DESENVOLVIMENTO DE UMA FERRAMENTA PARA CONSTRUIR DINAMICAMENTE RELATÓRIOS NO CRYSTAL REPORTS

Luiz Felipe Stangarlin[1], Francisco Assis da Silva[2], Marcelo Vinícius Creres Rosa[2]

[1]Discente, [2]Docente da Faculdade de Informática da UNOESTE, Presidente Prudente, SP.

RESUMO

A utilização de ferramentas sempre foi necessária desde os primórdios da computação, devido ao trabalho repetitivo da tarefa ou da complexidade em se fazer manualmente. Por isso, ao longo do tempo, surgiram vários tipos de ferramentas de desenvolvimento. Nos dias de hoje, ferramentas CASE podem gerar diversos tipos de artefato como formulários, código-fonte, documentação e até diagramas. Porém, quando se trata de relatórios, o processo ainda é feito pelo desenvolvedor, que precisa criar layout e adicionar os itens que deseja, montando assim o relatório (artefato). Este trabalho descreve uma ferramenta que gera esse artefato de software, através da manipulação de objetos da biblioteca de vínculo dinâmico do Crystal Report (usado para confecção de relatórios). Para gerar o artefato são utilizados templates como base, de onde são copiadas as informações visuais, que simplificam a definição e melhoram o reuso. A arquitetura desenvolvida é modular e permite que a ferramenta seja anexada em aplicativos do desenvolvedor ou funcione conjuntamente com geradores CASE.

Palavras-chave: Artefato de software, gerador de relatório, CASE.

DEVELOPMENT OF A TOOL TO DYNAMICALLY BUILD REPORT IN CRYSTAL REPORTS

ABSTRACT

The use of tools has always been necessary since the beginning of computing, due to repetitive tasks or due to complexity when they were made manually. Because of this, there were several kinds of development tools over the time. Nowadays, CASE tools can generate several kinds of artifacts such as forms, source code, documentation and diagrams. However, the report building process is still done by the developer, where he needs to create layout and to add the items what he wants, assembling the report (artifact). This paper describes a tool that generates this software artifact, through the manipulation of objects in the Crystal Report dynamic link library (used for reporting purposes). To generate the artifact templates are used as a base from which visual information are copied, which simplify the definition and improve reuse. The developed architecture is modular and allows the tool to be attached by application developer or to work in conjunction with CASE generators.

Keywords: Software artifact, report generator, CASE.

1. INTRODUÇÃO

Desde a "crise do software" (RANDELL, 1996) em meados da década de 60, e o surgimento da engenharia do software, métodos e técnicas vem sendo desenvolvidos para o reuso e reaproveitamento de software com o intuito de evitar os tradicionais problemas que afetam o desenvolvimento de software como estouro de prazos, orçamentos, falta de qualidade, software que não realiza o esperado e alto custo de manutenção. É nesse panorama que surge a geração de artefatos[1] e o reuso de componentes.

A geração de artefatos é uma forma de reuso, pois consolida o conhecimento em um gerador (BIGGERSTAFF, 1988), este que pode ser reutilizado para gerar automaticamente diversos componentes que são necessários para confecção do software. Vários autores (KRUEGER, 1992; GRISS, 1995; FRAKES; ISODA, 1994; JACOBSON et al., 1997) ressaltam que o uso de reutilização em alto nível gera maiores benefícios do que o simples reuso em código-fonte. Segundo Levy (1986), quando o custo de desenvolver e manter manualmente a multiplicidade de artefatos é maior que o custo de desenvolver e manter um gerador, é possível obter uma redução do custo total com o uso de um gerador, apesar do investimento inicial. Conforme Czarnecki e Eisenecker (1999), o gerador de aplicação funciona quando se tem uma linha de produtos em comum, ou quando é possível modelar o conhecimento da configuração de forma que descreva os artefatos que serão criados, ou quando o gerador implementa o conhecimento da configuração.

Com esse intuito foi desenvolvido uma ferramenta que possibilite a geração dos artefatos responsáveis pela geração de relatórios de um sistema de aplicativos empresariais. Aplicativos empresariais são constituídos de entrada-processamento-saída. A entrada é obtida com o uso de formulários, enquanto que a saída é obtida por meio de relatórios. Tanto a entrada (formulários) quanto a saída (relatórios), geralmente são criados pelo desenvolvedor através do uso de ferramentas RAD (*Rapid Application Development*). Todos os componentes são unidos através de linguagens da quarta geração e linguagens de *script* (SOMMERVILLE, 2007).

RAD são ferramentas que automatizam a conversão da representação (artefato) em chamadas do sistema operacional. O artefato ainda é produzido manualmente pelo desenvolvedor, porém sem o uso da ferramenta RAD, o desenvolvedor precisaria escrever código para efetuar as chamadas do sistema operacional. O que elevaria ainda mais a quantidade de trabalho necessário por parte do desenvolvedor. Como o artefato está intimamente relacionado com a modelagem do problema, é possível gerar o artefato de forma automática quando se tem conhecimento do domínio. Desta forma o conhecimento da configuração, representado pelo esquema do banco de dados, possibilita a um gerador criar todos os artefatos de relatórios. Diminuindo o esforço necessário do desenvolvedor, que fica livre para focar em aspectos mais importantes com a modelagem do domínio.

As demais seções deste trabalho estão organizadas da seguinte maneira: na seção 2 são apresentados os passos para a geração de artefatos referentes aos termos encontrados na literatura; na seção 3 é descrito sucintamente o gerador de relatórios proposto neste trabalho, demonstrando todas as etapas para criar um arquivo do Crystal Reports, contendo o acesso ao

[1] Um artefato é qualquer item criado como parte da definição, manutenção ou utilização de um processo de software. Inclui entre outros, descrições de processo, planos, procedimentos, especificações, projeto de arquitetura, projeto detalhado, código, documentação para o usuário. Artefatos podem ou não ser entregues a um cliente ou usuário final (STAA, 2000).

esquema do banco de dados, a descrição do componente central do gerador e o sistema de *templating*, utilizado para prover reaproveitamento de partes comuns a vários relatórios; na seção 4 é apresentada a interface com o usuário demonstrando a ferramenta desenvolvida; e por fim, na seção 8 são apresentadas as conclusões e considerações finais do trabalho.

2. GERADOR DE ARTEFATOS

Segundo Donegan (2008) e Cleaveland (1988), uma abordagem detalhada para construir geradores de artefatos possui os seguintes passos:

1. Reconhecimento de domínio: reconhecer onde um gerador deve ser usado é o passo mais difícil, uma forma é pelo reconhecimento de padrões que ocorram no código ou em artefatos de mais alto nível de abstração. No caso deste trabalho, os relatórios contém alta taxa de padrões que repetem com os padrões definidos no esquema do banco de dados. Muitos relatórios são obtidos diretamente de entidades do banco de dados, como tabelas. Raros são os casos de relatórios que são arbitrariamente definidos sem que exista concretamente seu tipo, ou tabela no banco de dados, exceto quando o banco de dados foi mal projetado ou não atende aos requisitos do sistema. E, para outros casos, podem ser utilizadas visualizações virtuais que auxiliam o gerador a compreender a estrutura do banco de dados.

2. Definição de fronteiras de domínios: determinar o alcance do gerador, sabendo quais características devem ser incluídas ou excluídas. O gerador trabalha com a saída do sistema utilizando o modelo de domínio, representado no esquema do banco de dados. Possui como característica gerar os artefatos de relatórios através de um arquivo guia no formato XML (FRANCA, 2002), usando *templates* visuais como base para copiar a descrição visual do artefato que será gerado. Entende-se por descrição visual fontes, tipo de texto, formatação de caracteres, campos especiais como data/hora da emissão e até mesmo a logomarca da empresa. Os *templates* são expandidos através de cópia de acordo com o visual que está sendo gerado. De acordo com o domínio da aplicação pode-se combinar modelos da aplicação para gerar os artefatos (WEISS; LAI, 1999).

3. Definição de modelo: determinar um modelo matemático para que o gerador seja mais compreensível, consistente e completo, pois cada característica poderá ser explicada em termos do modelo. As boas práticas de engenharia do software foram utilizadas possibilitando que a ferramenta fosse mais compreensível, pois neste caso foi aproveitada a definição formal já existente no esquema do banco de dados.

4. Definição de partes fixas e variáveis: definir as partes que estarão sob o controle do projetista do sistema (variáveis) e as partes que não poderão ser alteradas (fixas). As partes variáveis correspondem geralmente à especificação do sistema, enquanto as partes fixas são assertivas fixas do domínio ou da implementação. O desenvolvedor poderá escolher quais partes do modelo (esquema do banco de dados) serão utilizadas para compor o relatório. Os *templates* podem ser substituídos pelo desenvolvedor. A ferramenta assume que o modelo foi bem construído, e que o usuário conheça o domínio dos dados, pois será questionado no momento da seleção dos relacionamentos entre as entidades do banco de dados.

5. Definição da entrada da especificação: as entradas das especificações podem ser feitas de forma interativa, em que o usuário seleciona as características desejadas por meio de escolhas em uma sequência de formulários ou menus, ou podem ser especificadas de forma

gráfica ou textual. A ferramenta suporta que a especificação seja feita de forma interativa ou textualmente através da criação de um arquivo XRPT. Esta ferramenta pode ser reutilizada como componente, pois possui uma interface de programação, permitindo ao desenvolvedor manipular a ferramenta através de código, para gerar os artefatos de relatórios (SOMMERVILLE, 2007).

6. Definição dos produtos: definir o tipo de produto do gerador, podendo ser de diversos tipos, como um programa, documentação ou dados de teste. Como já foi elucidado, esse gerador vai gerar artefatos de programa, no caso, relatórios.

7. Implementação do gerador: consiste na escrita do programa que traduz a linguagem da especificação no produto desejado, podendo ser usada uma ferramenta de desenvolvimento. O gerador foi desenvolvido na linguagem de programação C# do .Net Framework usando princípios e boas práticas da engenharia do software e princípios de reuso na construção de componentes (SOMMERVILLE, 2007). O núcleo do gerador, que possui o comportamento da ferramenta é uma biblioteca de vinculação dinâmica extensível e contém *hot-spots* para anexar comportamento extra. *Hot-spot* é um ponto que define o que é variável em um domínio de aplicação (BUSCHMANN et al., 1996).

3. DESCRIÇÃO DO GERADOR

Para a confecção da parte visual do relatório utilizam-se ferramentas do tipo do Crystal Reports, que usando uma especificação (artefato) e os dados obtidos do gerenciador de banco de dados compõe o relatório. A ferramenta desenvolvida trabalha na construção automatizada dessa parte visual, ou seja, trabalha na criação deste tipo de artefato responsável pela geração do relatório. O esquema mostrado na Figura 1 mostra o funcionamento básico da ferramenta.

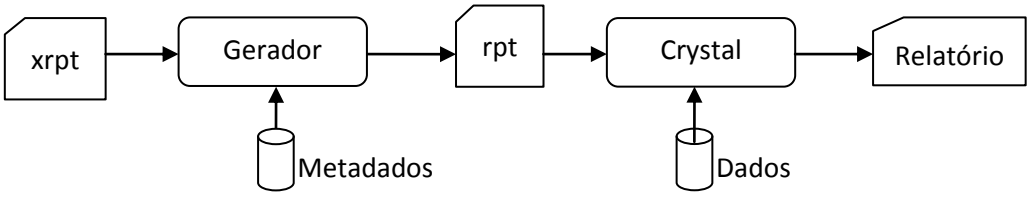


Figura 1. Esquema básico de funcionamento da ferramenta.

O esquema apresentado na Figura 1 segue uma arquitetura padrão para geradores de artefatos, segundo a literatura especializada (BALZER, 1989; BATORY; SMARAGDAKIS, 1999; MASIERO; MEIRA, 1993). Um dos objetivos é que a arquitetura seja aberta e extensível, para que o desenvolvedor possa adaptar o gerador a suas necessidades. Para isso, foi utilizado uma arquitetura de componente semelhante à descrita por Franca (2002).

A Figura 2 apresenta uma visão superficial da estrutura do gerador e de seus principais módulos, que serão descritos nas seções seguintes. O *Namespace*[2] "CRAXDRT.Report" representa o componente RDC (*Report Design Component*) do Crystal Reports (PEEK, 2008).

[2] *Namespace* é um espaço lógico onde se agrupam várias classes que funcionam conjuntamente ou se relacionam ao mesmo assunto e que estão dentro de um mesmo módulo ou biblioteca de vinculação dinâmica.

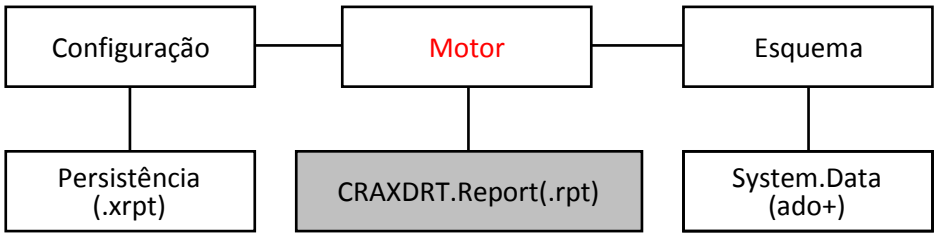


Figura 2. Diagrama Estrutural do gerador.

O gerador consome um arquivo denominado XRPT e produz um artefato do Crystal Reports, que é um arquivo de formato binário e possui a extensão RPT (PEEK, 2008).

O arquivo RPT representa a saída do gerador e para ser criado o gerador acessa a biblioteca de vinculação dinâmica RDC (PEEK, 2008) fornecida junto com o Crystal Reports. O diagrama do modelo de objeto da biblioteca está definido na Figura 3. O modelo segue um padrão orientado a objetos onde objetos são criados e métodos são acionados para gerar o artefato em formato binário.

O arquivo XRPT representa a entrada do gerador. Esse arquivo segue o padrão XML, um exemplo de sua estrutura está definido na Figura 4. Os itens que estão entre chaves "**[ ]**" são opcionais, e os itens com o asterisco "*" podem ser repetidos. A interface de usuário precisa persistir os dados que foram definidos pelo desenvolvedor, portanto, foi desenvolvido esse layout de arquivo para representar essa informação. O fato de ser um arquivo XML permite que outros programas chamem o gerador pela linha de comando somente passando esse arquivo, sem qualquer interação por interface de usuário. Caso o desenvolvedor adicione o gerador como biblioteca de vinculação dinâmica em seu programa, é possível salvar esse arquivo utilizando somente objetos, sem que seja preciso escrever código para gerar o XML.

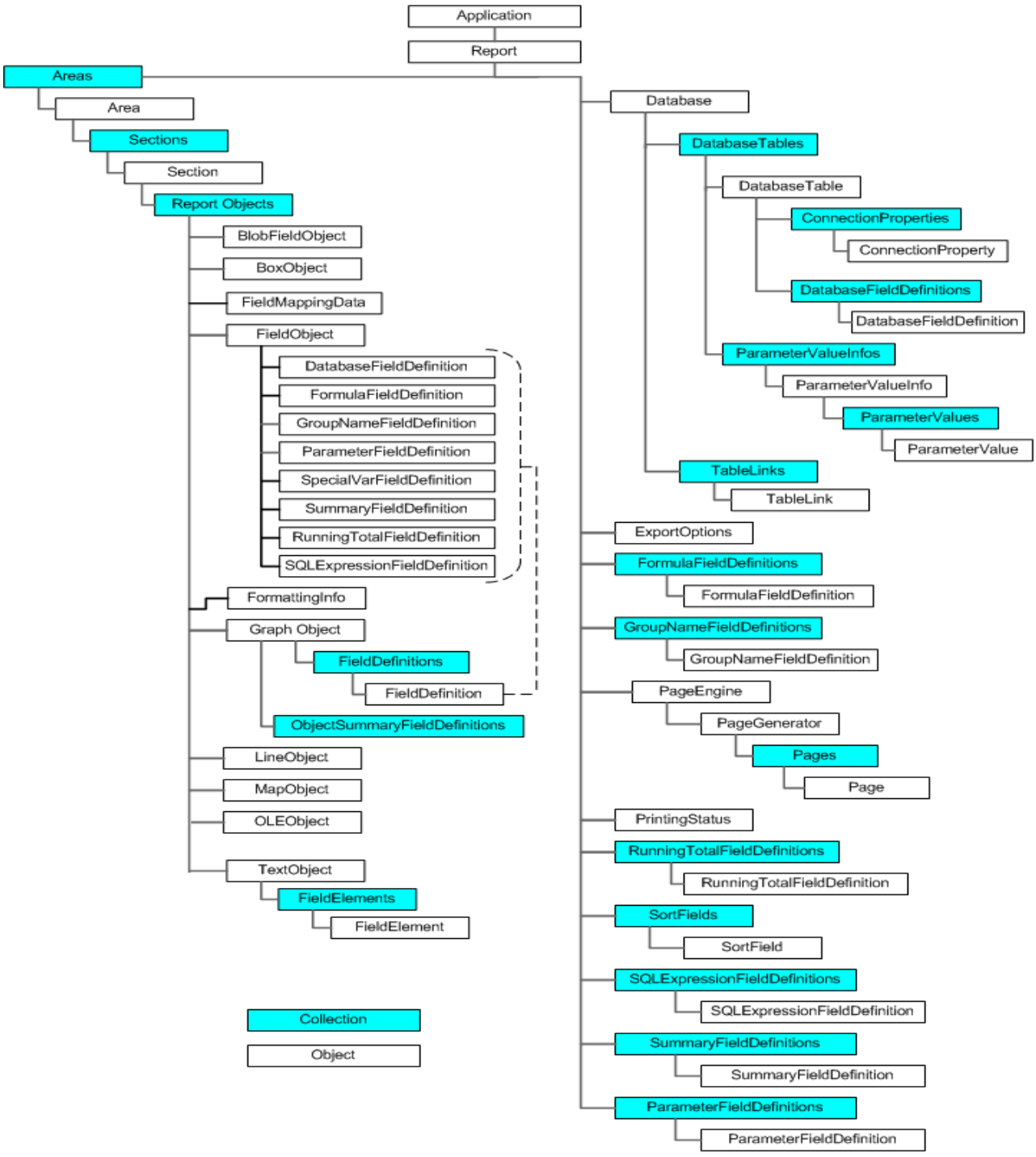


Figura 3. Crystal Reports Designer Object Model - RDC. Algumas partes não utilizadas nesse trabalho foram omitidas.

```
<?xml version="1.0" encoding="utf-8"?>
<Gerador>
  <BancoDados>
    <Conexao>
      <DataSource>.\sqlexpress</DataSource>
      <PersistSecurityInfo>Boolean</PersistSecurityInfo>
      <UserID></UserID>
      <Password></Password>
      <InitialCatalog>NomeBancoDados</InitialCatalog>
      <IntegratedSecurity>Boolean</IntegratedSecurity>
    </Conexao>
    <Colunas><Coluna*>NomeColuna</Coluna></Colunas>
    <Tabelas><Tabela*>NomeTabela</Tabela></Tabelas>
    <Relacoes><Relacao*>NomeRelacao</Relacao></Relacoes>
  </BancoDados>
  [<ColunaLargura>
    <Retangulos>
      <Retangulo*>
        <Colunas><Coluna*>NomeColuna</Coluna></Colunas>
        <Auto>Boolean</Auto><Visivel>Boolean</Visivel><Percent>Boolean</Percent>
      </Retangulo>
    </Retangulos>
  </ColunaLargura>]
  <Posicionamento>
    <Esquerda>Double</Esquerda><EspacoHorizonatal>Double</EspacoHorizonatal>
    <EspacoVertical>Double</EspacoVertical><Justificado>Booelan</Justificado>
    <Tipo>Tabela or Formulario</Tipo>
    [<Tabela><Quebrar>Boolean</Quebrar></Tabela>]
    [<Formulario><Linhas>Integer</Linhas><Colunas>Integer</Colunas></Formulario>]
  </Posicionamento>
  <Cabecalho>
    <Titulo>TituloRelatorio</Titulo>
    <Secao><Nome>ReportHeader</Nome><Estilo>LayoutFormatacao</Estilo></Secao>
  </Cabecalho>
  <Detalhe>
    <Secao><Nome>Details</Nome><Estilo>LayoutFormatacao</Estilo></Secao>
  </Detalhe>
  <Agrupamentos>
    <Grupo*>
      <Estilo>Simples</Estilo>
      <Coluna>NomeColunaAgrupar</Coluna>
      <Condicao>Qualquer</Condicao>
      <Ordem>Ascendente</Ordem>
      <Posicionamento>Igual o definido Anteriormente</Posicionamento>
      <Colunas><Coluna*>NomeColunaExibir</Coluna></Colunas>
    </Grupo>
  </Agrupamentos>
  <Rodape>
    <Secao><Nome>ReportFooter</Nome><Estilo>LayoutFormatacao</Estilo></Secao>
  </Rodape>
</Gerador>
```

Figura 4. Exemplo de um arquivo XRPT.

3.1. Criação de um arquivo do Crystal Reports

Para criar um arquivo RPT é necessário manipular a biblioteca RDC, a Figura 5 contém um exemplo de uso dessa biblioteca. Este exemplo cria um relatório com um titulo e um campo de texto para cada coluna de cada tabela especificada na lista "Tables" (linha 22), e não leva em consideração o posicionamento e a formatação. Também é demonstrado como a conexão com o banco de dados deve ser inicializada para que funcione corretamente, de

acordo com o manual do RDC (PEEK, 2008). Foram destacados em negrito na Figura 5 as classes e métodos que correspondem ao RDC.

```
1 public void Gerar(string ArquivoDestino)
2 {
3   ApplicationClass app = new ApplicationClass();
4   Report report = app.NewReport();
5
6   System.Data.SqlClient.SqlConnectionStringBuilder conn = new
7     System.Data.SqlClient.SqlConnectionStringBuilder(Tables[0].ConnectionString);
8   String curdir =
9    System.Reflection.Assembly.GetAssembly(typeof(ApplicationClass)).Location;
10  System.IO.FileInfo path = new System.IO.FileInfo(curdir);
11    String dll = System.IO.Path.Combine(path.DirectoryName, "p2soledb.dll");
12  System.Data.OleDb.OleDbConnectionStringBuilder cns = new
13    System.Data.OleDb.OleDbConnectionStringBuilder();
14  cns.Provider = "SQLOLEDB.1";
15  cns.DataSource = conn.DataSource;
16  cns.PersistSecurityInfo = conn.PersistSecurityInfo;
17  cns.Add("User ID", conn.UserID);
18  cns.Add("Password", conn.Password);
19  cns.Add("Initial Catalog", conn.InitialCatalog);
20  cns.Add("Integrated Security", conn.IntegratedSecurity ? "SSPI" : null);
21
22  foreach (Table table in Tables)
23  {
24    report.Database.AddOLEDBSource(cns.ConnectionString, table.Name);
25  }
26  report.ReportTitle = Header.Texto;
27  report.ReportComments = "Generated by tool";
28  //area 1 report header, 2 page header, 3 details, 4 page footer, 5 report footer
29  report.Areas[1].Sections[1].AddTextObject(report.ReportTitle, 0, 0, null);
30
31  foreach (Table table in Tables)
32  {
33    DatabaseTable crTable = report.Database.Tables[Tables.IndexOf(table) + 1];
34    int left = 0;
35    foreach (Column column in table.Columns)
36    {
37      DatabaseFieldDefinition crField =
38        crTable.Fields[table.Columns.IndexOf(column) + 1];
39      TextObject label = report.Areas[2].Sections[1].AddTextObject(column.Name,
40        left, 0, null);
41      FieldObject field = report.Areas[3].Sections[1].AddFieldObject(crField,
42        left, 0);
43      label.Width = 1000;
44      field.Width = 1000;
45      left += 1200;
46    }
47  }
48  report.Save(ArquivoDestino);
49}
```

Figura 5. Gerando um RPT.

3.2. Acesso ao esquema do banco de dados

Para a geração automatizada dos artefatos à partir do modelo é necessário o conhecimento do esquema do banco de dados, para isso foi escolhido utilizar o mecanismo do provedor de dados OLEDB (NILSEN, 2009). Este mecanismo funciona através de uma interface de programação e permite que qualquer banco de dados para o qual existe o provedor OLEDB seja utilizado. Porém, devido a utilização direta de um provedor OLEDB ser complexa, optou-se por utilizar uma abstração provida pela biblioteca

ADO+ disponível no .Net Framework da Microsoft (STELLMAN, 2008). Dessa forma, foi desenvolvido o seguinte código para recuperar informações do esquema de um banco de dados, conforme o Figura 6. Foram destacados em negrito as classes e os métodos do ADO+ na Figura 6.

```
1 protected DataTable LoadSchema(Guid collectionguid, string collectionName,
2   string RestrictionName, string RestrictionValue)
3 {
4   using (OleDbConnection connection = new
5       OleDbConnection(this.FindConnectionString().OleDb.ToString()))
6   {
7     connection.Open();
8     DataTable items;
9     if (RestrictionValue == null || RestrictionValue == null)
10    {
11        items = connection.GetOleDbSchemaTable(collectionguid, null);
12    }
13    else
14    {
15      DataTable metacollections = connection.GetSchema();
16      DataTable restrictions = connection.GetSchema("Restrictions");
17      string filtercollection = "CollectionName = '" +
18        CollectionName.Replace("_", "") + "'";
19      string filterrestriction = filtercollection + " AND RestrictionName = '" +
20        RestrictionName + "'";
21      DataRow collection = metacollections.Select(filtercollection)[0];
22      DataRow restriction = restrictions.Select(filterrestriction)[0];
23      int numparam = (int)collection["NumberOfRestrictions"];
24      int ixparam = (int)restriction["RestrictionNumber"];
25      string[] param = new string[numparam];
26      param[ixparam - 1] = RestrictionValue.Split('.').Last();
27      items = connection.GetOleDbSchemaTable(collectionguid, param);
29    }
30    return items;
31  }
32}
```

Figura 6. Recuperando o esquema de um banco de dados.

A função que realmente efetua a leitura do esquema é a "GetOleDbSchemaTable" (linhas 11 e 27) e para ler as informações do esquema é necessário conhecer quais as tabelas de informações podem ser utilizadas. Cada tabela de informação é identificada por um número único global "Guid" (linha 1), um exemplo de tabela de informação seria quais colunas uma tabela possui, ou que relacionamentos uma tabela tem com outras tabelas. Ao chamar a função sem parâmetros todas as informações sobre o banco de dados são retornadas. Para resolver esse problema é feito o uso de restrições, que são um array contendo valores de restrições. A posição da restrição desejada no array deve ser calculada de acordo com o "Guid". Para efetuar esse cálculo, existe uma tabela de metadados do esquema, que é o esquema do esquema. Esta tabela é simples e contém o nome do "Guid", o número de restrições, o índice da restrição e o nome da restrição. Com base nesses dados é possível efetuar uma consulta ao esquema do banco de dados. Um exemplo de restrição seria o nome da tabela ao procurar as colunas de uma tabela. A obtenção dessas informações e a escolha da biblioteca mais adequada ao trabalho constituiu uma das etapas de maior uso de tempo devido a extensa experimentação e testes que foram necessários.

3.2.1. Representação do esquema

Após a obtenção do esquema, os metadados devem ser representados de uma forma que possibilite o fácil entendimento e uso pelo gerador. Devido a esse motivo foi modelado o conjunto de classes representado na Figura 7. Foi feito o uso do padrão de projeto Composite (GAMMA et al., 1995) que é utilizado sempre que é necessário representar elementos que são compostos por outros elementos similares. Possibilita o uso de recursão no acesso aos dados do esquema, simplificando o desenvolvimento da ferramenta. Todas as classes derivadas de "SchemaItem" representam alguma parte do esquema e o comportamento descrito anteriormente é reutilizado em cada uma das classes através da herança do método "LoadSchema", que é sobrescrito nas classes derivadas e completado com o "Guid" necessário.

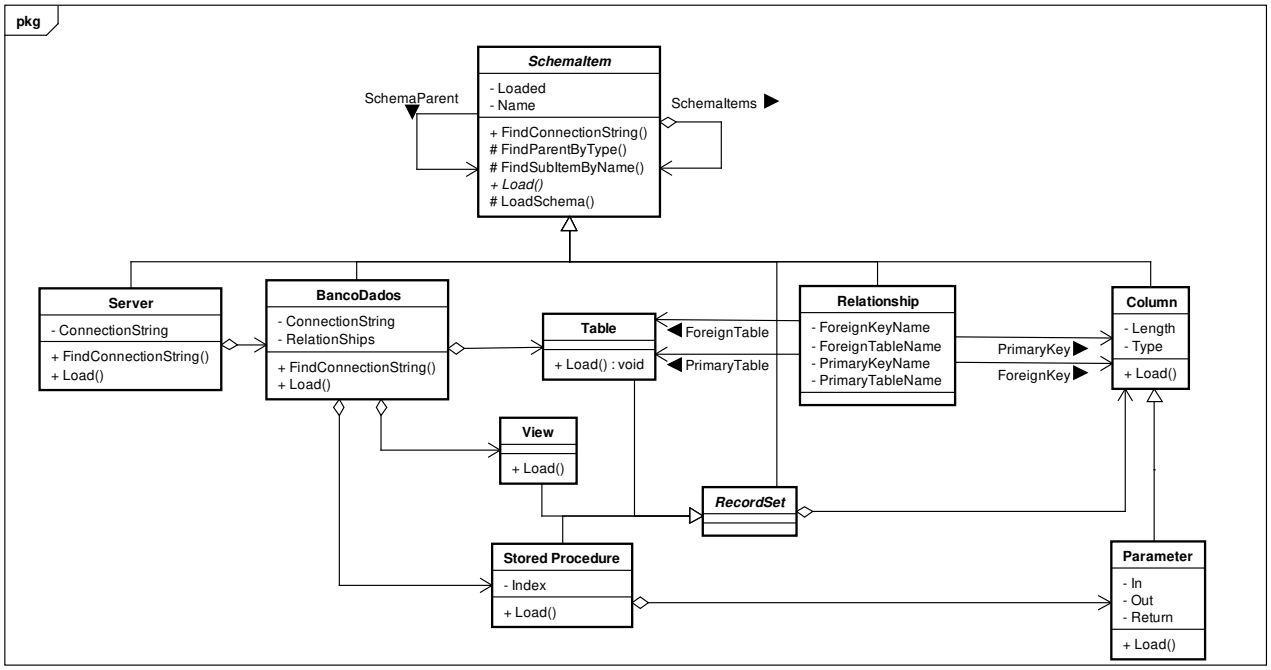


Figura 7. Representação OO do esquema do banco de dados (Gerador.Esquema).

3.3. Motor de Geração

O componente central do gerador de artefato é o motor. Ele é o componente responsável por consultar o esquema, ler as informações do XRPT e manipular o componente RDC, gerando o arquivo de saída RPT. A Figura 8 detalha as principais classes envolvidas no controle desse comportamento. O motor é implementado como uma biblioteca de vinculação dinâmica, e pode ser utilizado por qualquer linguagem de programação do .Net Framework.

A classe "Report" contém a referência ao objeto do RDC e representa o relatório que está sendo construído. O "SchemaManager" é o responsável por manter listas de objetos do esquema (Gerador.Esquema) que representam a entrada de informações no gerador. Esta lista de objetos é manipulada pela interface gráfica, pelo desenvolvedor ou pelo componente responsável pela leitura e escrita do arquivo XRPT (esse componente foi omitido na Figura 8).

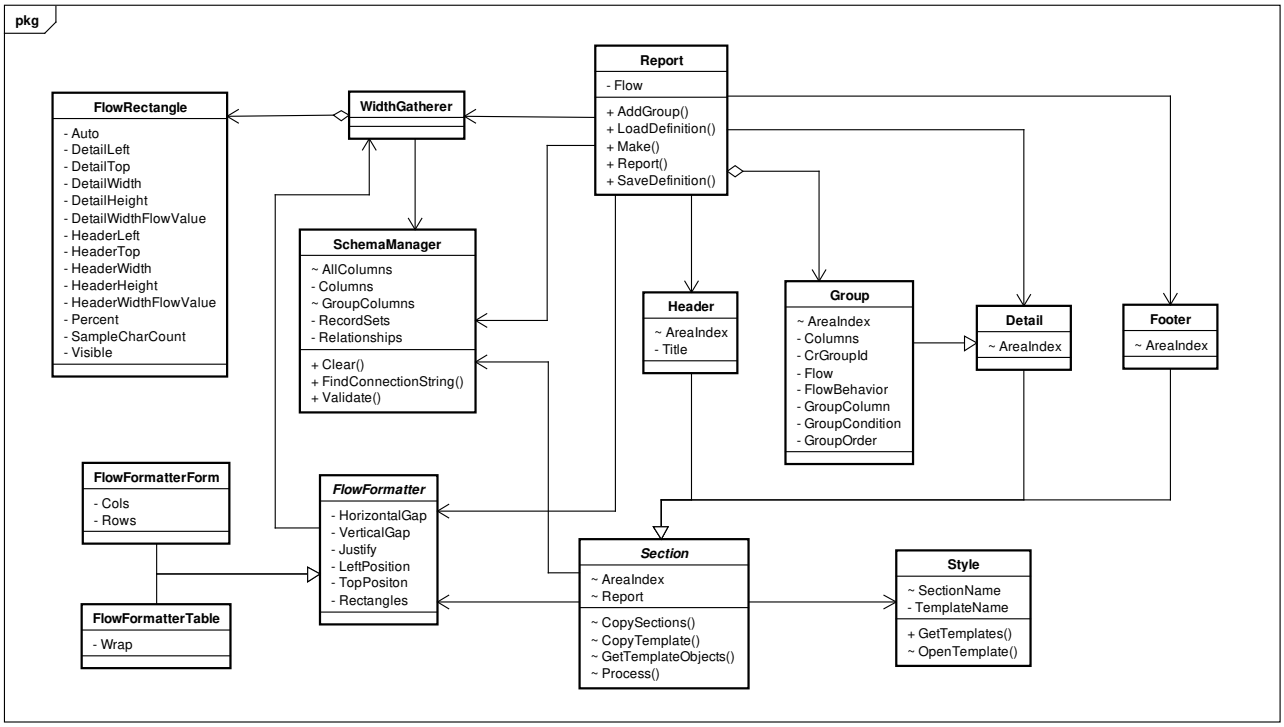


Figura 8. Diagrama de classes do motor do gerador (Gerador.Motor).

O "FlowFormatter" e derivados é o componente responsável pelo posicionamento de cada objeto na parte visual do relatório. O padrão de orientação a objetos utilizado foi o *Strategy*[3] e permite que o comportamento seja trocado conforme a necessidade do desenvolvedor. Como a classe não possui o tipificador *Sealed*[4], é possível ao desenvolvedor usar a herança para agregar novos comportamentos à ferramenta através da injeção de uma classe derivada na propriedade "FlowBehavior" da classe "Report" provendo um ponto de extensibilidade (*Hot-spot*). Todo o comportamento de posicionamento é aplicado para cada seção do relatório.

O Comportamento "FlowFormatterTable" gera uma tabela semelhante a uma tabela do Microsoft Excel, o funcionamento desse comportamento consiste basicamente em obter as colunas do "WidthGatherer", manter um contador de posição X e Y. Ao percorrer a próxima coluna, acumula-se a largura no contador X, e ao ficar maior que a largura da seção, incrementa-se o contador Y caso o "Wrap" esteja ativo, caso contrário, os retângulos são gerados com a propriedade "Visible" em falso. Então se aplica o valor de X e Y em "*Left" ou "*Top", respectivamente. O asterisco representa o posicionamento no "Header" ou "Detail", ambos devem ser calculados e no caso desse comportamento serão iguais.

O Comportamento "FlowFormatterForm" gera um formulário semelhante a um formulário de cadastro de um sistema. Em que o valor segue o nome da coluna utilizando uma linha para cada coluna. O funcionamento é semelhante ao comportamento "FlowFormatterTable", são utilizados contadores com a diferença que é necessário manter uma estrutura de dados com as larguras de cada coluna visual. E para a criação de outros comportamentos, é utilizado esse mesmo esquema, mudando somente o cálculo do "*Left" e "*Top".

[3] *Strategy* é um padrão de projeto que permite que uma família de algoritmos seja utilizada de modo independente e seletivo. Padrões de projeto para arquitetura de software são soluções de eficiência já comprovadas e amplamente utilizadas para a resolução de problemas comuns em projeto de software (GAMMA et al., 1995).

[4] *Sealed* é uma classe cuja herança foi bloqueada pelo desenvolvedor da classe, não permitindo mais que outras classes herdem da classe selada (STELLMAN, 2008).

Responsável pela amostragem estatística sobre os dados do banco de dados, o componente "WidthGatherer" é utilizado para inferir a largura das colunas em um relatório, essa informação é armazenada na coluna "SampleCharCount". O componente segue o padrão *Strategy*, e pode ser estendido através da injeção de um derivado na propriedade "WidthGatherer" do objeto "Report". Este componente é opaco e não possui nenhuma propriedade importante exceto a lista de retângulos com as larguras das colunas selecionadas no "SchemaManager". A lista de retângulos deve ser inicializada pelo "WidthGatherer", mas nem todas as propriedades do retângulo precisam ser preenchidas, deixando essa função a cargo do posicionador "FlowFormatter". A única estratégia disponível efetua o cálculo da largura de cada coluna utilizando uma amostra de 10% dos dados, calculando o máximo de caracteres na coluna e multiplicando o valor pela média de largura de cada caractere da fonte escolhida pelo estilo do *template* selecionado.

Um relatório pode conter várias seções, que são quebras lógicas ou áreas com comportamento de geração diferentes. Todo relatório sempre possui um cabeçalho, um rodapé e uma seção detalhe, que é a parte que se repete várias vezes de acordo com os dados. O relatório também possui agrupamentos, que são quebras entre a seção detalhe, e servem para agrupar várias seções detalhe sobre o mesmo argumento. Um grupo também é uma seção detalhe, e vários grupos podem ser combinados em uma hierarquia. O grupo é representado pela classe "Group" e a seção é pela classe "Section".

Uma importante propriedade da classe "Section" é o "Style". O "Style" representa o *template* visual utilizado na geração de toda a descrição visual como fontes, tipo de texto, formatação de caracteres, campos especiais como data/hora da emissão e logotipos ou figuras.

3.4. Sistema de *templating*

O sistema de *templates* do gerador é o mecanismo que garante o funcionamento das principais características da ferramenta. O sistema de *template* e posicionamento automático são responsáveis pelo ganho de tempo no desenvolvimento de relatórios, pois permite o reaproveitamento de partes comuns a vários relatórios e a parte visual da formatação.

O designer do Crystal Reports também fornece a capacidade de escolher um *template* básico, porém, só é possível utilizar da primeira vez, e ao tentar substituir o *template* depois do relatório criado, o posicionamento é perdido e deve ser refeito manualmente. Além disso, constitui uma tarefa árdua devido a ser necessário efetuar o mesmo procedimento para cada relatório do sistema.

Por causa dessa dificuldade, o sistema de *template* extensível foi desenvolvido. Diferente do sistema do Crystal, este utiliza outro arquivo RPT como *template*. E esse arquivo contém as partes que são iguais entre todos os relatórios. O sistema de *template* separa o relatório em duas camadas, que são a descrição visual básica e a descrição comportamental XRPT, permitindo um reaproveitamento mais eficiente dos esforços do desenvolvedor. Inclusive e possível que um mesmo sistema tenha um conjunto de RPTs de *template* para cada cliente, o que aumenta o ganho da ferramenta ainda mais, pois cada cliente sempre tem uma preferência visual diferente.

Para o funcionamento do gerador, o sistema de *templating* fornece métodos para que informações visuais sobre o *template* sejam obtidas. O sistema de *templating* é implementado pela classe "Style", cuja referencia é gerenciada pela classe "Section". Portanto, para cada seção

do relatório é possível utilizar um *template* diferente.

A Figura 9 contém um exemplo de um *template* visual e a Figura 10, um exemplo da expansão do *template* através da cópia de elementos. O título da coluna é copiado várias vezes em "a", e o campo do banco de dados também é copiado em "b" de acordo com dados lidos do esquema.

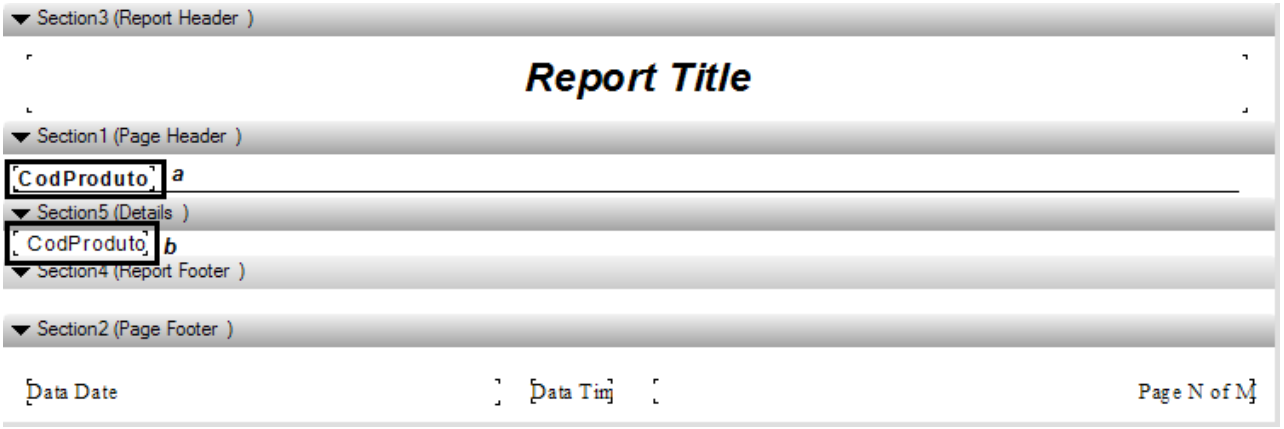


Figura 9. *Template* utilizado.

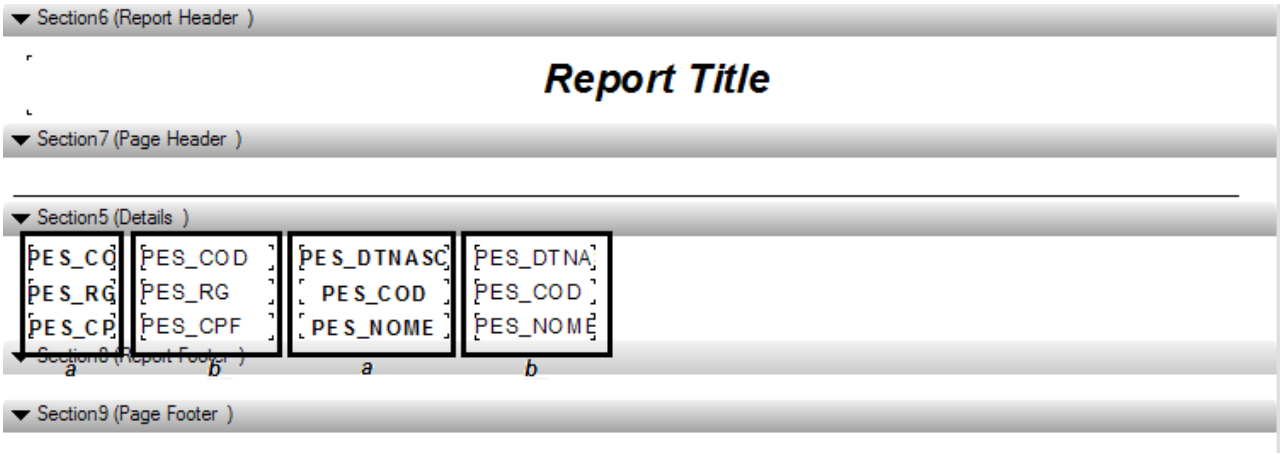


Figura 10. Expansão do *template.*

São fornecidos na classe "Style" métodos que efetuam a cópia dos objetos do *template* para o relatório novo que está sendo gerado. A cópia é efetuada chamando métodos RDC para criar novos objetos na árvore de objetos do relatório novo, depois são copiados todos os valores de propriedade do objeto do *template* para o objeto novo. Após a cópia referente aos objetos que referenciam colunas do banco de dados, as propriedades referentes à ligação com o banco de dados são definidas de acordo com dados do esquema.

O sistema de *templating* é o componente do gerador que está mais intimamente relacionado com o RDC. Caso fosse utilizada outra ferramenta de confecção de relatórios, esse seria o principal componente afetado, os outros componentes seriam as classes "Section".

4. INTERFACE COM O USUÁRIO

A Figura 11 apresenta a interface do usuário (UI). A interface representa o arquivo XRPT, com ela o usuário pode criar visualmente o relatório sem se preocupar com arquivos e formatos. A UI importa a biblioteca do motor do gerador e a manipula para construir tanto o XRPT quanto o RPT. O usuário pode criar um novo XRPT ou carregar um arquivo deste previamente armazenado, também é possível especificar onde o gerador vai armazenar o RPT.

Apesar da UI ser um executável, ela também é uma biblioteca de vinculação dinâmica, pois todas suas classes estão exportadas com o tipificador *Public*[5]. Além disso, cada objeto referente a parte do XRPT dispara eventos que podem ser usados para anexar um comportamento extra, pois também são pontos de extensibilidade. Essa é a principal diferença em relação ao *wizard* do Crystal Reports que é fechado, apesar do RDC permitir esse tipo de uso, o *wizard* não o permite. E esse conceito foi utilizado para permitir ao desenvolvedor anexar a interface de usuário do gerador em seu sistema facilmente, possibilitando que um usuário com conhecimento do domínio utilize a ferramenta também.

Outra diferença em relação ao *wizard* do Crystal Reports é o sistema de *templates* evidenciado pelas caixas de combinações "Formato", que dão acesso aos arquivos de *template* para cada seção. A interface desenvolvida é bem mais simples e fácil de operar que o *wizard* do Crystal Reports. A UI também é menos obstrutiva, já que o Crystal Reports tem um *popup modal*[6] que fica sob a tela do relatório, obstruindo a visão do relatório e não permite ver em tempo real as alterações feitas no *template*.

A Figura 12 mostra um código fonte de exemplo de utilização da biblioteca do gerador de relatórios. Nesse exemplo, é criado um relatório contendo as informações necessárias para formar o relatório demonstrado na Figura 13.

[5] *Public* permite que qualquer classe de dentro ou de fora do programa utilize a classe (STELLMAN, 2008).
[6] *popup* é uma janela que flutua sobre as outras, e *modal* é um termo utilizado em interfaces com usuário e significa que a janela bloqueia o acesso a todas as outras janelas do sistema quando é aberta.

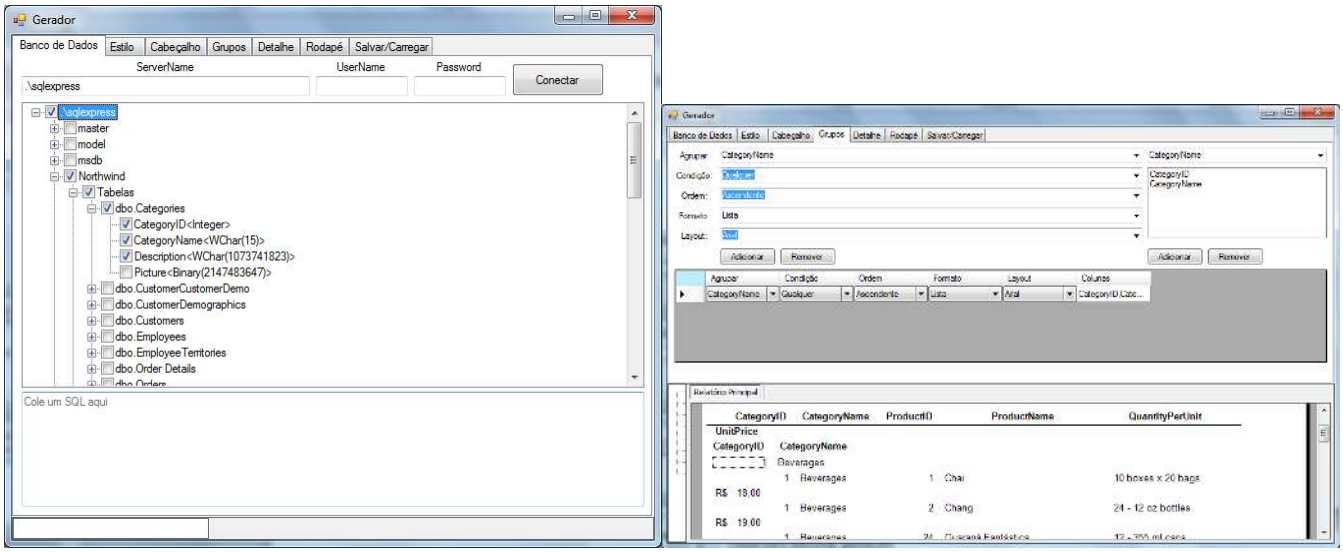


Figura 11. Interface do Usuário (UI).

```
1  Report rpt = new Report();
2  rpt.Header.Title = "Produtos";
3
4  Server server = new Server(".\\sql2008", null, null);
5  server.Load();
6  Database database = server.Databases.Find(d => d.Name == "BancoDados");
7  database.Load();
8  Table table = database.Tables.Find(t => t.Name == "dbo.Produto");
9  table.Load();
10 rpt.Schema.RecordSets.Add(table);
11 rpt.Schema.Columns.Add(table.Columns.Find(c => c.Name == "Codigo"));
12 rpt.Schema.Columns.Add(table.Columns.Find(c => c.Name == "Nome"));
13
14 rpt.Flow = FlowBehavior.Form;
15 rpt.Header.FormatStyle.TemplateName = "Arial";
16 rpt.Detail.FormatStyle.TemplateName = "Small";
17 rpt.Footer.FormatStyle.TemplateName = "Teste";
18
19 rpt.Make("report.rpt");
20 rpt.SaveDefinition("report.xrpt");
21
22 Vizualiza f2 = new Vizualiza();
23 f2.ShowDialog("report.rpt", server.ConnectionString.Sql);
```

Figura 12. Código fonte que demonstra o uso da biblioteca do Gerador.

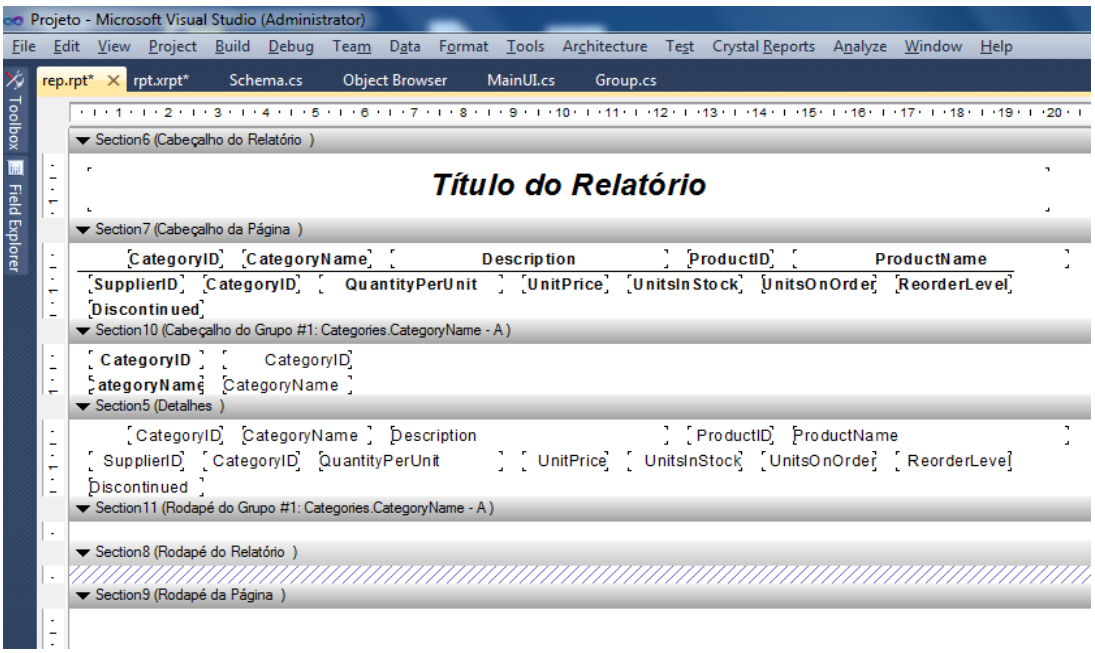


Figura 13. Relatório gerado automaticamente.

5. CONCLUSÕES

Com esse trabalho foi possível demonstrar como uma ferramenta para a confecção de relatórios pode ser utilizada como um gerador CASE através da manipulação de árvores de objetos. Apesar do alto nível na utilização dos objetos para leitura do esquema e criação do relatório, o nível de complexidade na manipulação dessas árvores de objetos é semelhante ao enfrentado pelos geradores CASE de código fonte ao trabalhar com árvores AST[7].

Ao utilizar um arquivo de padrão aberto XRPT, é possível que o Gerador seja integrado para trabalhar junto com outras ferramentas CASE que geram código fonte e formulários de entrada de dados, pois elas podem gerar o XRPT a partir de seu modelo. Também é possível integrar o gerador usando programação, pois o gerador é uma biblioteca de vinculação dinâmica.

[7] AST significa *Abstract Sintax Tree*, são estruturas de dados em formato de árvore utilizados por geradores e compiladores (FRANCA, 2002).

REFERÊNCIAS

BALZER, R. A **fifteen-year perspective on automatic programming.** Addison-Wesley/ACM Press, p.289-311, 1989.

BATORY, D.; SMARAGDAKIS, Y. **Application Generators**. John Wiley and Sons, 1999.

BIGGERSTAFF, T. J. A Perspective of Generative Reuse. **Annals of Software**, p.169-226, 1988.

BUSCHMANN, F. et al. **Pattern-Oriented Software Architecture Volume 1: A System of Patterns**. John Wiley & Sons, 1996.

CLEAVELAND, J. C. Building application generators. **IEEE Software**, v.7, n.1, p.25-33, 1988.

CZARNECKI, K.; EISENECKER, U. **Components and Generative Programming**. Acmd Sigsoft, 1999.

DONEGAN, P. M. **Geração de famílias de produtos de software com arquitetura baseada em componentes.** São Carlos, 2008.

FRAKES, S.; ISODA, S. Success factors of systematic software reuse. **IEEE Software**, v.11, n.1, p.14-19, 1994.

FRANCA, L. P. A. Uma Arquitetura Aberta para Geradores de Artefatos. **XVI Simpósio Brasileiro de Engenharia de Software**, p. 44-46, 2002.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1 ed. Addison-Wesley, 1995.

GRISS, M. Making software reuse work at hewlett-packard. **IEEE Software**, v.12, n.1, p. 105-107, 1995.

JACOBSON, I.; GRISS, M.; JONSSON, P. **Reuse-driven Software Engineering Business**. Addison-Wesley, 1997.

KRUEGER, C. Software reuse. **ACM Computing Surveys**, v.24, n.2, p.131-183, 1992.

LEVY, L. A Metaprogramming Method and Its Economic Justification. **IEEE Transactions on Software Engineering**, v.12, n.2, p.272-277, 1986.

MASIERO, P. C.; MEIRA, C. A. Development and Instantiation of a Generic Application Generator. **Journal of Systems and Software**, p.27-37, 1993.

NILSEN, P. **Microsoft SQL Server 2008 Bible**. Wiley, 2009.

PEEK, G. **Crystal Reports 2008:** The Complete Reference. McGraw-Hill Osborne Media, 2008. 50 p.

RANDELL, B. **The 1968/69 nato software engineering reports. In History of Software Engineering**. 1996.

SOMMERVILLE, I. **Software Enginnering**. 8.ed., Addison-Wesley, 2007.

STAA, A. V. **Programação Modular**. 2000.

STELLMAN, A. **Use a cabeça C#**. Rio de Janeiro: Alta Books, 2008.

WEISS, D.; LAI, C. **Software Product-line Engineering:** a Family-based Software Development. Addison-Wesley Longman Publishing Co., 1999.

Recebido em: 05/06/2013

Revisado em: 25/06/2013

Aceito em: 01/07/2013