de comando;

j) caso algum dos comandos SQL analisados tenha um erro de sintaxe a ferramenta armazena o comando, o nome do Formulário em que o comando estava inserido e a tabela a qual se refere o comando, para apresentar ao usuário. Esses comandos são inseridos na classe de comandos incorretos.

```
01  while xAux <= ArqAnalisar.Count -1 do
02    begin
03      Linha   := ArqAnalisar.Strings[xAux];
04      PosQuery := Pos(BuscaTSQLQuery, Linha);
05      if PosQuery > 0 then
06        begin
07          Ini := Pos(BuscaObject, Linha) + Length(BuscaObject);
08          Fim := Pos(BuscaDoisPontos, Linha) - Ini;
09          NomeQuery := Copy(Linha, Ini, Fim);
10        end;
11      PosIni  := Pos(BuscaSQLStrings, Linha);
12      if PosIni > 0 then
13        begin
14          Inc(xAux);
15          Linha   := ArqAnalisar.Strings[xAux];
16          UltPosic := Copy(Linha, Length(Linha), 1);
17          if UltPosic = BuscaFinalStrSQL then
18           begin
19             ListaSQL.Values[NomeQuery]:=ListaSQL.Values[NomeQuery]+''+Copy(Trim(Linha),2,
20             Inc(xAux);
21            end
22          else
23            begin
24              while UltPosic <> BuscaFinalStrSQL do
25                begin
26                  if AcabouConcat then
27                    begin
28                      AcabouConcat := False;
29                      ListaSQL.Values[NomeQuery] := ListaSQL.Values[NomeQuery] +
30                    end
31                  else
32                    ListaSQL.Values[NomeQuery] := ListaSQL.Values[NomeQuery] + ' ' +
33                  Inc(xAux);
34                  .
35                  .
36                  .
37                  .
38                end;
39            end;
40        end
41       else
42        begin
43          ArquivoConvertido.Add(Linha);
44         Inc(xAux);
45        end;
46    end;
47 ArquivoConvertido.SaveToFile(PCaminhoDFM);
```

Quadro 10 - Parte do código fonte do método que analisa os arquivos DFM

```
01 function TClasseDAO.AddInsert(ComandoSQL, NomeFormUnit, NomeCompon, Tabela, ListaInsert: 02 string) : string;
03 var
04   xAux : Integer;
05   MI   : TInsertMetodo;
06 begin
07  for xAux := 0 to Pred(ListaInserts.Count) do
08    begin
09      MI := TInsertMetodo(ListaInserts[xAux]);
10      if MI.InsertIgual(ListaInsert) then
11        begin
12          if (UpperCase(MI.Componentes.Strings[0]) = UpperCase(MetodoAlterar)) or
13             (UpperCase(MI.Componentes.Strings[0]) = UpperCase(MetodoExcluir)) or
14             (UpperCase(MI.Componentes.Strings[0]) = UpperCase(MetodoInserir)) or
15             (UpperCase(MI.Componentes.Strings[0]) = UpperCase(MetodoSelecionar)) then
16            MI.Componentes.Strings[0] := NomeCompon
17          else
18            MI.Componentes.Add(NomeCompon);
19          Result := MI.FNomeMetodo;
20          Exit;
21        end;
22    end;
23  MI                  := TInsertMetodo.Create;
24  if UpperCase(NomeCompon) = UpperCase(MetodoInserir) then
25    MI.FNomeMetodo      := MetodoInserir
26  else
27    MI.FNomeMetodo      := MI.RegraNomeacaoMetodo(Tabela, tcInsert);
28  MI.FNomeFormUnit      := NomeFormUnit;
29  MI.Componentes.Add(NomeCompon);
30  MI.CamposTabelas.Text := Tabela;
31  MI.FTipoComando       := tcInsert;
32  MI.CamposInsert.Text  := ListaInsert;
33  MI.ComandoInsert.Text := ComandoSQL;
34  ListaInserts.Add(MI);
35  Result := MI.FNomeMetodo;
36 end;
```

Quadro 11 – Código fonte que insere os comandos de *insert*

Para a verificação de comandos SQL iguais cada tipo de comando tem itens diferentes a analisar. No método de inserir são verificadas as colunas envolvidas. Já no comando de *update* são verificadas as colunas a serem alteradas (da cláusula `set`) e as colunas da cláusula *where*. Contudo, no comando *delete* são verificadas as colunas da cláusula *where*. No comando de *select* é que está a maior verificação de todos os comandos: as colunas de projeção, as tabelas envolvidas, colunas da cláusula *where*, *groupby* e *orderby*. No Quadro 12 é apresentado o método que faz a verificação dos comandos de *delete* iguais, que funcionam inicialmente verificando se a quantidade de colunas da cláusula *where* do comando que está sendo inserido é igual a quantidade de colunas do comando já inserido. Em seguida é feita a verificação de todos os campos para verificar se os campos que estão sendo inseridos não são iguais aos já inseridos.

```
Function TDeleteMetodo.DeleteIgual(ListaWhere: string): Boolean;
var
  Wheres : TStringList;
  xAux, xInt : Integer;
  Campo : string;
  AchouIgual : Boolean;
begin
  Result     := True;
  xAux       := 0;  xInt       := 0;  AchouIgual := False;
  Wheres     := TStringList.Create;
  try
    Wheres.Text     := ListaWhere;
    if Wheres.Count <> CamposWhere.Count then
      begin
        Result := False;
        Exit;
      end;
    while xAux <= Wheres.Count - 1 do
      begin
        Campo   := Wheres.Strings[xAux];
        while xInt <= CamposWhere.Count - 1 do
          begin
            if UpperCase(Campo) = UpperCase(CamposWhere.Strings[xInt]) then
              begin
                AchouIgual := True;
                Break;
              end;
            Inc(xInt);
          end;
        if not AchouIgual then
          Break;
        xInt := 0;
        Inc(xAux);
      end;
    if not AchouIgual then
      begin
        Result := False;
        Exit;
      end;
  finally
    FreeAndNil(Wheres);
  end;
end;
```

Quadro 12 – Código fonte do método que verifica se existem comandos *delete* iguais

3.3.2.2 Análise dos arquivos PAS

O objetivo desta análise é encontrar comandos SQL que possam estar especificados nos arquivos fontes PAS diretamente nos componentes ou em constantes. O processo de análise dos arquivos PAS é dividido nas fases busca dos arquivos, busca dos comandos SQL em constantes e dos componentes `TSQLQuery`, referências aos componentes e análise léxica, sintática e semântica dos comandos SQL encontrados.

A partir do arquivo de projeto DPR são buscados todos os arquivos PAS armazenando-os em uma lista para que a ferramenta saiba quais arquivos devem ser analisados.

A ferramenta invoca o método `AnalisaArquivoPAS` (Método da classe `TAnalisador`) para analisar o arquivo fonte selecionado. Inicia-se carregando o arquivo fonte para a memória. Conforme é apresentado no Quadro 13 da linha 01 até a 40 são verificados se existem comandos SQL adicionados em constantes e se existir os mesmos são adicionados em uma lista de comandos para futuramente criar o método DAO. Conforme mostra o Quadro 14, a ferramenta busca todos os componentes do tipo `TSQLQuery` que estão declarados no arquivo fontes PAS. Quando descoberto são armazenados o nome em uma lista de componentes, pois é através destes que a busca por comandos SQL é feita. Se ao final da análise não for encontrado nenhum componente, a análise deste arquivo PAS é interrompida, pois não existe comando SQL sendo adicionado a um componente deste arquivo PAS. No Quadro 13 é apresentado um trecho do código fonte que busca expressões SQL em constantes. Já no Quadro 14 é mostrado o código fonte que busca o nome dos componentes declarados no formulário analisado.

```
01 while xAux <= ArquivoOld.Count - 1 do
02  begin
03    Linha := ArquivoOld.Strings[xAux];
04    AchouConst := Pos(UpperCase(BuscaConst), UpperCase(Linha)) > 0;
05    if AchouConst then
06      begin
07        Inc(xAux);
08        Linha := ArquivoOld.Strings[xAux];
09        while xAux <= ArquivoOld.Count - 1 do
10          begin
11            xSelect := Pos(UpperCase(BuscaSelect), UpperCase(Linha)) > 0;
12            xInsert := Pos(UpperCase(BuscaInsert), UpperCase(Linha)) > 0;
13            xUpdate := Pos(UpperCase(BuscaUpdate), UpperCase(Linha)) > 0;
14            xDelete := Pos(UpperCase(BuscaDelete), UpperCase(Linha)) > 0;
15            NomeConst := Trim(Copy(Linha, 1, Pos(BuscaIgual, Linha) - 1));
16            if (xSelect) or (xInsert) or (xUpdate) or (xDelete) then
17              begin
18                Comando := '';
19              NomeConst:=Trim(Copy(Linha,1,Pos(UpperCase(BuscaIgual),UpperCase(Linha))-1));
20                Ini  := Pos(BuscaApostrofo, Linha) + 1;
21                Fim  := VerificaUltimoApostrofo(Linha) - Ini;
22                xStr := Copy(Linha, Ini, Fim);
23                Comando := Comando + xStr;
24                Inc(xAux);
25                while xAux <= ArquivoOld.Count - 1 do
26                  begin
27                    Linha := ArquivoOld.Strings[xAux];
28                    Ini := Pos(BuscaApostrofo, Linha) + 1;
29                    Fim  := VerificaUltimoApostrofo(Linha) - Ini;
30                    xStr := Copy(Linha, Ini, Fim);
31                    Comando := Comando + xStr;
32                    TemPtoVirgula := Pos(BuscaPtoVirgula, Linha) > 0;
33                    Inc(xAux);
34                    if TemPtoVirgula then
35                      Break
36                  end;
37                ArqConstante.Values[NomeConst] := Comando
38              end
39            else
40              Inc(xAux);
41            Linha             := ArquivoOld.Strings[xAux];
42            TemVar            := Pos(UpperCase(BuscaVar), UpperCase(Linha)) > 0;
43            TemType           := Pos(UpperCase(BuscaType), UpperCase(Linha)) > 0;
44            TemBegin          := Pos(UpperCase(BuscaBegin), UpperCase(Linha)) > 0;
45            TemImplementation := Pos(UpperCase(BuscaImplementation), UpperCase(Linha)) > 0;
46            if (TemVar) or (TemType) or (TemBegin) or (TemImplementation) then
47              Break;
48          end;
49      end
50    else
51      Inc(xAux);
52  end;
```

Quadro 13 - Parte do código fonte que verifica a existência de comandos SQL em constantes

```
while xAux <= ArquivoOld.Count - 1 do
  begin
    Linha := ArquivoOld.Strings[xAux];
    Achou := Pos(BuscaTSQLQuery, Linha) > 0;
    if Achou then
      begin
        NomeQuery := Trim(Copy(Linha, 1, Pos(':', Linha) - 1));
        ArquivoQuery.Add(NomeQuery);
      end;
    Inc(xAux);
    ArquivoNew.Add(Linha);
    if Pos(Implem, Linha) > 0 then
      begin
        IniciarDe := xAux;
        ArqAux.Text := ArquivoNew.Text;
      end;
  End;
```

Quadro 14 - Parte do código fonte que verifica o nome dos componentes `TSQLQuery` no arquivo PAS

A ferramenta busca linha por linha no código fonte uma referência ao nome do

componente armazenado na busca anterior. Na linha em que for encontrado é verificado se existem dois membros desse componente: o método `Add` e a propriedade `Text`. Se existir retira o comando e armazena em uma lista de comandos. Essa busca é repetida para todos os componentes do tipo `TSQLQuery` que forem encontrados no arquivo PAS. O Quadro 15 apresenta o código fonte que executa a busca por comandos SQL nos arquivos fontes PAS.

```
...
...
  TemSQLAdd  := Pos(UpperCase(BuscaSQLAdd), UpperCase(Linha)) > 0;
  TemSQLText := Pos(UpperCase(BuscaSQLText), UpperCase(Linha)) > 0;
  if TemSQLAdd then
    begin
      TemComandSQL := Pos(BuscaIniComanSQL, UpperCase(Linha)) > 0;
      if TemComandSQL then
        begin
          Ini     := Pos(BuscaIniComanSQL, Linha) + Length(BuscaIniComanSQL);
          Fim     := VerificaUltimoApostrofo(Linha) - Ini;
          ArqComandSQL.Values[BuscaQuery] := ArqComandSQL.Values[BuscaQuery] + ' ' +
Copy(Linha, Ini, Fim);
        end
      else if TemConst then
        begin
          //aqui ver como fazer posi o comando esta na constante.
          //Inicialmente tem que guardar o comando quando faz a busca pelas querys nos forms.
          Ini     := Pos('(', Linha) + 1;
          Fim     := Pos(')', Linha) - Ini;
          NomeConst := Copy(Linha, Ini, Fim);
          ArqComandSQL.Values[BuscaQuery] := ArqConstante.Values[NomeConst];
        end;
    end
  else if TemSQLText then
    begin
      TemComandSQL := Pos(BuscaApostrofo, UpperCase(Linha)) > 0;
      TemConst     := not TemComandSQL;
      if TemComandSQL then
        begin
          Ini     := Pos(BuscaApostrofo, Linha) + Length(BuscaApostrofo);
          Fim     := VerificaUltimoApostrofo(Linha) - Ini;
          ArqComandSQL.Values[BuscaQuery] := ArqComandSQL.Values[BuscaQuery] + ' ' +
Copy(Linha, Ini, Fim);
        end
      else if TemConst then
        begin
          //aqui ver como fazer posi o comando esta na constante.
          //Inicialmente tem que guardar o comando quando faz a busca pelas querys nos forms.
          Ini     := Pos(BuscaIniComSQLText, Linha) + Length(BuscaIniComSQLText);;
          Fim     := Pos(BuscaPtoVirgula, Linha) - Ini;
          NomeConst := Copy(Linha, Ini, Fim);
          ArqComandSQL.Values[BuscaQuery] := ArqConstante.Values[NomeConst];
        end;
    end
  else
    ArquivoNew.Add(Linha);
...
...
```

Quadro 15 – Parte do código fonte que executa busca por comandos SQL nos arquivos fontes PAS

A ferramenta com base nas classes geradas a partir da BNF faz a análise léxica, sintática e semântica dos comandos SQL encontrados nos arquivos fontes PAS, para que sejam armazenados dados importantes para a verificação de comandos SQL iguais, como os campos de projeção de um *select*, os campos da cláusula `where` ou os campos da cláusula `set` do comando *update*;

No término da análise de cada arquivo fonte é invocado o método `VerificaComando`

(método da classe `TAnalisador`) que invoca o método `BuscaArmazenaDadosSQL` (método privado da classe `TAnalisador`) que é responsável por fazer a análise léxica, sintática e semântica de cada comando SQL encontrado, armazenando-os para posteriormente fazer a criação do método na classe DAO e a substituição do comando pelo método. No Quadro 16 é apresentado o método executa a análise dos comandos SQL que foram retirados dos arquivos fontes PAS.

```
Function TAnalisador.BuscaArmazenaDadosSQL(NomeComponente, Linha : string): string;
var
  NomeMetodoDAO, NomeClasseDAO : string;
  TipoComando : TTipoComandoSQL;
begin
  //cria objetos para a anlise do comando SQL.
  Lexico    := TLexico.Create;
  Sintatico := TSintatico.Create;
  Semantico := TSemantico.Create;
  try
    try
      //submete o comando SQL ao analisador
      Lexico.setInput(Linha);
      Sintatico.parse(Lexico, Semantico);
      //retorna o tipo do comando
      TipoComando := Semantico.GetTipoComando;
      //Arqmazena dados do comando SQL e retorna o nome do metodo a ser criado na classe DAO.
      NomeMetodoDAO := RetornaMetodoDAO(TipoComando, NomeComponente, UpperCase(Linha));
      //retornar o nome do metodo Criado
      Result := NomeMetodoDAO;
    except
      On E: ESyntaticError do
        begin
          //Trada erros sintáticos. Comandos fora do padrao ansii 92.
          AdicionaSQLErrados(Linha);
        end;
    end;
  finally
    FreeAndNil(Lexico);
    FreeAndNil(Sintatico);
    FreeAndNil(Semantico);
  end;
end;
```

Quadro 16 – Método que executa a analise dos comandos SQL

3.3.3 Substituição dos métodos

Ao final da análise dos arquivos DFM e PAS, acontece a substituição dos comandos SQL por chamadas a métodos da classe DAO. A substituição acontece nos arquivos fontes PAS, pois são neles que os comandos SQL estão sendo referenciados através de componentes do tipo `TSQLQuery`. Inicia-se a substituição buscando no arquivo PAS quais os componentes do tipo `TSQLQuery` estão adicionados no arquivo PAS analisado e armazena em uma lista de componentes o nome dos mesmos. Em seguida é buscada na lista de componentes o primeiro a ser encontrado no arquivo PAS. Após, é analisado linha a linha para identificar uma referência ao componente buscado da lista. Quando encontrado verifica se na linha em análise

existe uma chamada aos métodos `ParamByName`, `Open` ou `ExecSQL`. Se a chamada for para o método `ParamByName` se substitui todas as linhas pelo método criado. Se for uma chamada ao método `Open` ou o método `ExecSQL` verifica se a substituição ainda não aconteceu e em caso negativo substitui pelo método criado. Além de adicionar o método substituído, a linha deve ser adicionada no novo arquivo fonte. Em seguida é buscado o nome do próximo componente até que todos tenham sido analisados e se necessário substituídos.

3.3.4 Operacionalidade da implementação

Nesta seção é apresentada a operacionalidade da ferramenta, sendo iniciado com os diagramas de classe de interface logo em seguida uma explicação sobre o uso da ferramenta.

3.3.4.1 Diagrama de classes da interface da ferramenta

Na Figura 11 é apresentada a tela principal da ferramenta. Já Figura 12 apresenta o diagrama de classes da interface da ferramenta de geração de classes e métodos DAO em relação à classe `TCreateClass`.

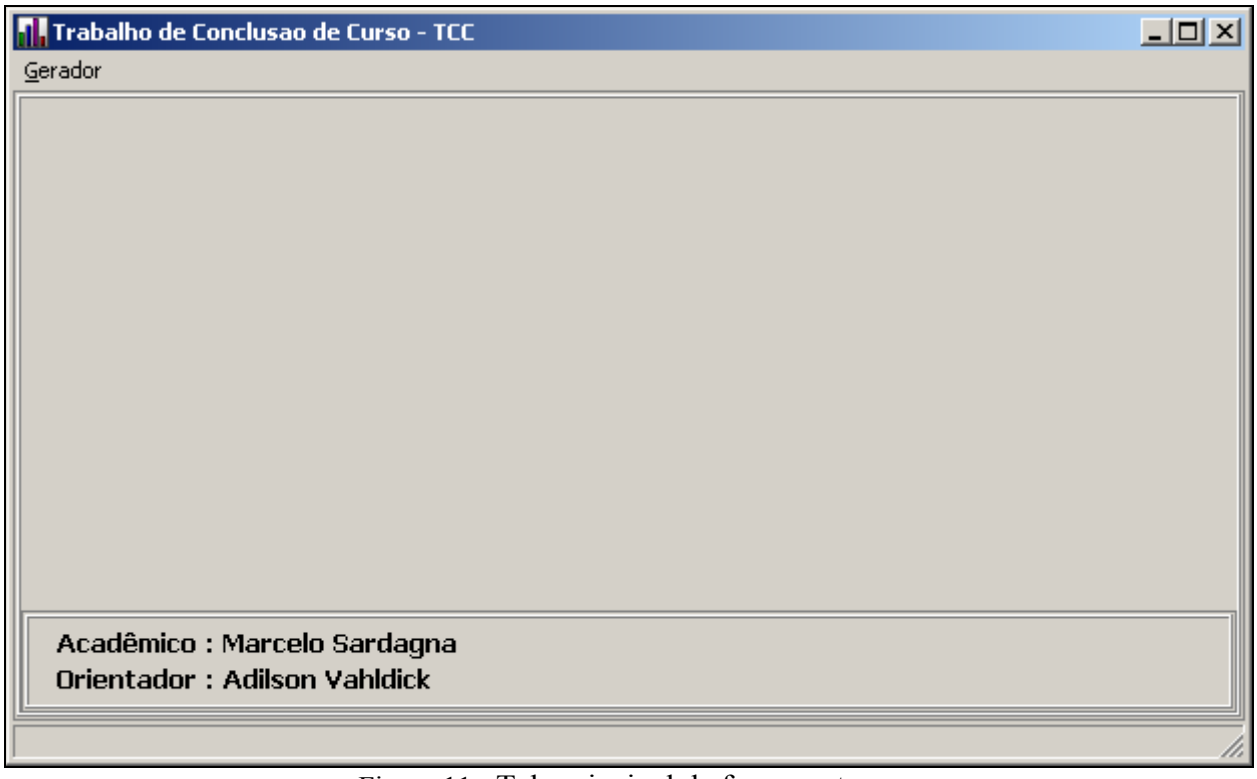


Figura 11 - Tela principal da ferramenta

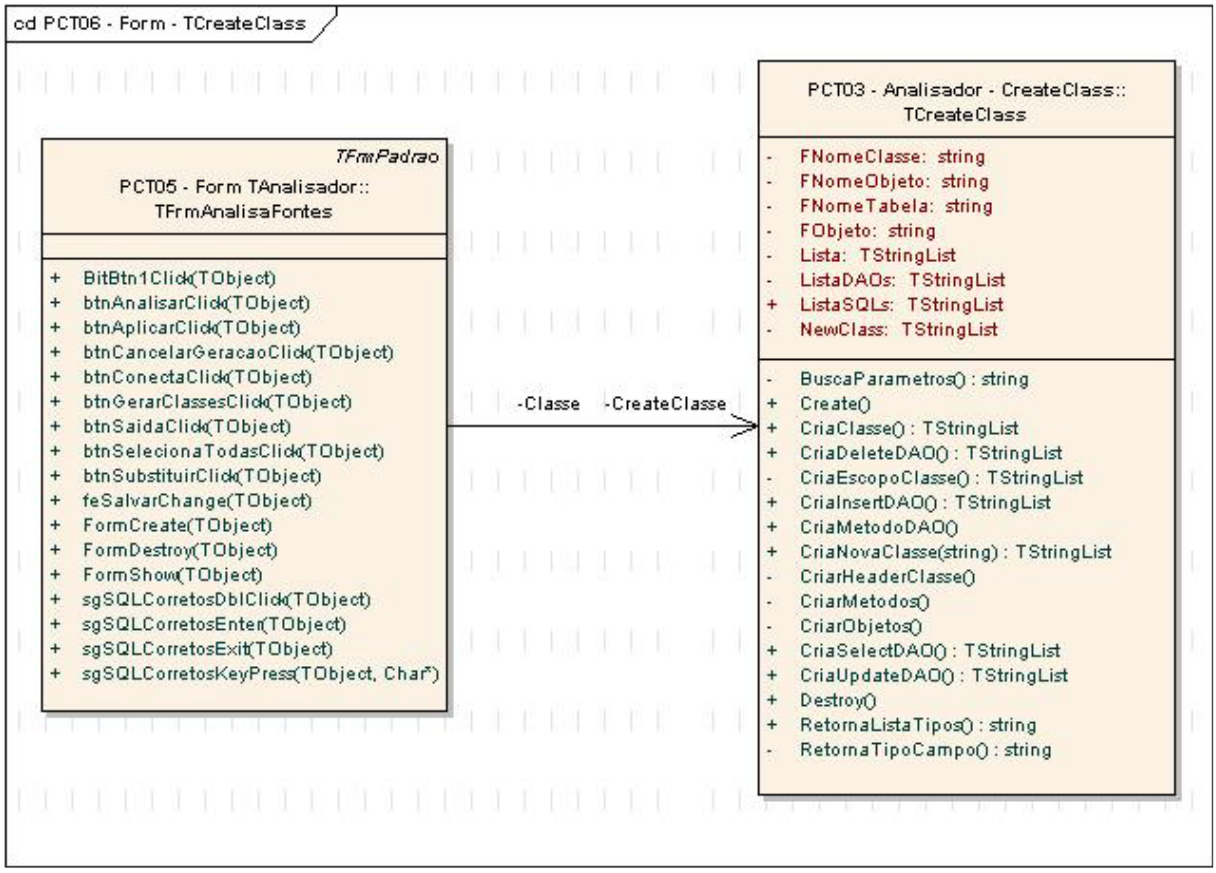


Figura 12 - Diagrama de classes da interface em relação à classe `TCreateClass`

Conforme mostra a Figura 13, o usuário inicia a análise selecionando o banco de dados do projeto (1) e o login e senha (2) do mesmo para que em seguida seja possível conectar ao banco de dados (3) para buscar as informações das tabelas do banco. Em seguida é informada a pasta onde irão ser salvas as novas classes DAO (4). Assim que conectado, são listadas todas as tabelas (5) e o usuário deve selecionar as tabelas a serem geradas as classes DAO ou selecionar a opção de selecionar todas (6). Logo após o usuário preciona o botão `Gerar Classes` (7) e a ferramenta, instancia a classe `TCreateClasse` e invoca o método `CriaClasse` que é responsável por criar as classes DAO juntamente com os métodos. A classe `TCreateClass` retorna as classes DAO para que a interface possa salvar a classe no local selecionado pelo usuário. A ferramenta gera as classes, apresenta a mensagem que as classes DAO foram geradas com sucesso e habilita o formulário para a análise dos arquivos fonte DFM e PAS. Para ilustrar a explicação a Figura 14 apresenta o diagrama de seqüência.

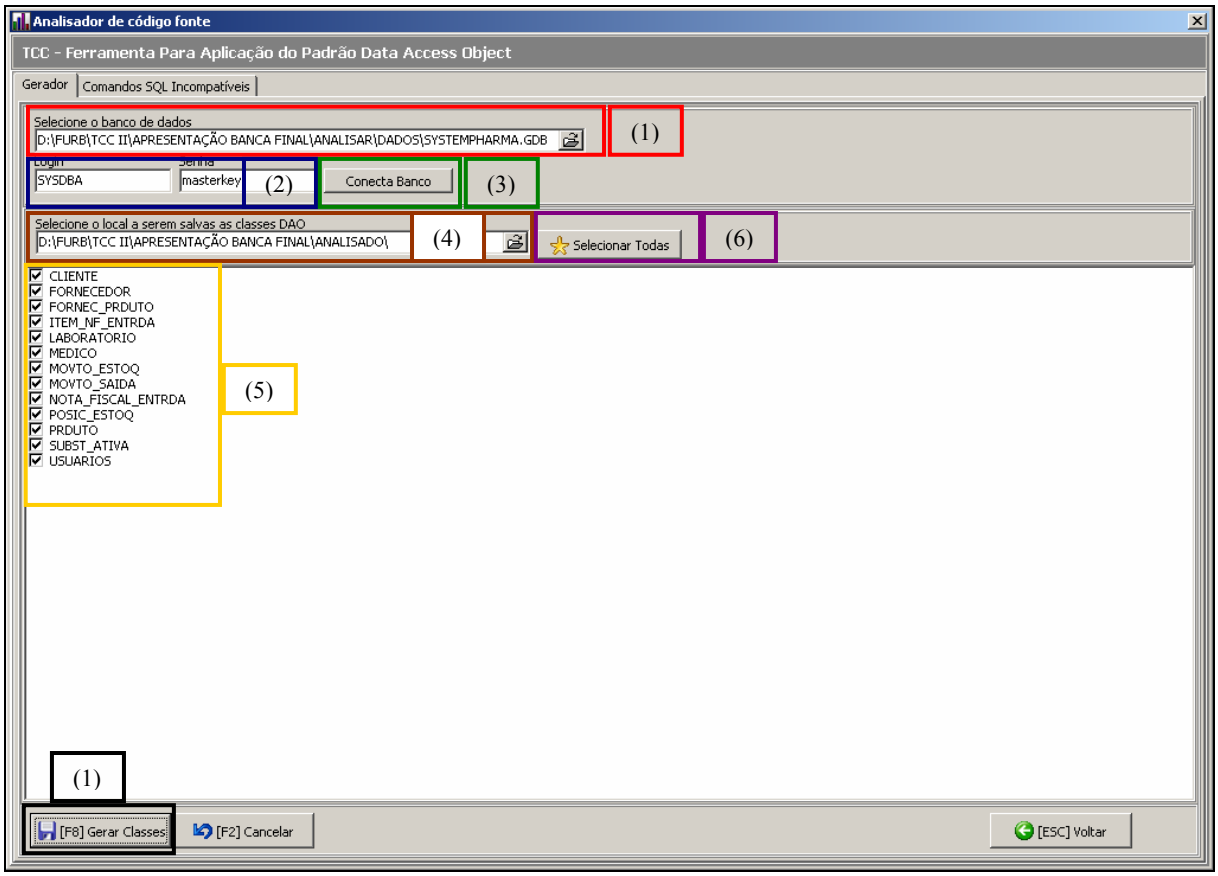


Figura 13 – Seleção das tabelas para serem geradas as classes DAO

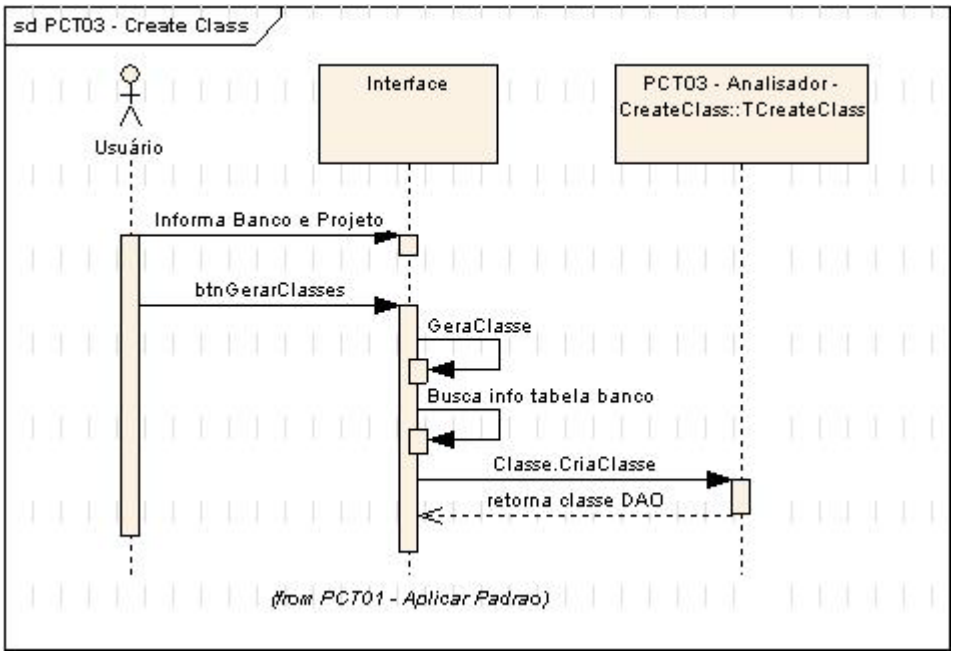


Figura 14 – Diagrama de seqüência da criação da classe DAO a partir de uma tabela no banco

A Figura 15, apresenta o diagrama de classes da interface de análise dos arquivos fontes DFM e PAS em relação à classe `TClasseDAO`.

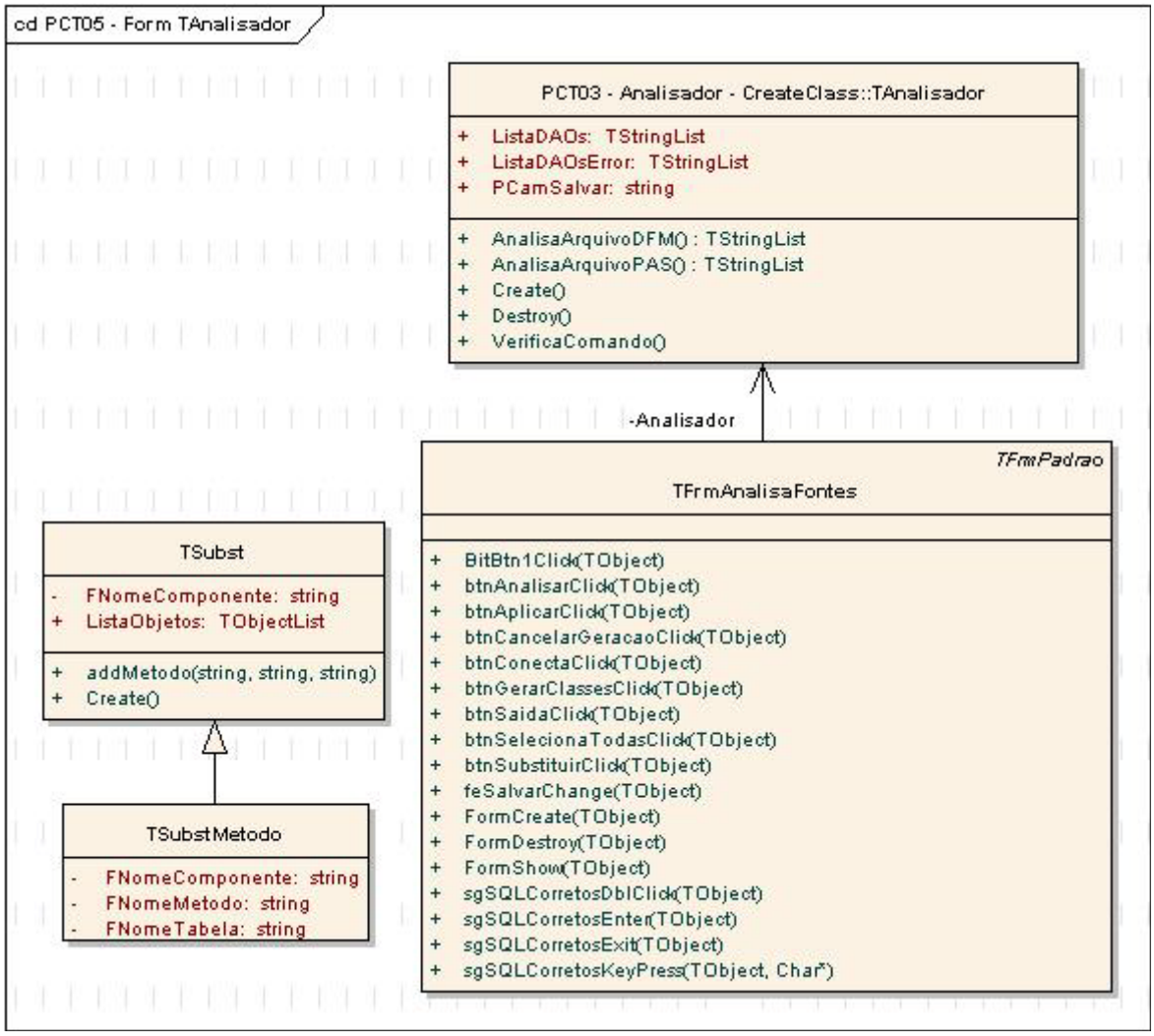


Figura 15 - Diagrama de classe da interface em relação à classe `TAnalisador`

Partindo da premissa que o usuário tenha selecionado o projeto (1), indicado a pasta onde serão salvo os novos arquivos fontes (2) e clicado no botão `Analisar Fontes` (3), é invocado o método de interface `btnAnalisarClick`. Este método por sua vez chama os métodos privados `AnalisaArquivosDFM` e `AnalisaArquivosPAS`. O método `AnalisaArquivosDFM` faz uma chamada ao método `AnalisaArquivoDFM` da classe `TAnalisador` que faz a análise dos arquivos fontes DFM. Já o método `AnalisaArquivosPAS` invoca o método `AnalisaArquivoPAS` que também está na classe `TAnalisador` na qual é responsável por fazer a análise dos arquivos fontes PAS. Após a análise os comandos SQL são apresentados em tela conforme mostra Figura 16.

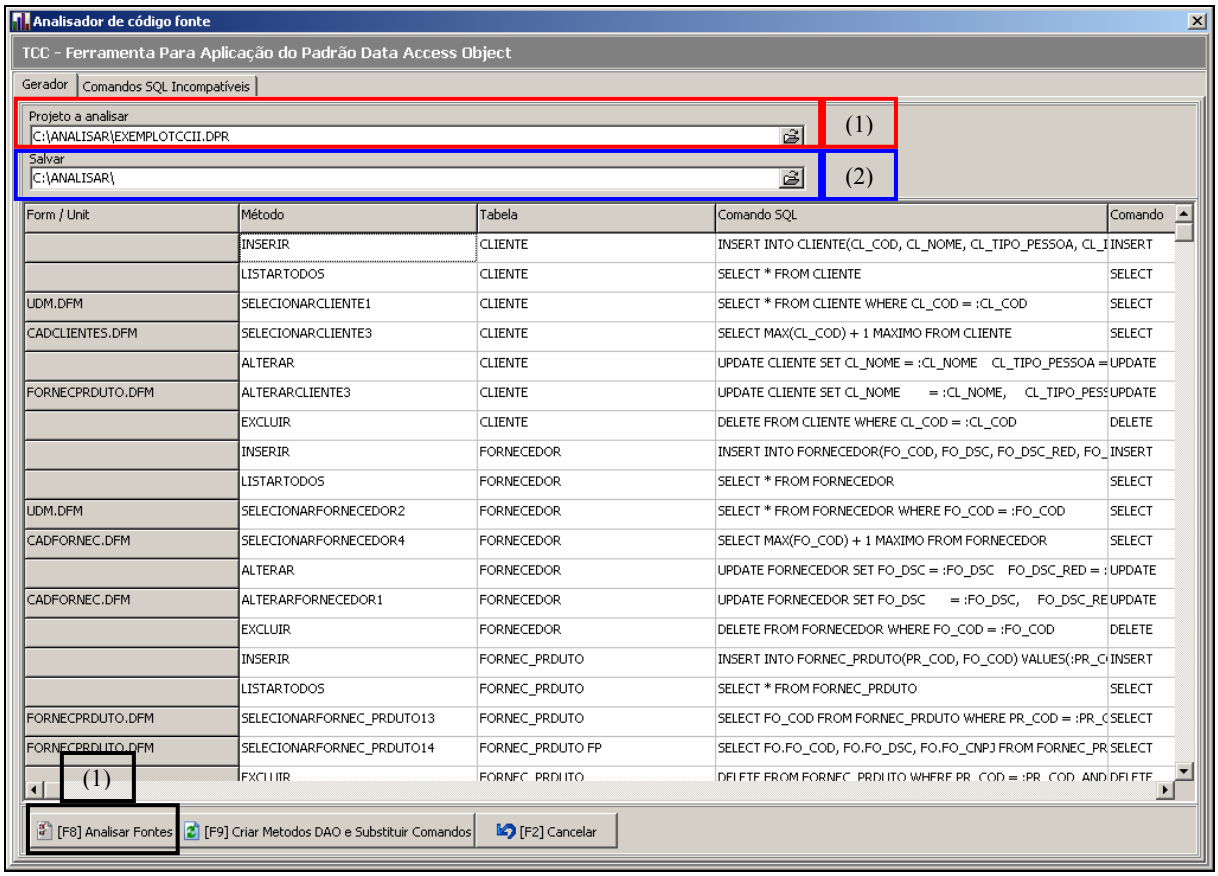


Figura 16 – Tela com o resultado da análise dos arquivos fontes DFM e PAS

O usuário tem a opção de alterar o nome dos métodos listados pressionando um *enter* ou um duplo *click* do *mouse* no método desejado para alteração. A tela para alteração do nome do método será apresentada e basta digitar o nome desejado (1) e pressionar o botão `Salvar` (2). A tela para a alteração do nome do método é apresentada na Figura 17.

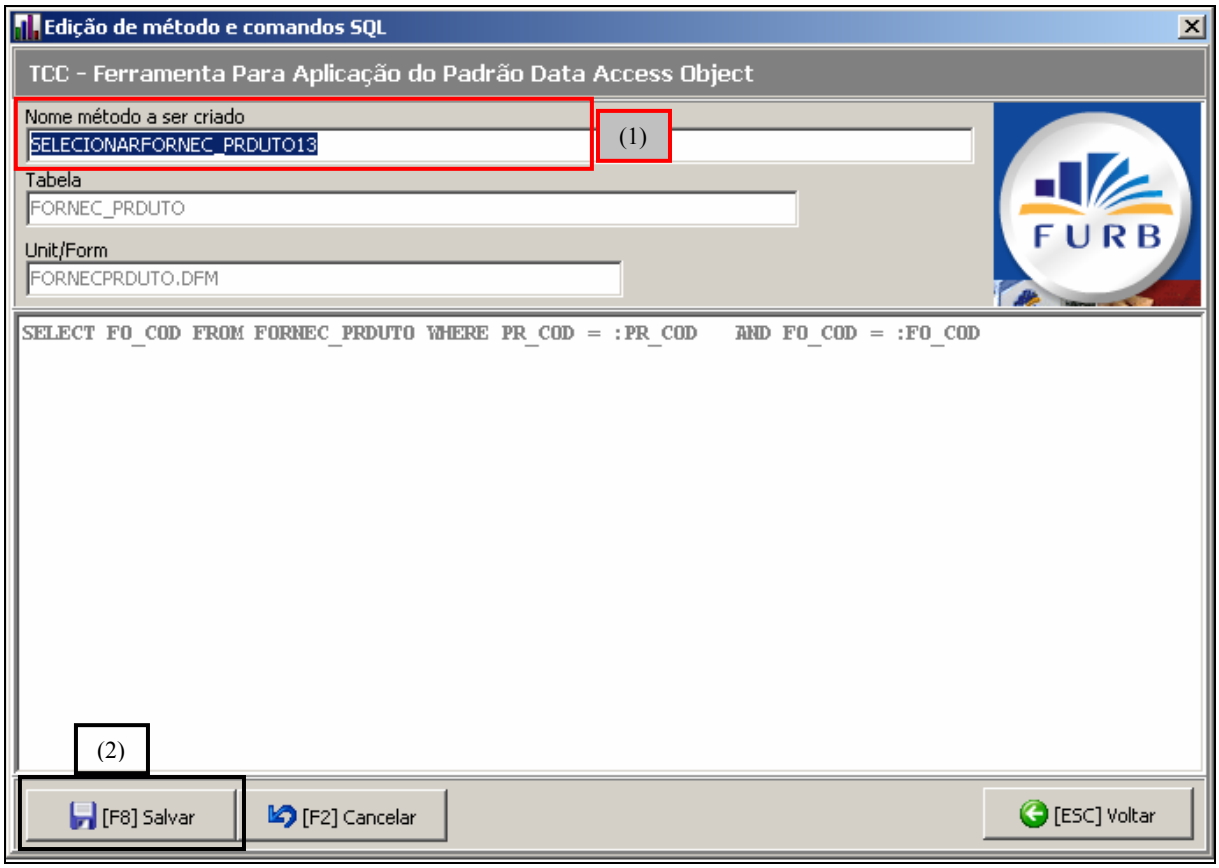


Figura 17 – Tela para alterar nome do método DAO a ser criado

Na seqüência o usuário pressiona o botão `Criar Métodos DAO e Substituir Comandos` para que a ferramenta crie os métodos DAO e substitua os comandos pelos métodos da classe DAO.

3.4 ESTUDO DE CASO

Para o desenvolvimento da ferramenta foi implementada uma aplicação de teste para que se pudesse verificar de onde iniciar e como deveria ficar após a análise. A mesma foi implementada tendo três telas de cadastros básicos contendo os comandos necessários para entendimento do problema a ser resolvido.

A Figura 18, apresenta o modelo MER da aplicação feita para exemplificar o uso da ferramenta. A partir da modelagem são geradas as tabelas do banco de dados que por sua vez acabam resultando em classes DAO.

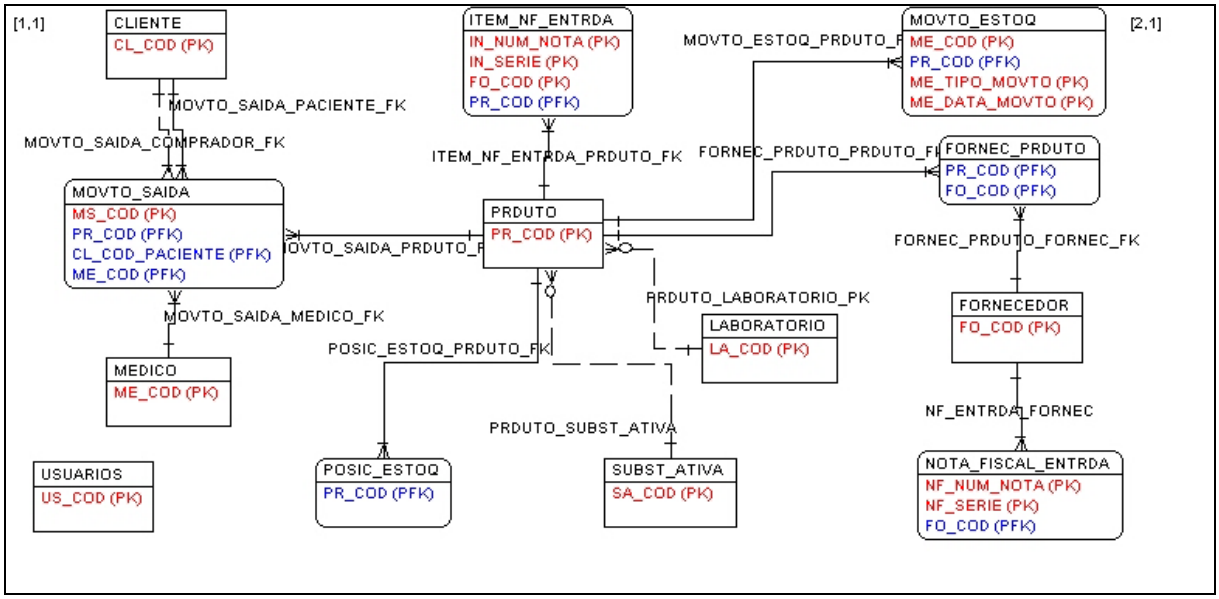
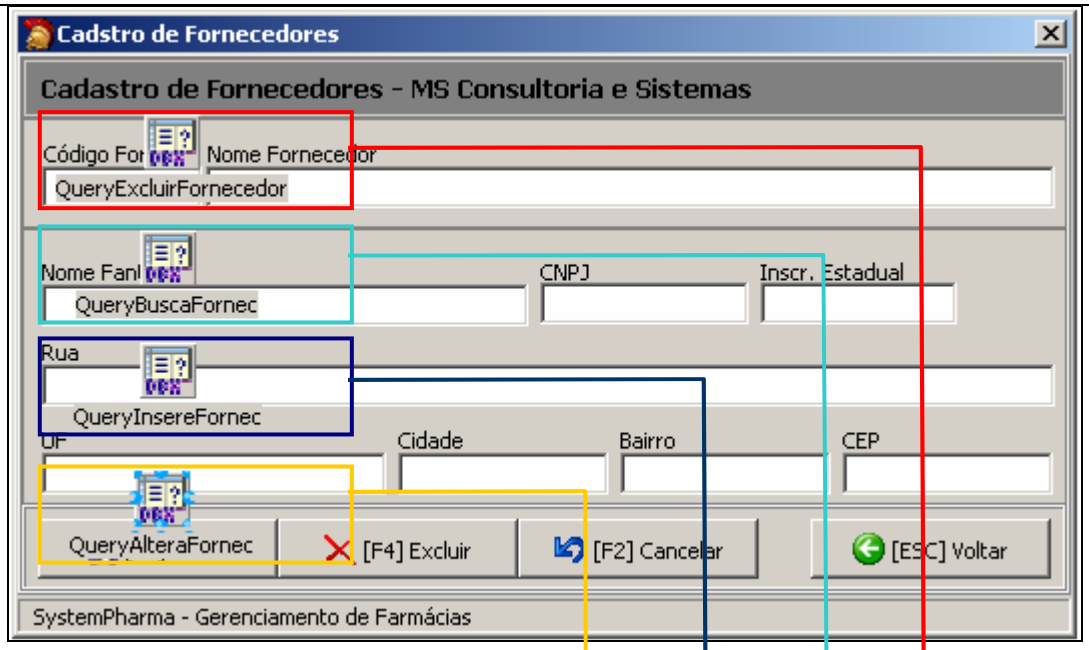


Figura 18 – Modelagem de dados da aplicação de exemplo

No Quadro 17, é apresentado parte de um arquivo fonte DFM antes de ser analisado pela ferramenta. Já no Quadro 18 é apresentado um arquivo fontes PAS também antes de ser analisado pela ferramenta. No Quadro 19 é apresentado o mesmo arquivo fonte DFM após ser analisado pela ferramenta. Em seguida no Quadro 20 é apresentado o arquivo fonte PAS modificado pela ferramenta. A diferença entre os Quadro 19 e Quadro 20 é que neste último não existem mais os comandos SQL. A ferramenta retirou todos os comandos e substituiu por chamadas a métodos da classe DAO. Para finalizar no Quadro 21 é apresentada uma classe DAO gerada a partir de uma tabela no banco de dados e após a análise dos arquivos DFM e PAS.



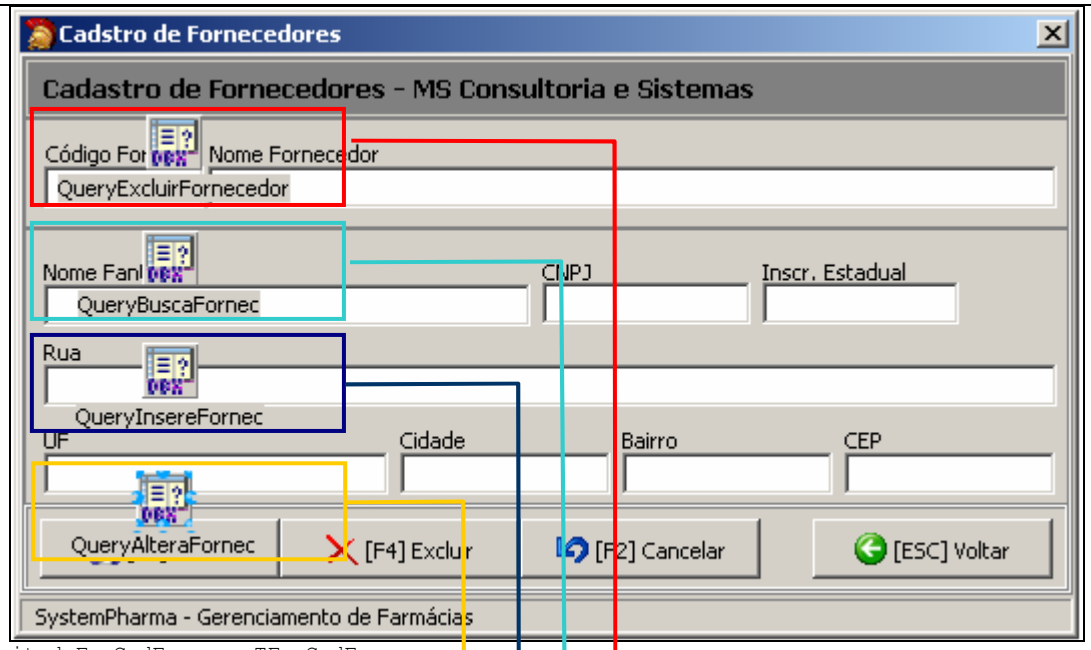

```
inherited FrmCadFornec: TFrmCadFornec
  .
  .
  object QueryExcluirFornecedor: TSQLQuery
    MaxBlobSize = -1
    Params = <>
    Left = 96
    Top = 128
  end
  object QueryBuscaFornec: TSQLQuery
    MaxBlobSize = -1
    Params = <>
    SQL.Strings = (
      'Select Max(Fo_Cod) + 1 Maximo From Fornecedor')
    Left = 206
    Top = 128
  end
  object QueryInsereFornec: TSQLQuery
    MaxBlobSize = -1
    .
    .
    SQL.Strings = (
      'Insert Into Fornecedor'
      '(Fo_Cod,  Fo_Dsc, Fo_Dsc_Red, Fo_CNPJ, Fo_Inscr_Est,'
      ' Fo_Rua, Fo_CEP, Fo_Cidade, Fo_Bairro, Fo_UF)'
      'Values'
      '(:Fo_Cod,  :Fo_Dsc, :Fo_Dsc_Red, :Fo_CNPJ, :Fo_Inscr_Est,'
      ' :Fo_Rua, :Fo_CEP, :Fo_Cidade, :Fo_Bairro, :Fo_UF)')
    Left = 320
    Top = 128
  end
  object QueryAlteraFornec: TSQLQuery
    MaxBlobSize = -1
    .
    .
    SQL.Strings = (
      'Update Fornecedor'
      'Set Fo_Dsc       = :Fo_Dsc,'
      '    Fo_Dsc_Red   = :Fo_Dsc_Red,'
      '    Fo_CNPJ      = :Fo_CNPJ,'
      '    Fo_Inscr_Est = :Fo_Inscr_Est,'
      '    Fo_Rua       = :Fo_Rua,'
      '    Fo_CEP       = :Fo_CEP,'
      '    Fo_Cidade    = :Fo_Cidade,'
      '    Fo_Bairro    = :Fo_Bairro,'
      '    Fo_UF        = :Fo_UF'
      'Where Fo_Cod     = :Fo_Cod')
    Left = 439
    Top = 128
  end
End
```

Quadro 17 - Arquivo fonte DFM antes de ter sido analisado pela ferramenta

```
Unit CadFornec;
...
type
  TFrmCadFornec = class(TFrmPadrao)
    QueryExcluirFornecedor: TSQLQuery;
    QueryBuscaFornec: TSQLQuery;
    QueryInsereFornec: TSQLQuery;
    QueryAlteraFornec: TSQLQuery;
  private
  public
  end;
implementation
{$R *.dfm}
procedure TFrmCadFornec.AlteraFornecedor;
begin
  DM.Conecta.StartTransaction(Transacao);
  try
    With QueryAlteraFornec do
      begin
        ParamByName('FO_DSC').AsString         := edNome.Text;
        ParamByName('FO_DSC_RED').AsString     := edNomeFantasia.Text;
        ParamByName('FO_CNPJ').AsString        := edInscrEst.Text;
        ParamByName('FO_INSCR_EST').AsString   := edCNPJ.Text;
        ParamByName('FO_RUA').AsString         := edRua.Text;
        ParamByName('FO_CIDADE').AsString      := edCidade.Text;
        ParamByName('FO_CEP').AsString         := edCEP.Text;
        ParamByName('FO_BAIRRO').AsString      := edBairro.Text;
        ParamByName('FO_UF').AsString          := edUF.Text;
        ParamByName('FO_COD').AsInteger        := edCodFornec.AsInteger;
        ExecSQL;
        LimpaCampos;
        HabBotoes(False, False, False, edCodFornec);
        DM.Conecta.Commit(Transacao);
      end;
  except
  end;
end;
procedure TFrmCadFornec.ExcluirFornecedor;
begin
  if MessageDlg('Você tem certeza que deseja excluir este fornecedor?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    begin
      Transacao.TransactionID:= 1;
      Transacao.IsolationLevel:=  xilReadCommitted;
      DM.Conecta.StartTransaction(Transacao);
      try
        with QueryExcluirFornecedor do
          begin
            SQL.Add('Delete From Fornecedor');
            SQL.Add('Where Fo_Cod = :Fo_Cod');
            ParamByName('FO_COD').AsInteger := edCodFornec.AsInteger;
            ExecSQL;
          end;
        DM.Conecta.Commit(Transacao);
      except
        On E: Exception do
          begin
           DM.Conecta.Rollback(Transacao);
           MessageDlg('Ocorreu o seguinte erro ao gravar o fornecedor.', mtError, [mbOK], 0);
           Abort;
          end;
      end;

      LimpaCampos;
      HabBotoes(False, False, False, edCodFornec);
    end;
end;
procedure TFrmCadFornec.InserirFornecedor;
begin
  Transacao.TransactionID:= 1;
  Transacao.IsolationLevel:=  xilReadCommitted;
  DM.Conecta.StartTransaction(Transacao);
  try
    With QueryInsereFornec do
      begin
        ParamByName('FO_COD').AsInteger        := BuscaCodFornec;
        ParamByName('FO_DSC').AsString         := edNome.Text;
        ParamByName('FO_DSC_RED').AsString     := edNomeFantasia.Text;
        ParamByName('FO_CNPJ').AsString        := edInscrEst.Text;
        ParamByName('FO_INSCR_EST').AsString   := edCNPJ.Text;
        ParamByName('FO_RUA').AsString         := edRua.Text;
        ParamByName('FO_CIDADE').AsString      := edCidade.Text;
        ParamByName('FO_CEP').AsString         := edCEP.Text;
        ParamByName('FO_BAIRRO').AsString      := edBairro.Text;
        ParamByName('FO_UF').AsString          := edUF.Text;
        ExecSQL;
        LimpaCampos;
        HabBotoes(False, False, False, edCodFornec);
        DM.Conecta.Commit(Transacao);
      end;
  except
...
function TFrmCadFornec.BuscaCodFornec: Integer;
begin
  With QueryBuscaFornec do
    begin
      Close;
      Open;
      Result := FieldByName('MAXIMO').AsInteger;
    end;
end;
end.
```

Quadro 18 - Arquivo fontes PAS antes de ser analisado pela ferramenta




```
inherited FrmCadFornec: TFrmCadFornec
  ...
  object QueryExcluirFornecedor: TSQLQuery
    MaxBlobSize = -1
    Params = <>
    Left = 96
    Top = 128
  end
  object QueryBuscaFornec: TSQLQuery
    MaxBlobSize = -1
    Params = <>
    Left = 206
    Top = 128
  end
  object QueryInsereFornec: TSQLQuery
    MaxBlobSize = -1
    Params = <>
    Left = 320
    Top = 128
  end
  object QueryAlteraFornec: TSQLQuery
    MaxBlobSize = -1
    Params = <>
    Left = 439
    Top = 128
  end
End
```

Quadro 19 - Código fonte DFM após ser analisado pela ferramenta

```
Unit CadFornec;
...
type
  TFrmCadFornec = class(TFrmPadrao)
    QueryExcluirFornecedor: TSQLQuery;
    QueryBuscaFornec: TSQLQuery;
    QueryInsereFornec: TSQLQuery;
    QueryAlteraFornec: TSQLQuery;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
Implementation
{$R *.dfm}
procedure TFrmCadFornec.AlteraFornecedor;
var
  Fornec : TFornecedor
begin
  Transacao.TransactionID  := 1;
  Transacao.IsolationLevel :=  xilReadCommitted;
  DM.Conecta.StartTransaction(Transacao);
  try
    With QueryAlteraFornec do
      begin
        FORNECEDOR.ALTERARFORNECEDOR1(Fornec);
        LimpaCampos;
        HabBotoes(False, False, False, edCodFornec);
        DM.Conecta.Commit(Transacao);
      end;
  except
    ...
  end;
end;
procedure TFrmCadFornec.ExcluirFornecedor;
var
  Fornec : TFornecedor
begin
 if MessageDlg('Deseja excluir?',mtConfirmation,[mbYes,mbNo],0)=mrYes then
    begin
      Transacao.TransactionID:= 1;
      Transacao.IsolationLevel:=  xilReadCommitted;
      DM.Conecta.StartTransaction(Transacao);
      try
        with QueryExcluirFornecedor do
          begin
            FORNECEDOR.EXCLUIR(Fornec);
          end;
        DM.Conecta.Commit(Transacao);
      except
        On E: Exception do
          begin
            ...
          end;
      end;
      ...
    end;
end;
procedure TFrmCadFornec.InserirFornecedor;
var
  Fornec : TFornecedor
begin
  ...
  DM.Conecta.StartTransaction(Transacao);
  try
    With QueryInsereFornec do
      begin
        FORNECEDOR.INSERIR(Fornec);
        LimpaCampos;
        HabBotoes(False, False, False, edCodFornec);
        DM.Conecta.Commit(Transacao);
      end;
  except
    On E: Exception do
      begin
        ...
      end;
  end;
end;
function TFrmCadFornec.BuscaCodFornec: Integer;
begin
  With QueryBuscaFornec do
    begin
      if Active then
        Close;
      QueryBuscaFornec := FORNECEDOR.SELECIONARFORNECEDOR4;
      Open;
      Result := FieldByName('MAXIMO').AsInteger;
    end;
end;
end.
```

Quadro 20 - Arquivo fonte PAS após ser analisado pela ferramenta

```
unit FORNECEDORDAO;
interface
uses
  Windows, SysUtils, Variants, Classes, Controls, FMTBcd, DB, SqlExpr;
type
  TFORNECEDOR = class
  private
    FFO_COD : Integer;
    FFO_DSC : String;
    FFO_DSC_RED : String;
    FFO_CNPJ : String;
    FFO_INSCR_EST : String;
    FFO_RUA : String;
    FFO_CEP : String;
    FFO_CIDADE : String;
    FFO_BAIRRO : String;
    FFO_UF : String;
  public
    property FO_COD : Integer Read FFO_COD Write FFO_COD ;
    property FO_DSC : String Read FFO_DSC Write FFO_DSC ;
    property FO_DSC_RED : String Read FFO_DSC_RED Write FFO_DSC_RED ;
    property FO_CNPJ : String Read FFO_CNPJ Write FFO_CNPJ ;
    property FO_INSCR_EST : String Read FFO_INSCR_EST Write FFO_INSCR_EST ;
    property FO_RUA : String Read FFO_RUA Write FFO_RUA ;
    property FO_CEP : String Read FFO_CEP Write FFO_CEP ;
    property FO_CIDADE : String Read FFO_CIDADE Write FFO_CIDADE ;
    property FO_BAIRRO : String Read FFO_BAIRRO Write FFO_BAIRRO ;
    property FO_UF : String Read FFO_UF Write FFO_UF ;
  end;
  TFORNECEDORDAO = class
  private
    Conecta : TSQLConnection;
  public
    constructor Create(Conexao : TSQLConnection);
    procedure Inserir(FORNECEDOR : TFORNECEDOR);
    procedure Alterar(FORNECEDOR : TFORNECEDOR);
    procedure Excluir(FORNECEDOR : TFORNECEDOR);
    function ListarTodos(FQuery : TSQLQuery) : TSQLQuery;
    procedure AlterarFornecedor1(Fornecedor : TFORNECEDOR);
    function SelecionarFornecedor2(Fo_Cod : String) : TSQLQuery;
    function SelecionarFornecedor4() : TSQLQuery;
  end;
implementation
constructor TFORNECEDORDAO.Create(Conexao: TSQLConnection);
begin
  inherited Create;
  Conecta := Conexao;
end;
procedure TFORNECEDORDAO.Inserir(FORNECEDOR : TFORNECEDOR);
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Insert Into FORNECEDOR');
    FQuery.SQL.Add('(FO_COD, FO_DSC, FO_DSC_RED, FO_CNPJ, FO_INSCR_EST, FO_RUA, FO_CEP, FO_CIDADE, FO_BAIRRO, FO_UF)');
    FQuery.SQL.Add('Values');
    FQuery.SQL.Add('(:FO_COD, :FO_DSC, :FO_DSC_RED, :FO_CNPJ, :FO_INSCR_EST, :FO_RUA, :FO_CEP, :FO_CIDADE, :FO_BAIRRO,
:FO_UF)');
    FQuery.ParambyName('FO_COD').AsInteger := FORNECEDOR.FFO_COD;
    FQuery.ParambyName('FO_DSC').AsString := FORNECEDOR.FFO_DSC;
    FQuery.ParambyName('FO_DSC_RED').AsString := FORNECEDOR.FFO_DSC_RED;
    FQuery.ParambyName('FO_CNPJ').AsString := FORNECEDOR.FFO_CNPJ;
    FQuery.ParambyName('FO_INSCR_EST').AsString := FORNECEDOR.FFO_INSCR_EST;
    FQuery.ParambyName('FO_RUA').AsString := FORNECEDOR.FFO_RUA;
    FQuery.ParambyName('FO_CEP').AsString := FORNECEDOR.FFO_CEP;
    FQuery.ParambyName('FO_CIDADE').AsString := FORNECEDOR.FFO_CIDADE;
    FQuery.ParambyName('FO_BAIRRO').AsString := FORNECEDOR.FFO_BAIRRO;
    FQuery.ParambyName('FO_UF').AsString := FORNECEDOR.FFO_UF;
    FQuery.ExecSQL;
  finally
    FreeAndNil(FQuery);
  end;
end;
procedure TFORNECEDORDAO.Alterar(FORNECEDOR : TFORNECEDOR);
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Update FORNECEDOR');
    FQuery.SQL.Add('Set FO_DSC = :FO_DSC,');
    FQuery.SQL.Add('    FO_DSC_RED = :FO_DSC_RED,');
    FQuery.SQL.Add('    FO_CNPJ = :FO_CNPJ,');
    FQuery.SQL.Add('    FO_INSCR_EST = :FO_INSCR_EST,');
    FQuery.SQL.Add('    FO_RUA = :FO_RUA,');
    FQuery.SQL.Add('    FO_CEP = :FO_CEP,');
    FQuery.SQL.Add('    FO_CIDADE = :FO_CIDADE,');
    FQuery.SQL.Add('    FO_BAIRRO = :FO_BAIRRO,');
    FQuery.SQL.Add('    FO_UF = :FO_UF');
    FQuery.SQL.Add(' Where FO_COD = :FO_COD');
    FQuery.ParambyName('FO_COD').AsInteger := FORNECEDOR.FFO_COD;
    FQuery.ParambyName('FO_DSC').AsString := FORNECEDOR.FFO_DSC;
    FQuery.ParambyName('FO_DSC_RED').AsString := FORNECEDOR.FFO_DSC_RED;
    FQuery.ParambyName('FO_CNPJ').AsString := FORNECEDOR.FFO_CNPJ;
    FQuery.ParambyName('FO_INSCR_EST').AsString := FORNECEDOR.FFO_INSCR_EST;
    FQuery.ParambyName('FO_RUA').AsString := FORNECEDOR.FFO_RUA;
    FQuery.ParambyName('FO_CEP').AsString := FORNECEDOR.FFO_CEP;
```

```
    FQuery.ParambyName('FO_CIDADE').AsString := FORNECEDOR.FFO_CIDADE;
    FQuery.ParambyName('FO_BAIRRO').AsString := FORNECEDOR.FFO_BAIRRO;
    FQuery.ParambyName('FO_UF').AsString := FORNECEDOR.FFO_UF;
    FQuery.ExecSQL;
  finally
    FreeAndNil(FQuery);
  end;
end;
procedure TFORNECEDORDAO.Excluir(FORNECEDOR : TFORNECEDOR);
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Delete From FORNECEDOR');
    FQuery.SQL.Add('Where FO_COD = :FO_COD');
    FQuery.ParambyName('FO_COD').AsInteger := FORNECEDOR.FFO_COD;
    FQuery.ExecSQL;
  finally
    FreeAndNil(FQuery);
  end;
end;
function TFORNECEDORDAO.ListarTodos(FQuery : TSQLQuery) : TSQLQuery;
begin
  FQuery.SQL.Clear;
  FQuery.SQLConnection := Conecta;
  FQuery.SQL.Add('Select * From FORNECEDOR');
  Result := FQuery;
end;
procedure TFORNECEDORDAO.AlterarFornecedor1(Fornecedor : TFornecedor);
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Update Fornecedor');
    FQuery.SQL.Add('Set Fo_Dsc = :Fo_Dsc');
    FQuery.SQL.Add('    Fo_Dsc_Red = :Fo_Dsc_Red');
    FQuery.SQL.Add('    Fo_CNPJ = :Fo_CNPJ');
    FQuery.SQL.Add('    Fo_Inscr_Est = :Fo_Inscr_Est');
    FQuery.SQL.Add('    Fo_Rua = :Fo_Rua');
    FQuery.SQL.Add('    Fo_CEP = :Fo_CEP');
    FQuery.SQL.Add('    Fo_Cidade = :Fo_Cidade');
    FQuery.SQL.Add('    Fo_Bairro = :Fo_Bairro');
    FQuery.SQL.Add('    Fo_UF = :Fo_UF');
    FQuery.SQL.Add('Where Fo_Cod= :Fo_Cod');
    FQuery.ParamByName('FO_COD').AsInteger := Fornecedor.FFo_Cod;
    FQuery.ParamByName('FO_DSC').AsString := Fornecedor.FFo_Dsc;
    FQuery.ParamByName('FO_DSC_RED').AsString := Fornecedor.FFo_Dsc_Red;
    FQuery.ParamByName('FO_CNPJ').AsString := Fornecedor.FFo_CNPJ;
    FQuery.ParamByName('FO_INSCR_EST').AsString := Fornecedor.FFo_Inscr_Est;
    FQuery.ParamByName('FO_RUA').AsString := Fornecedor.FFo_Rua;
    FQuery.ParamByName('FO_CEP').AsString := Fornecedor.FFo_CEP;
    FQuery.ParamByName('FO_CIDADE').AsString := Fornecedor.FFo_Cidade;
    FQuery.ParamByName('FO_BAIRRO').AsString := Fornecedor.FFo_Bairro;
    FQuery.ParamByName('FO_UF').AsString := Fornecedor.FFo_UF;
    FQuery.ExecSQL;
  finally
    FreeAndNil(FQuery);
  end;
end;
function TFORNECEDORDAO.SelecionarFornecedor2(Fo_Cod : String) : TSQLQuery;
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Select *');
    FQuery.SQL.Add('From Fornecedor');
    FQuery.SQL.Add('Where Fo_Cod= :Fo_Cod');
    FQuery.ParamByName('FO_COD').AsString := Fo_Cod;
    Result := FQuery;
  finally
    FreeAndNil(FQuery);
  end;
end;
function TFORNECEDORDAO.SelecionarFornecedor4() : TSQLQuery;
var
  FQuery : TSQLQuery;
begin
  FQuery := TSQLQuery.Create(Nil);
  try
    FQuery.SQLConnection := Conecta;
    FQuery.SQL.Clear;
    FQuery.SQL.Add('Select Max(Fo_Cod)+1');
    FQuery.SQL.Add('From Fornecedor');
    Result := FQuery;
  finally
    FreeAndNil(FQuery);
  end;
end;
end.
```

Quadro 21 - Classe DAO gerada a partir de uma tabela e após análise dos arquivos fontes DFM e PAS

3.5 RESULTADOS E DISCUSSÃO

Após a análise da ferramenta foi verificado que todos os comandos SQL da aplicação de exemplo foram retirados dos códigos fontes DFM e PAS e os mesmos inseridos nas respectivas classes DAO. Ainda foi constatado que a ferramenta substituiu todos os comandos por métodos das classes DAO que foi criada.

Em relação aos trabalhos correlatos observa-se que existem dois trabalhos similares a ferramenta proposta. Em relação a Grott (2003), ele exemplifica o uso de padrões de projeto e a ferramenta desenvolvida neste trabalho aplica um padrão de projeto. Em relação a Ferreira (2003), a semelhança está na análise de código fonte Delphi, pois ele analisa os arquivos fontes para poder identificar qual o tipo de comando SQL e o nome do formulário no qual esta expressão SQL está contida. A ferramenta do presente trabalho analisa os arquivos fontes DFM e PAS para buscar as expressões SQL e substituí-las por chamadas a métodos da classe DAO. Ferreira (2003) executa a análise léxica, sintática e semântica para descobrir qual o tipo de comando, e neste trabalho a análise é feita para verificar se existem expressões SQL iguais.

Na Quadro 22 é apresentada uma comparação das principais características entre a ferramenta desenvolvida e os trabalhos correlatos.

Comparação			
	Ferramenta para Aplicação do Padrão Data Access Object em Sistemas Desenvolvidos na Linguagem de Programação Delphi	**Ferramenta para Extração e Documentação de Rotinas de Projetos de Sistemas de Informação**	**Reutilização de Soluções com *Patterns* e *Frameworks* na Camada de Negócio**
Análise léxica, sintática e semântica	X	X	
Aplicação de Padrões	X		X
Análise dos arquivos fontes DFM e PAS	X	X	

Quadro 22 - Quadro de comparação entre a ferramenta desenvolvida e os trabalhos correlatos

4 CONCLUSÕES

O desenvolvimento da ferramenta foi voltado à aplicação de padrões de projeto objetivando elevar a qualidade na fase de desenvolvimento e que possa auxiliar no processo de manutenções, adaptações e melhorias no decorrer do desenvolvimento de software.

Os objetivos propostos no desenvolvimento deste trabalho foram alcançados. A ferramenta consegue analisar os arquivos fontes DFM e PAS de um projeto Delphi retirando as expressões SQL dos mesmos, criando na seqüência uma classe DAO para cada tabela ou grupo de tabelas com as expressões retiradas dos arquivos fontes DFM e PAS e finalizando o processo substitui os comandos encontrados por métodos da classe DAO.

A aplicação do padrão DAO no desenvolvimento de software traz benefícios como: facilidade em manutenções, centralização das expressões SQL e reutilização de código e de comandos SQL. A ferramenta pode ser aplicada para sistemas que não foram aplicados padrões de projeto e em sistemas legados. A reutilização de código acontece pelo fato de ter os comandos SQL centralizados em classes como sugere o padrão DAO.

O desenvolvimento resultou em uma ferramenta na qual o usuário apenas informa o banco de dados, o projeto e a pasta onde deverão ser salva as classes DAO geradas e os novos arquivos fonte DFM e PAS. Sendo assim, além do padrão DAO estar aplicado, existe a possibilidade de reutilizar as classes DAO em outro projeto, os comandos SQL estão centralizados e haverá maior rapidez em manutenções futuras.

Foram estudados diversos conceitos durante o desenvolvimento deste trabalho como, sintaxe de comandos SQL, componentes de acesso à base de dados, análise de códigos fontes DFM e PAS, e o GALS que auxilia na geração das classes responsáveis pela análise léxica, sintática e semântica, os quais foram essenciais para a conclusão dos objetivos deste trabalho.

4.1 LIMITAÇÕES DA FERRAMENTA

No desenvolvimento deste trabalho observou-se que:

a) a geração das classes DAO a partir de uma tabela do banco limita-se a gerar classes apenas para projetos que tenham o Interbase como banco de dados;
b) na análise léxica, sintática e semântica, a BNF não analisa as cláusulas de

SubSelect, *Union* e *Having*;

c) para os componentes de acesso a ferramenta limita-se a analisar projetos com componentes de acesso do pacote `DBExpress`, em especial o `TSQLQuery`;

d) na análise dos arquivos fontes DFM e PAS a ferramenta busca apenas componentes que estejam declarados e utilizados no arquivo fonte analisado, não sendo possível substituir o comando SQL de um componente declarado em um arquivo fonte e utilizado em outro;

e) se existir mais de um componente na mesma tela com nomes iguais a substituição pode acontecer de forma equivocada podendo substituir apenas um ou substituir todos, mas com o comando igual.

4.2 EXTENSÕES

Como extensões da ferramenta sugerem-se:

a) implementar e usar um motor de *templates* para criação das classes DAO;

b) possibilitar que os comandos SQL incorretos sejam alterados e validados;

c) alterar a gramática para que seja possível analisar as cláusulas de *SubSelect*, *Union* e *Having*;

d) possibilitar que salve os comandos encontrados e os nomes dos métodos gerados em arquivo XML, para que a criação dos métodos e substituição dos mesmos seja realizado em outro momento.

REFERÊNCIAS BIBLIOGRÁFICAS

AÉCE, I. **Analisando o Microsoft PetShop 3.0**. São Paulo, 2005. Disponível em: <http://www.projetando.net/Sections/ViewArticle.aspx?ArticleID=14>. Acesso em: 07 abr. 2007.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002.

BORLAND SOFTWARE CORPORATION. **Borland Delphi for Microsoft Windows**: help. Version 7.10.3077. [S.l.], 2005. Documento eletrônico disponibilizado com o Ambiente Delphi 2006.

FERREIRA, L. A. R. **Ferramenta para extração e documentação de rotinas de projetos de sistemas de informação.** 2003. 63 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FOWLER, M. **Padrões de arquitetura de aplicações corporativas**. Tradução Acauan Fernandes. Porto Alegre: Artmed, 2006.

GAMMA, E. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. 12. ed., Tradução Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

GROTT, M. C. **Reutilização de soluções com patterns e frameworks na camada de negócio.** 2003. 116 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HERRINGTON, J. **Code generation in action**. Greenwich: Manning Puplications, 2003.

LOUDEN, K. **Compiladores:** princípios e práticas. Tradução Flavio Soares Correia da Silva. São Paulo: Pioneira Thomson Learning, 2004.

PRICE, A. M. A.; TOSCANI, S. S. **Implementação de linguagens de programação**: compiladores. 2. ed. Porto Alegre: Sagra Luzzatto, 2001.

SHALLOWAY, A.; TROTT, J. **Explicando padrões de projeto:** uma nova perspectiva em projeto orientado a objeto. 4. ed. Tradução Ana M. de Alencar Price. Porto Alegre: Bookman, 2004.

SOUZA, A. et al. RAFF**:** um compilador para facilitar o aprendizado de algoritmos. In: ENCONTRO DE ESTUDANTES DE INFORMÁTICA DO TOCANTINS, 1., 2002, Palmas. **Anais...** Palmas: [s.n.], 2002. Não paginado. Disponível em: <http://www.ulbra-to.br/ensino/43020/artigos/anais2002/Encoinfo2002/RAFF.pdf>. Acesso em: 5 nov. 2007.

SUN. **Core J2EE patterns:** data access object. Santa Clara, [2007]. Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>. Acesso em: 15 abr. 2007

VALENTE, L. **Design patterns:** padrões para projetos. 2003. 24 f. Dissertação (Mestrado em Computação)-Universidade Federal Fluminense, Niterói. Disponível em: <http://www.ic.uff.br/~lvalente/docs/DesignPatterns_Texto.pdf>. Acesso em: 07 abr. 2007.