Técnicas de programação no desenvolvimento de um sistema em Delphi

Carlos A. P. Araújo

Centro Universitário Luterano de Santarém (CEULS)
Santarém – Pará – Brasil

carlos@controlautomacao.com.br

Resumo. Tempo de desenvolvimento e qualidade do produto de *software* são algumas das preocupações dos desenvolvedores. Neste artigo são apresentadas algumas técnicas de programação que veem ao encontro dessas necessidades. Reuso de código e generalização são técnicas que possibilitam aos desenvolvedores oferecerem aos usuários sistemas customizáveis e principalmente, diminuir o tempo de desenvolvimento. Como objeto de estudo foi utilizado um sistema em Delphi, desenvolvido pelo autor do trabalho.

1. Introdução

O objeto de estudo deste artigo é um sistema desenvolvido em Delphi e Firebird para empresas do ramo de distribuição de mercadorias em geral. O sistema tem cerca de 180 *units*/formulários implementados em Delphi e aproximadamente 60 tabelas e 90 gatilhos no SGDB Firebird. Algumas técnicas foram utilizadas em sua implementação, sendo que, julgou-se importante expor apenas três delas: herança de formulários, criação dinâmica de componentes, e relatórios dinâmicos. O artigo está organizado da seguinte maneira: na seção 2 é apresentada a herança de formulários, em seguida destaca-se a criação dinâmica de componentes e na seção 4, é explicado como criar relatórios dinâmicos e finalmente são feitas algumas considerações.

2. Herança de formulários

2.1. Criação do formulário pai

Um dos objetivos da Orientação a Objetos é o reuso. A idéia central é criar componentes de software que possam ser reutilizados no próprio projeto ou em projetos futuros. Quando se pensa no tempo de desenvolvimento, o conceito de reuso se torna mais importante ainda. Evitar reescrever o mesmo código se torna uma necessidade, mais do que a obediência a um conceito preestabelecido. Pensando no reuso como forma de poupar tempo, inicialmente projetou-se um padrão de interface para o sistema. Foram idealizadas como seriam as interfaces de manutenção das tabelas e as interfaces que recebem os dados necessários para a impressão dos relatórios. Será apresentado aqui apenas como foi implementada a herança dos formulários usados na manutenção das tabelas, que está apresentado na Figura 1.

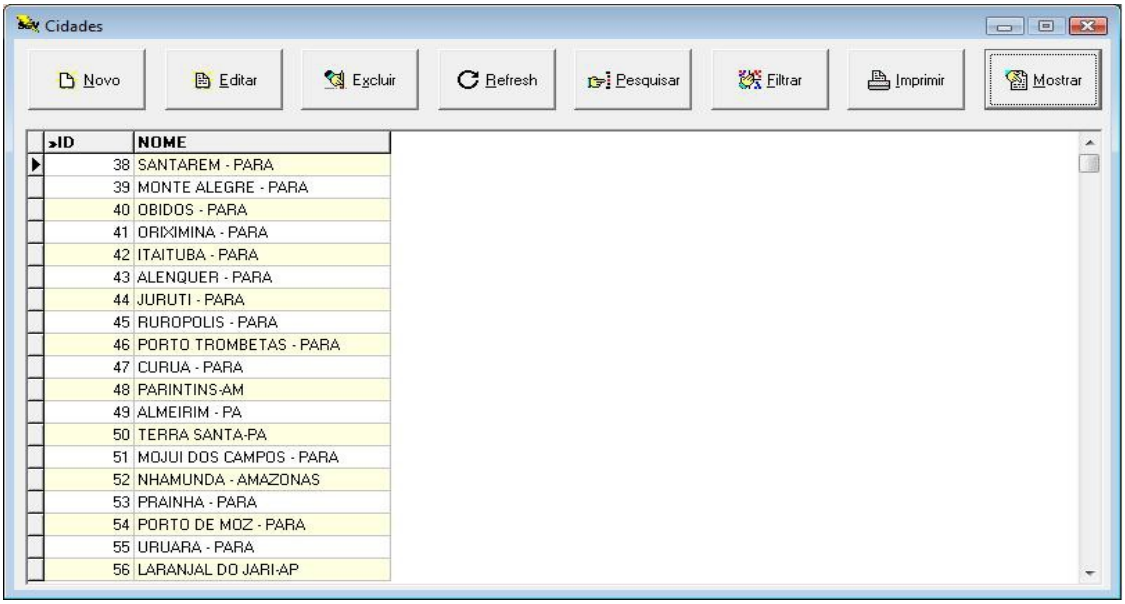

Figura 1 Formulário de manutenção de tabelas

Este formulário visualiza dados de uma tabela de banco de dados e oferece atalhos para inserir, alterar ou remover um registro, entre outros.

Sendo um formulário genérico, ele deve permitir a definição dinâmica de uma tabela, pois cada formulário derivado estará associado com uma tabela diferente. Por isso definiu-se uma propriedade denominada *Table*, conforme a Figura 2. Outras propriedades foram declaradas, mas o foco será apenas na propriedade *Table*.

```
private
  { Private declarations }
  fArquivoIni: TIniFile;
  fTopico: string;
  fLista: TStringList;
  fTable: TIBTable;
  function GetTable: TIBTable;
  procedure SetTable(NewTable: TIBTable);
  function GetArquivoIni: TIniFile;
  procedure SetArquivoIni(NewArquivoIni: TIniFile);
  function GetTopico: string;
  procedure SetTopico(NewTopico: string);
  function GetLista: TStringList;
  procedure SetLista(NewLista: TStringList);
public
  { Public declarations }
  property Table: TIBTable read GeTTable write SetTable;
  property ArquivoIni: TIniFile read GetArquivoIni write SetArquivoIni;
  property Topico: string read GetTopico write SetTopico;
  property Lista: TStringList read GetLista write SetLista;
```

Figura 2 Declaração da propriedade *Table*

Para declarar uma propriedade, é definido um campo na seção *private.* Nesta seção são informados os campos acessíveis apenas na própria *unit*. Em geral os campos são declarados com o nome da propriedade precedido da letra *f*. Por isso o campo recebeu o nome *fTable*. Na mesma seção são declarados os métodos *get* e *set.* Esses métodos leem e escrevem o valor da propriedade. Finalmente, na seção *public*, são declaradas a propriedade e seus métodos *get* e *set*, usando a palavra chave *property.*

Os dados da tabela são visualizados em um componente *TDBGrid*. Para isso é necessário um *TDataSource*, que está associado à propriedade *Table*. Após definir o campo *fTable,* ele é usado em todos os métodos que necessitem manipular a tabela de dados. Na Figura 3 é mostrado o código

referente à busca usando um campo genérico da tabela, implementado para o botão Pesquisar.

```
procedure TMostrarDados.BitBtn4Click(Sender: TObject);
var
  Valor: string;
begin
  Valor := InputBox('Pesquisa',grdDados.Fields[grdDados.SelectedIndex].DisplayLabel+' com','');
  fTable.Locate(grdDados.SelectedField.FieldName,Valor,[loPartialKey, loCaseInsensitive]);
end;
```

Figura 3 Método do botão Pesquisar

Este método faz busca pelo campo da tabela que estiver selecionado na *TDBGrid,* e chama o método *Locate* de *fTable*. Ou seja, *Locate* será chamado para a tabela que tiver sido atribuída à propriedade *Table*. Cada botão da barra de botões chama um método correspondente em função de *fTable.* Assim, o formulário e sua *unit* correspondente são criados.

2.2. Herdando o formulário

Depois que o formulário pai está criado, pode-se herdar o mesmo. O procedimento é simples: Pode-se usar o menu **File > New > Other...**, ou clicar direto sobre o botão **New items** da barra de ferramentas. Seleciona-se a aba referente ao projeto e em seguida o formulário pai que foi criado. Na parte inferior dessa caixa há três botões de rádio: **Copy**, **Inherit** e **Use**. Escolhe-se **Inherit** e pressiona-se o botão **OK**.

Tem-se agora um novo formulário com todos os objetos existentes no formulário pai. Os métodos criados anteriormente também são herdados, como pode ser visto na Figura 4.

```
procedure TCidade.BitBtn6Click(Sender: TObject);
begin
  inherited;
  if relatorio then
  begin
     frmRelatorio.PrintFast1.HeaderData.Left2 := 'Cidades';
     frmRelatorio.Caption := 'Relatório de cidades';
     frmRelatorio.Show;
  end;
end;
```

Figura 4 Método do botão Imprimir.

Neste método, assim como em todos do formulário filho, na primeira linha de código aparece a palavra reservada **inherited**, indicando a herança de código. A variável *relatorio* é declarada e inicializada no formulário pai.

Alguns cuidados são necessários. Qualquer modificação feita no formulário pai será refletida em todos os formulários filhos herdeiros dele. Mudanças no formulário filho refletirão apenas neste.

Para concluir é mostrado parte do código que chama o formulário filho na Figura 5.

```
begin
  Lista.Clear;
  Lista.Add('ID_CIDADE');
  Lista.Add('NOME');
  if TCidade(Application.FindComponent('Cidade')) = nil then
     Cidade := TCidade.Create(Application);
  Cidade.Table := dmWinSgv.tbCidade;
  Cidade.Topico := 'CIDADE';
  Cidade.Lista := Lista;
  Cidade.ArquivoIni := IniFile;
```

Figura 5 Parte do código que prepara a chamada ao formulário herdeiro

No exemplo usado, o formulário filho foi denominado *Cidade*. Antes de chamar o método **onShow**, a propriedade *Table* é inicializada com o objeto *tbCidade*. *tbCidade* é o componente *TDataSet* no *DataModule* que foi chamado *dmWinSgv* que está associado à tabela de banco de dados *Cidade*.

A solução apresentada inclui o formulário pai como pertencente ao projeto que está sendo desenvolvido. Essa é uma solução que facilita a herança no projeto atual. Se for necessário reutilizar o formulário em projetos futuros pode-se adicioná-lo ao repositório. Para fazer isso, simplesmente seleciona-se o menu local do formulário que se deseja adicionar e escolhe-se a opção **Add to Repository...** Essa opção irá solicitar um nome, a aba onde se deseja inserir e um ícone. Para esse caso pode-se selecionar a aba **Forms**. Depois de tudo definido basta confirmar. Quando se desejar herdar o formulário pode-se pressionar o botão **New items**, em seguida na aba **Forms** e depois de selecionar o modelo desejado, selecionar a opção **Inherit** e confirmar.

3. Criação dinâmica de componentes

No formulário discutido na seção anterior existe um botão denominado *Mostrar*. Esse botão exibe uma janela que permite ao usuário selecionar quais as colunas da tabela que serão visualizadas na *TDBGrid*. Sua aparência é vista na Figura 6.

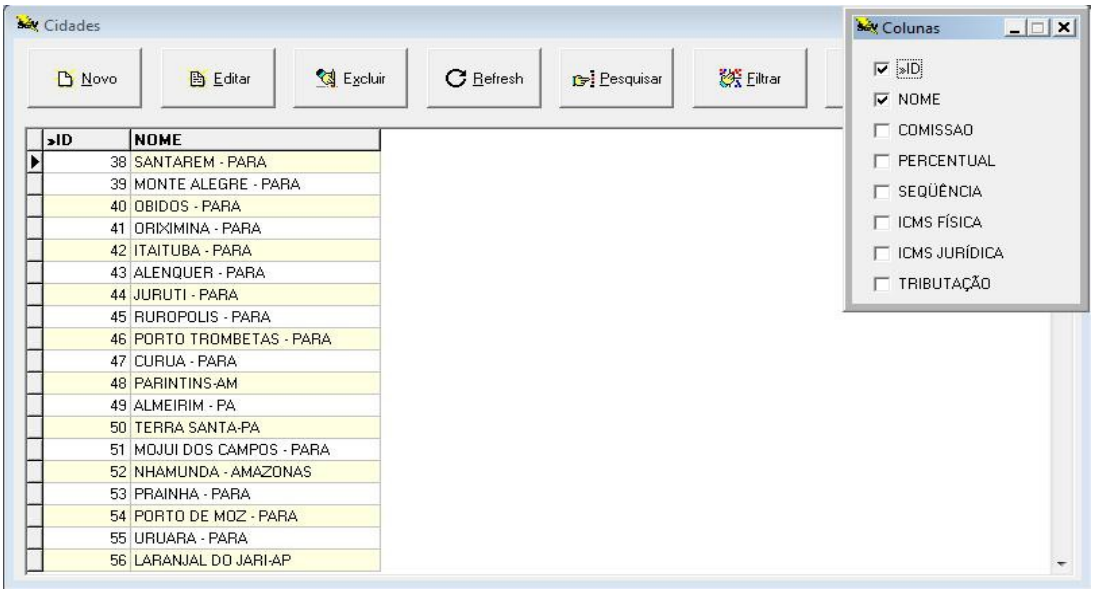

Figura 6 Janela que seleciona colunas da tabela

Esse é outro formulário herdado. No entanto, além da herança, existe outra técnica utilizada que possibilita a sua generalização, a criação dinâmica de componentes. Como é necessário apresentar as colunas das tabelas, e cada tabela possui nomes e quantidade de colunas diferentes umas das outras, não seria possível definir esses componentes estaticamente. A não ser que fosse criado um formulário para cada tabela. A solução dada consiste em herdar um único formulário para o projeto inteiro, e a cada vez que ele for chamado, os componentes *TCheckBox* são instanciados dinamicamente, um para cada coluna da tabela. Na Figura 7 é mostrado o código que implementa a criação dos componentes.

```
procedure TMostrarCampos.FormShow(Sender: TObject);
var
   Coluna : integer;
   NomeColuna : TCheckBox;
   Topo : integer;
begin
   Topo := 15;
   Mostrar := fArquivoIni.ReadString(fTopico,'Mostrar','');
   SetLength(vetorCheckBox, fTable.FieldCount);
   for Coluna := 0 to fTable.FieldCount - 1 do
   begin
     NomeColuna := TCheckBox.Create(Self);
     NomeColuna.Parent := Self;
     NomeColuna.Left := 15;
     NomeColuna.Top := Topo;
     NomeColuna.Width := 140;
     NomeColuna.Caption := fTable.Fields[Coluna].DisplayName;
     NomeColuna.Tag := Coluna;
     Topo := Topo + 25;
     NomeColuna.Visible := True;
     NomeColuna.Name := 'CheckBox'+IntToStr(Coluna);
     vetorCheckBox[Coluna] := NomeColuna;
     if copy(Mostrar,Coluna + 1,1) = 'T' then
        NomeColuna.Checked := True
     else
        NomeColuna.Checked := False;
     NomeColuna.OnClick := ClicaCheckBox;
   end;
   if Topo + 25 > 445 then
      Height := 445
   else
      Height := Topo + 25;
end;
```

Figura 7 Código de criação dos componentes

Nesse formulário também foi definido um campo *fTable* que recebe a tabela do banco de dados. Foi declarado um *array* dinâmico, chamado *vetorCheckBox*. É um *array* que irá armazenar os *TCheckBox* criados, que deve ser declarado como:

```
vetorCheckBox : array of TCheckBox;
```

No método da Figura 7 o tamanho deste *array* é definido através do procedimento *SetLength*(), que recebe como argumento a quantidade de campos da tabela. Desta forma a dimensão do vetor fica dependente do número de campos da tabela.

No laço *for* é onde ocorrem as instanciações dos componentes. *Create(Self)* cria propriamente dito o objeto. Em seguida são definidas algumas propriedades necessárias para o componente. A propriedade *Top* recebe um valor diferente a cada objeto criado. É dessa propriedade que depende a posição vertical do componente. A propriedade *DisplayName* do campo *TField* é definido como o *Caption* do *TCheckBox*. Os campos *TField* são definidos no componente *TTable.* Em geral a propriedade *Tag* de um componente é irrelevante, mas neste caso ela foi usada para manter o número de ordem do campo na tabela. Isso vai ser útil em ações futuras. A caixa de verificação é criada com uma marca se o usuário do sistema tiver escolhido o campo correspondente para ser visualizado na *TDBGrid*. Os campos marcados pelo usuário ficam armazenados em um arquivo INI. Finalmente, à medida que vão sendo inseridos novos *TCheckBox* no formulário, este precisa ser redimensionado para acomodar todos, e evitar a barra de rolagem vertical. Por isso a propriedade *Height* do formulário é alterada para o valor de *Topo* + 25, sempre esse valor ultrapassar 445, que é a altura inicial da janela. *Topo* é a posição vertical do último componente que seria adicionado. Assim, a altura da janela fica sempre proporcional á quantidade de caixas de verificação inseridas. De qualquer forma, o crescimento vertical da janela é limitado.

4. Relatórios dinâmicos em impressora matricial

O formulário apresentado na seção 2, tem um botão que comanda a impressão de um relatório dinâmico, ou seja, um relatório que é construído em tempo de execução. Esse relatório imprime as mesmas colunas e linhas que são visualizadas na *TDBGrid*. Além do botão *Mostrar*, que seleciona colunas, existe o botão *Filtrar* que seleciona linhas a serem visualizadas, como mostra a Figura 8. O método implementado no botão Filtrar tem um grau de complexidade considerável.

Figura 8 Janela para filtrar linhas da tabela

O campo selecionado na *TDBGrid* é o mesmo campo usado no filtro. O usuário pode selecionar um operador de comparação válido para a linguagem SQL (=, >, <, e até like). É possível criar uma expressão juntando várias condições através dos operadores lógicos AND ou OR. Não será comentado o código do filtro, pois o foco é o relatório.

A complexidade do relatório está em determinar a posição onde cada coluna será impressa, pois isso depende do tipo de dado da coluna e do rótulo da coluna, e ter cuidado especial para que a largura da linha a ser impressa não seja maior que a largura do papel. Esta é a limitação do relatório. O código que faz essa verificação está na Figura 9.

```
for Coluna := 0 to fTable.FieldCount - 1 do
   if copy(Mostrar, Coluna + 1, 1) = 'T' then
   begin
      if (fTable.Fields[Coluna].DataType = ftString) then
         if (fTable.Fields[Coluna].Size > Length(fTable.Fields[Coluna].DisplayLabel)) then
            Tamanho := Tamanho + fTable.Fields[Coluna].Size
         else
            Tamanho := Tamanho + Length(fTable.Fields[Coluna].DisplayLabel);
      if (fTable.Fields[Coluna].DataType = ftDateTime) then
         if (10 > Length(fTable.Fields[Coluna].DisplayLabel)) then
            Tamanho := Tamanho + 10
         else
            Tamanho := Tamanho + Length(fTable.Fields[Coluna].DisplayLabel);
      if (fTable.Fields[Coluna].DataType = ftInteger) then
         if (7 > Length(fTable.Fields[Coluna].DisplayLabel)) then
            Tamanho := Tamanho + 7
         else
            Tamanho := Tamanho + Length(fTable.Fields[Coluna].DisplayLabel);
      if (fTable.Fields[Coluna].DataType = ftFloat) or (fTable.Fields[Coluna].DataType = ftBCD) then
         if (12 > Length(fTable.Fields[Coluna].DisplayLabel)) then
            Tamanho := Tamanho + 12
         else
            Tamanho := Tamanho + Length(fTable.Fields[Coluna].DisplayLabel);
      Quantidade := Quantidade + 1;
   end;
if (Tamanho + Quantidade - 1 > 132) then
begin
   Application.MessageBox('Tamanho da linha excede 132 colunas','Alerta',MB_ICONEXCLAMATION OR MB_OK);
   relatorio := false;
```

Figura 9 Código que verifica o tamanho da linha a ser impressa

A variável *Tamanho* recebe o tamanho total da linha. Pode-se observar que, dependendo, do tipo de dado da coluna, especificado por *DataType*, e do rótulo da coluna, dado por *DisplayLabel*, os valores acrescentados a *Tamanho* são diferentes. O sistema limita a impressão em 132 colunas.

Este código usa tipos enumerados *TFieldType*, que identificam os tipos de dados dos campos: *ftString*, *ftBDC*, *ftDateTime*, *ftFloat*, etc.

Os relatórios deste sistema utilizam um componente de terceiros que imprime em impressoras matriciais. Pode parecer algo obsoleto, mas no Brasil ainda é necessário imprimir notas fiscais em cinco vias. E isso as impressoras a jato de tinta ou laser não fazem. Desde 2008 está sendo implantado o sistema de notas fiscais eletrônicas, o que pode decretar o fim da era das matriciais no mercado brasileiro.

Os relatórios são escritos usando código, não existe nada visual. Isso pode tornar mais fácil a tarefa de escrever um relatório dinâmico. O código do relatório pode ser visualizado nas Figuras 10 e 11.

No componente utilizado para implementar os relatórios existe uma propriedade chamada *ColHeader*, que define os rótulos das colunas que serão impressas. A primeira tarefa do código, mostrado na Figura 10, é definir esses rótulos. Pode-se notar que é semelhante ao código da Figura 9, mas desta vez, ele determina a posição onde o rótulo será impresso.

Os tipos de dados determinam as posições e a propriedade *DisplayLabel* define o rótulo a ser impresso. Na Figura 12 é mostrada uma prévia do relatório.

```
PrintFast1.HeaderData.Title1       := '';
PrintFast1.HeaderData.ColHeader1 := '';
Mostrar := fArquivoIni.ReadString(fTopico,'Mostrar','');
Posicao := 0;
for Coluna := 0 to fTable.FieldCount - 1 do
  if copy(Mostrar, Coluna + 1, 1) = 'T' then
  begin
     PrintFast1.HeaderData.ColHeader1 := PrintFast1.HeaderData.ColHeader1 + fTable.Fields[Colun
     if (fTable.Fields[Coluna].DataType = ftString) then
        if (fTable.Fields[Coluna].Size > Length(fTable.Fields[Coluna].DisplayLabel)) then
           Posicao := Posicao + fTable.Fields[Coluna].Size + 1
        else
           Posicao := Posicao + Length(fTable.Fields[Coluna].DisplayLabel) + 1;
     if (fTable.Fields[Coluna].DataType = ftDateTime) then
        if (10 > Length(fTable.Fields[Coluna].DisplayLabel)) then
           Posicao := Posicao + 11
        else
           Posicao := Posicao + Length(fTable.Fields[Coluna].DisplayLabel) + 1;
     if (fTable.Fields[Coluna].DataType = ftInteger) then
        if (7 > Length(fTable.Fields[Coluna].DisplayLabel)) then
           Posicao := Posicao + 8
        else
           Posicao := Posicao + Length(fTable.Fields[Coluna].DisplayLabel) + 1;
```

Figura 10 Código para imprimir os rótulos das colunas

Na Figura 11 o código para imprimir as linhas detalhe do relatório utiliza o método *ImprString* do componente usado. Esse método recebe a cadeia de caracteres a ser impressa, a posição e uma indicação se é fim de linha ou não.

```
while not EOF do
begin
   Posicao := 0;
   for Coluna := 0 to fTable.FieldCount - 1 do
     if copy(Mostrar, Coluna + 1, 1) = 'T' then
     begin
        if (fTable.Fields[Coluna].DataType = ftString) then
        begin
           if not IsNull(fTable.Fields[Coluna].Value) then
              PrintFast1.ImprString('',fTable.Fields[Coluna].AsString,Posicao,false);
           if (fTable.Fields[Coluna].Size > Length(fTable.Fields[Coluna].DisplayLabel)) then
              Posicao := Posicao + fTable.Fields[Coluna].Size + 1
           else
              Posicao := Posicao + Length(fTable.Fields[Coluna].DisplayLabel) + 1;
        end;
        if (fTable.Fields[Coluna].DataType = ftDateTime) then
        begin
           if not IsNull(fTable.Fields[Coluna].Value) then
              PrintFast1.ImprString('',DateToStr(fTable.Fields[Coluna].Value),Posicao,false);
           if (10 > Length(fTable.Fields[Coluna].DisplayLabel)) then
              Posicao := Posicao + 11
           else
              Posicao := Posicao + Length(fTable.Fields[Coluna].DisplayLabel) + 1;
        end;
```

Figura 11 Código que imprime a linha detalhe

Na prévia do relatório apresentada na Figura 12 é possível notar que a coluna *Nome* está ordenada alfabeticamente. A ordem é determinada pressionando-se o título da coluna correspondente na *TDBGrid*. A marca >> indica que coluna está ordenada.

```
18/10/2009 11:28:16
Cidades

ID      »NOME                          COMISSAO TRIBUTAÇÃO
-------------------------------------------------------------------------------
43      ALENQUER - PARA                A
49      ALMEIRIM - PA                  A
60      ALTAMIRA - PARA                A
69      ALTER DO CHAO - PA             A
961     ANANINDEUA-PA                  A
911     AVEIRO-PA                      A
70      B VISTA DO CUCARI              A
65      BARREIRINHA - AM               A         0
67      BELEM - PARA                   A
66      BELTERRA-PA                    A
889     BENEVIDES-PA                   A
83      BOA ESPERANCA/STM - PA         A
958     BRACO DO NORTE-SC              A
74      BRASIL NOVO-PA                 A
72      BREVES-PA                      A
965     CABEDELO-PB                    A
84      CAMPO MOURAO - PR              A
79      CAMPO VERDE - PA               A
895     CARACOL                        A
833     COARI-AM                       A         0
47      CURUA - PARA                   A
869     CURUAI (LAGO GRANDE) - PA      A
59      FARO-PARA                      A
71      FORTALEZA -CE                  A         0
803     GURUPA-PA                      A
42      ITAITUBA - PARA                A
885     ITAJAI-SC                      A
81      ITUMBIARA - GO                 A         0
```

Figura 12 Prévia de um relatório

5. Considerações finais

Uma aplicação pode ser desenvolvida de forma trivial, sem a utilização das técnicas apresentadas aqui. Isso levaria a muita repetição de código, um número maior de *units*/formulários e, em consequencia disso, um arquivo executável muitas vezes maior. Mas, pensando em reuso, o benefício para o desenvolvedor é enorme. Rotinas repetitivas deixam de ter que ser reescritas e passam a ser reusadas, diminuindo o tempo de desenvolvimento do projeto atual e de projetos

futuros. No entanto, acredita-se que o maior beneficiado com a utilização essas técnicas é o usuário. O resultado final é um sistema onde o usuário tem mais opções de customização, onde pode escolher como quer visualizar seus dados e como imprimir os relatórios, além de outros não citados no trabalho. *Delphi* usa *Object Pascal*, uma linguagem híbrida, que reúne o paradigma estruturado e o orientado a objetos. Os recursos da linguagem devem ser utilizados em beneficio do desenvolvedor e principalmente do usuário final. E o paradigma de orientação a objetos deve ser usado para cumprir esses objetivos.

6. Bibliografia recomendada

ALVES, W.P. e OLIVIERO, C.A.J. Sistema Comercial Integrado em Delphi 2005. São Paulo: Érica. 2005.

CANTU, M. Mastering Delphi 7. USA: Sybex. 2003.

SOARES, B. A. L. Aprendendo e Desenvolvendo Aplicações Para Banco de Dados com Borland Delphi 6. São Paulo: Ciência Moderna. 2006.

TEIXEIRA, S. e PACHECO, X. Borland Delphi 6 Developer's Guide. USA: Sams. 2002.