RODRIGO MERLIN

IMPLEMENTAÇÃO DE UM MÓDULO EM JAVA PARA GERAR LISTAGENS E RELATÓRIOS EM APLICAÇÕES DESKTOP E WEB

ASSIS
2009

IMPLEMENTAÇÃO DE UM MÓDULO EM JAVA PARA GERAR LISTAGENS E RELATÓRIOS EM APLICAÇÕES DESKTOP E WEB

RODRIGO MERLIN

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Ms. DOUGLAS SANCHES DA CUNHA

Analisador: Dr. ALMIR ROGÉRIO CAMOLESI

ASSIS
2009

RODRIGO MERLIN

IMPLEMENTAÇÃO DE UM MÓDULO EM JAVA PARA GERAR LISTAGENS E RELATÓRIOS EM APLICAÇÕES DESKTOP E WEB

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Ms. DOUGLAS SANCHES DA CUNHA

Área de Concentração: JAVA, BANCO DE DADOS E IREPORT.

ASSIS
2009

AGRADECIMENTOS

Ao mestre, DOUGLAS SANCHES DA CUNHA, pela orientação e pelo constante estímulo transmitido durante o trabalho.

Aos amigos, SIMONE CARDOSO, JABES FELIPE CUNHA, VINÍCIUS DE OLIVEIRA DIAS e a todos que colaboraram direta ou indiretamente na execução deste trabalho.

Aos familiares, SEBASTIÃO MERLIN, MARIA APARECIDADE MERLIN e aos demais, que nunca deixaram de acreditar que o objetivo seria alcançado.

RESUMO

Com a finalidade de dar suporte aos programadores iniciantes no quesito gerar relatórios, os estudos foram voltados à implementação de uma classe de modo a torná-la um módulo que possa ser usado sem alterações de código ou desempenho tanto em sistemas desktop ou web, com o objetivo de gerenciar os processos entre a aplicação e a base de dados, de modo que possibilite utilizar uma única tabela que através de diferentes objetos processará de forma genérica as informações, com isso os desenvolvedores responsáveis pela etapa do sistema que gera relatórios serão beneficiados, pois podem utilizar somente uma tabela para desenvolver o layout dos seus relatórios, alem de ter a garantia da integridade da base de dados.

Palavras chave: Relatório. Genérica. Tabela.

ABSTRACT

In order to support the programmers in the query to generate reports, studies were focused on the implementation of a class to make it a module that can be used without code changes or system performance in both desktop or web, with order to manage the process between application and database, so that allows using a single table across different objects in a generic process information, thus the developers responsible for the step system that reports will benefit because they can use only one table to develop the layout of your reports, besides having the guarantee of the integrity of the database.

Keywords: Reports. Generic.Table.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Dentre as tarefas de um sistema, a mais comum é a geração de relatórios. Presente na maioria dos sistemas, mas muitas vezes não suficientemente reconhecida, esta tarefa constitui um importante etapa para a construção de um sistema.
Basicamente, o processo de geração de relatório resume-se na definição do visual e mapeamento de dados para campos dentro de um estilo definido. Nesse contexto, surgiram ferramentas comerciais com intuito de auxiliar neste processo. No passado, essa área foi completamente dominada por produtos comerciais, como o *Crystal Reports*. Com o passar do tempo tornaram-se cada vez mais robustos no que diz respeito a novas funcionalidades, como o suporte a diferentes tipos de fontes de dados. Porém, o que se vê hoje é o surgimento de ferramentas *open-source* com o mesmo objetivo, e tão ou mais robustas que as comerciais.

1.1. OBJETIVO DO TRABALHO

Implementar uma classe para explorar os *frameworks* Java, *JasperReports* e *Hibernate*, de forma a acessar qualquer base de dados (que tenha o suporte do *Hibernate*), com o objetivo de preservar a integridade dos registros gravados de modo que, quando os relatórios forem solicitados pelo usuário alguns processos efetivamente preparados executem na base de dados a busca de todas as informações necessárias e de uma forma diferenciada, processá-las para que qualquer relatório acesse somente uma única tabela, que conterá todas as informações, que preencherá o relatório com o estilo solicitado pelo usuário.

1.2. PÚBLICO ALVO

Permitir aos programadores de *software* Java ou qualquer desenvolvedor de sistemas *desktop/web* um recurso diferenciado para o processamento de informações para o desenvolvimento dos relatórios. Os programadores podem

utilizar somente uma tabela para desenvolver visual de seus relatórios, de forma que, a manipulação das infinitas tabelas que poderão ser utilizadas na criação do relatório fique na responsabilidade dos analistas/programadores ou DBA's.

1.3. JUSTIFICATIVA

Disponibilizar aos desenvolvedores de software um recurso para facilitar o processo na etapa de criação do visual e estilo dos relatórios. Ajudar a representação das tomadas de decisões da empresa pelos relatórios. Minimizar as possibilidades de erro na apresentação dos dados no relatório.

1.4. MÉTODO DE DESENVOLVIMENTO

Serão realizadas algumas revisões bibliográficas, pesquisas e tutoriais necessários para o desenvolvimento desta. Serão utilizadas e avaliadas algumas ferramentas de criação dos relatórios (*iREPORT*), alguns *frameworks* Java para implementação das rotinas de chamada aos relatórios (*JASPER REPORTS*). O banco de dados *MYSQL* para demonstrar algumas características importantes para a construção de uma tabela genérica para a criação e implementação de relatórios.

1.4.1. **Ferramentas pesquisadas:**

- *JasperReports*: É um poderoso *framework*, escrito em Java, que é usado para geração de relatórios, permitindo gerar dinamicamente relatórios em vários formatos, como: PDF, HTML, e outros. Para facilitar a formatação e criação do visual dos relatórios será usada a ferramenta *iReport.*
- *iReport*: Uma ferramenta que permite definir o estilo do relatório através de um ambiente gráfico contendo todos os recursos que a biblioteca *Jasper* oferece. É possível definir relatórios com estilos modernos e complexos sem se quer escrever uma linha de código XML, pois este é gerado automaticamente. O ambiente ainda oferece atalhos para tarefas de compilação e visualização, permitindo a realização de testes, acelerando assim o processo de definição do estilo a ser usado.

- *NetBeans IDE*: É um ambiente de desenvolvimento - uma ferramenta para programadores escrever, compilar, depurar e implantar programas. É escrito em Java - mas pode suportar várias linguagens de programação. Existe também um enorme número de módulos para aprimorá-lo. O *NetBeans IDE* é um produto gratuito sem restrições de como ser utilizado. Esta ferramenta integra os dois *frameworks (JasperReports e iReports*) ao desenvolvimento, facilitando o uso das mesmas de forma integrada. Todas as compilações e validações de erros e consistências são oferecidas dentro do *NetBeans.*
- O *MySQL:* É o *software* de banco de dados de código aberto mais popular do mundo. Muitas organizações hoje em dia usam o *MySQL* para economizar tempo e dinheiro para ativar seus *web sites* de grande volume, sistemas críticos para os negócios, redes de comunicação e *software* comercial. O banco de dados está disponível para uma série de plataformas de *hardware* e *software*, incluindo *Red Hat Enterprise Linux, SuSE Enterprise Linux, Microsoft Windows*, Sistema Operacional *Sun Solaris 10, Mac OS X, Free BSD, HP-UX, IBM AIX, IBM i5/OS* e outras distribuições *Linux* populares.
- **1.4.2. Ferramenta para a análise:**

  JUDE: é uma *IDE* para Modelagem de Dados *(UML),* feita em Java e de uso fácil e intuitivo. Com a *IDE JUDE* é possível realizar uma modelagem de dados complexa, apresentar os dados para o usuário de forma clara e ainda com a vantagem de seu ambiente visual ser bem intuitivo. A *IDE* possui à sua esquerda uma árvore de estrutura com todos os dados à disposição do usuário para se criar diagramas, mapas etc.

1.5. ESTUDO DE CASO

O sistema de controle e gerenciamento de orçamentos desenvolvido pelo aluno Jabes Felipe Cunha, segue o paradigma de programação orientado a objetos. Com o propósito de desenvolver relatórios, este trabalho foca na construção dos mesmos, se integrando com o projeto do aluno exposto.

2. FRAMEWORK - JASPER REPORTS

Segundo Gonçalves (2008),

> Criado em 2001 por *Teodor Danciu*, quando este teve a tarefa de avaliar ferramentas de relatórios para um projeto que estava trabalhando. As soluções existentes eram muito caras para o orçamento do projeto, e em uma situação comum do mundo do desenvolvimento, ele decidiu escrever sua própria ferramenta de relatórios, que ao liberá-la para a comunidade, ficou imensamente popular em pouco tempo.

JasperReports é uma biblioteca escrita em Java, de código fonte *open source*, projetada para ajudar o desenvolvedor com a tarefa de criar relatórios para aplicações, tanto *Desktop* como *Web*, fornecendo uma API que facilita sua geração. Embora seja simples, ainda exige que o desenvolvedor conheça seu formato XML utilizado para criar os relatórios.

O *JasperReports* tem como objetivo tornar simples a forma de desenvolvimento do relatório, escrito totalmente em XML, sendo assim fica fácil sua compreensão e manutenção. No estilo do relatório, tem campos a serem preenchidos e seus respectivos tipos, como também a linguagem SQL embutida. Definidos em um arquivo XML através de *tags* que obedecem a uma estrutura e restrições declaradas em um arquivo DTD - *Document Type Definition* (*jasperreports.dtd*). Usando XML, o desenvolvedor pode definir textos estáticos, imagens, linhas, formas geométricas, como retângulos e elipses, e suas localizações dentro do relatório. Quando pronto esse arquivo XML precisa ser compilado, no qual é gerado um arquivo com a extensão.*jasper*, contendo a versão compilada do código XML.

2.1. FUNCIONAMENTO DO FRAMEWORK

A compilação do arquivo XML representando o estilo é feito pelo método *compileReport* encontrado na classe (*net.sf.jasperreports.engine.JasperCompileManager*). Nesta compilação, o estilo do

relatório é carregado em um objeto do projeto do relatório e então é colocado em série e armazenado no disco. Este objeto colocado em série é utilizado quando a aplicação quer preencher o projeto especificado do relatório com os dados. A compilação de um projeto do relatório implica na compilação de todas as expressões de Java definidas nas linhas de XML que representam o projeto do relatório. Várias verificações são feitas em tempo de compilação para checar a consistência do desenho do relatório. Como resultado você tem um relatório gerado que pode ser usado para mostrar resultados vindos de diferentes bancos de dados. O relatório pode ser enviado para impressora, tela ou ser transformado em documentos nos formatos PDF, HTML, XLS, CSV, RTF, TXT ou XML, o que o torna até mesmo independente do próprio *JasperReports* estar ou não instalado na máquina do usuário final.

2.2. ARQUITETURA DO FRAMEWORK

Para produzir um relatório precisamos fornecer dados ao *Jasper.* Esses dados podem ser consultas SQL inserida ao código XML ou geradas por uma classe Java por meio de um objeto *ResultSet,* que será passado às classes do *Jasper* para o preenchimento do relatório. O *JasperReports* suporta vários tipos de *datasources* (fonte de dados) através de uma interface específica chamada *JRDataSource.* Há uma implementação padrão desta interface para objetos *ResultSet*, chamada *JRResultSetDataSource.* Podemos dizer que um *datasource* somado a um arquivo.*jasper* gera um arquivo.*jrprint*, que pode ser exportado para os formatos PDF, HTML, XML, XLS, CVS, RTF ou TXT. A exportação será feita usando classes especiais que implementam exportadores específicos.

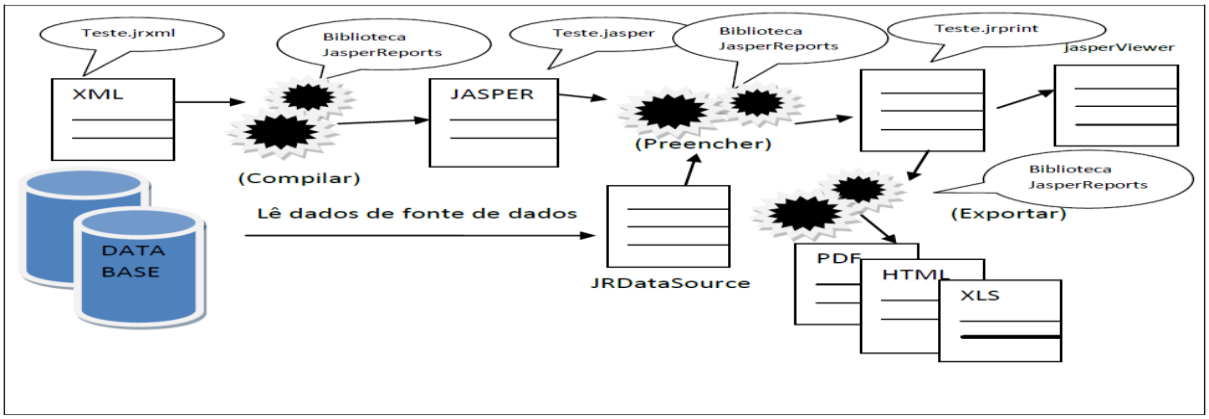


Figura 1. Ilustra o processo executado desde a compilação do jrxml até a exportação para diversos formatos.

O *JasperReports* divide o visual do relatório em áreas pré-definidas, chamadas seções (*Bands*).

As seções levam em consideração a estrutura visual de um relatório. São elas: *title*, *pageHeader*, *columnHeader*, *detail*, *columnFoter*, *page Footer*, *lastPageFooter e summary.* (Obs.: um relatório pode ter as seções acima descritas, mas não precisa necessariamente utilizar todas, podendo usar somente aquelas que forem necessárias.)
A seguir você tem a descrição de cada uma das seções existentes em um relatório *JasperReports:*

Title

É a primeira seção visível na construção de um relatório vazio. A seção de título. Esta seção só aparece uma vez no começo do relatório. Ex.: Relatório de Alunos.

```
<title>
  <band height="50">
     <staticText>
            <reportElement x="180" y="0" width="200"heigth=""20"/>
            <text>
              <![CDATA[Title]]>
            </text>
      </staticText>
  </band>
</title>
```

Figura 2. Ilustra o código XML da seção *title.*

Page Header

Esta seção aparece no começo de cada página impressa. A altura especificada durante a fase de desenho do relatório não se altera durante o processo de criação do mesmo (exceto que haja componentes redimensionáveis, como uma caixa de texto contendo um texto longo ou um sub-relatório). Neste local você pode ter data e hora e/ou nome ou informações da empresa. As seções *Title* e *Summary* não são inclusas no cabeçalho da página quando impressa em uma página separada.

```
<pageHeader>
    <band height="20">
        <staticText>
              <reportElement x="180" y="30" width="200"heigth=""20"/>
              <text>
                <![CDATA[Page Header]]>
              </text>
        </staticText>
    </band>
</pageHeader>
```

Figura 3. Ilustra o código XML da seção *pageHeader.*

Column Header

Esta seção só aparece no começo de cada interação com a seção *detail*. Se o relatório tem uma só coluna definida, formando um relatório tabular, então são ignoradas as seções cabeçalhos e rodapés de coluna. Como exemplo você pode listar nomes escolhidos de campos específicos como: Nome do Aluno, R.A, Data da Matrícula.

```
<columnHeader>
   <band height="20">
      <staticText>
            <reportElement x="180" y="50" width="200"heigth=""20"/>
            <text>
               <![CDATA[Page Header]]>
            </text>
      </staticText>
   </band>
</columnHeader>
```

Figura 4. Ilustra o código XML da seção *columnHeader*.

Detail

É o local de exibição dos dados de um objeto *datasource* ou *query*. Esta seção se repete enquanto houver linhas para serem colocadas no seu relatório de acordo com a fonte de dados transmitida.

```
<detail>
  <band height="20">
    <textField>
            <reportElement x="10" y="0" width="600"heigth=""20"/>
            <textFieldExpression class="java.lang.String">
            <![CDATA[$F{FieldName}]]>
          </textFieldExpression>
    </textField>
  </band>
</detail>
```

Figura 5. Ilustra o código XML da seção *detail.*

Column Footer

Esta seção aparece abaixo de cada coluna. Vale o mesmo comentário da seção *Column Header*.

```
<columnFooter>
   <band height="20">
      <staticText>
            <reportElement x="0" y="0" width="200"heigth=""20"/>
             <text>
                 <![CDATA[ColumnFooter]]>
             </text>
      </staticText>
   </band>
</detail>
```

Figura 6. Ilustra o código XML da seção *columnFooter*.

Page Footer

Representa o rodapé da página. Esta seção aparece no final de cada página. Esta seção pode conter informações como o número da página, o total de páginas encontradas, endereço da empresa e etc.

```
<pageFooter>
    <band height="20">
       <staticText>
            <reportElement x="0" y="5" width="200"heigth=""20"/>
          <text>
            <![CDATA[PageFooter]]>
          </text>
      </staticText>
    </band>
</pageFooter>
```

Figura 7. Ilustra o código XML da seção pageFooter.

Last Page Footer

A última seção do rodapé da página. Esta seção substitui o rodapé da página regular na última página do relatório. Se você quer que o último rodapé da página seja diferente dos demais, isto é possível usando esta seção especial. Se esta seção possuir o valor zero de altura, será completamente ignorada e o estilo desenhado para as páginas comuns será usado também para a última página.

```
<lastPageFooter>
  <band height="20">
     <staticText>
          <reportElement x="0" y="5" width="200"heigth=""20"/>
           <text>
             <![CDATA[LastPageFooter]]>
           </text>
      </staticText>
   </band>
</lastPageFooter>
```

Figura 8. Ilustra o código XML da seção *lastPageFooter*.

Summary

A seção de sumário, conhecido em outros sistemas como *Report footer* (Rodapé do relatório). Esta seção só aparece uma vez ao término do relatório e permite você adicionar um campo possuindo o total geral, por exemplo, de todo o relatório ou qualquer outra coisa que deseje no fim deste (como um gráfico).

```
<summary>
   <band height="20">
     <staticText>
           <reportElement x="0" y="5" width="200"heigth=""20"/>
             <text>
             <![CDATA[Summary]]>
            </text>
     </staticText>
   </band>
</summary>
```

Figura 9. Ilustra o código XML da seção *summary.*

3. IREPORT

Segundo Gonçalves (2008),

> Em 09 de outubro de 2002, *Giulio Toffoli* lançou, de forma independente, uma ferramenta para gerar relatórios visuais, chamando-a de *iReport,* atualmente este *framework* pode ser utilizado, totalmente integrado a *IDE NetBeans.* Essa integração não impede que o *iReport* seja usado com outra *IDE,* pois também poderá ser utilizado sem a necessidade do IDE para desenvolver os relatórios.

Desenvolver relatórios gerando o formato XML no padrão *JasperReports* definindo o visual do relatório através de um ambiente gráfico contendo todos os recursos que a biblioteca *Jasper* oferece. É integrado com *JFreeChart*, uma das mais difundidas bibliotecas de gráficos *open-source* para Java. É possível definir relatórios com visuais modernos e complexos sem se quer escrever uma linha de código XML, que é todo gerado automaticamente.

Com o objetivo de facilitar ao máximo a tarefa dos desenvolvedores de relatório o *iReport* engloba diversas características que o tornam uma ferramenta de desenvolvimento de relatórios profissional, no mesmo padrão de outros do mesmo tipo.

A seguir seguem algumas de suas principais características que fazem deste programa visual o oficial para desenvolvimento de relatórios *JasperReports:*

- Suporte a 100% das *tags XML* do *JasperReports;*
- Editor visual para criação de relatórios, possuindo ferramentas que incluem desenhos de retângulos, linhas, elipses, caixas de texto, rótulos, gráficos, sub-relatórios, códigos de barras e etc.;
- Um editor para escrever as expressões incluindo destaques (*highlights*) nas sintaxes;
- Suporte para Unicode e línguas não latinas como: Russo, Chinês, Japonês, Coreano e etc.;

- Suporte para todos os bancos de dados acessíveis pela ponte JDBC;
- Suporte virtual para todos os tipos de *DataSources;*
- Assistentes para criar relatórios rapidamente;
- Suporte para sub-relatórios;
- *Backup* para o código fonte;
- Suporte para modelos de documentos *(templates);*
- Suporte para fontes *TrueType;*
- Extensão através de *plug-ins;*
- Suporte a gráficos;
- Gerenciamento de bibliotecas de objetos padrão tais como numeração de páginas;
- Arrastar e soltar componentes do relatório*;*
- Ilimitados Desfazer e Refazer;
- Biblioteca de estilos e outros.

3.1. USO DO APLICATIVO

O ambiente oferece durante a criação atalhos para tarefas de compilação e visualização do relatório, permitindo a realização de testes. Os dados para imprimir podem ser recuperados de várias maneiras, incluindo múltiplas conexões JDBC, *TableModels, JavaBeans*, XML, etc. Resumindo com *iReport* é possível editar um arquivo .jrxml compilar e gerar o arquivo .jasper, além de poder executar e exportar para outros formatos dependendo do tipo de *datasource* usado

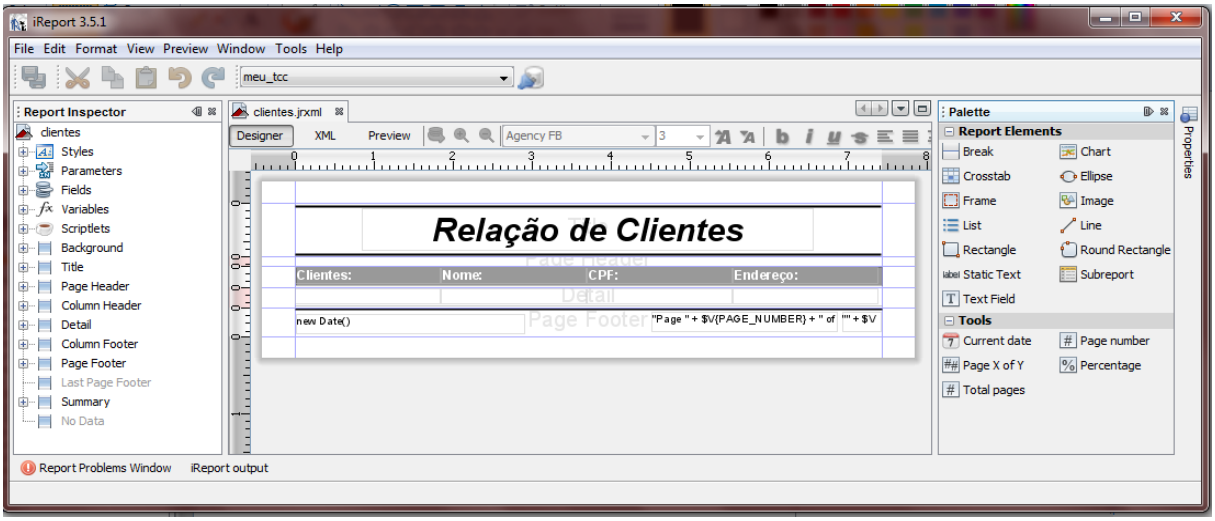


Figura 10. Ilustra o processo de construção de relatórios usando as facilidades do arrastar e soltar (*Drag'n Drop*).

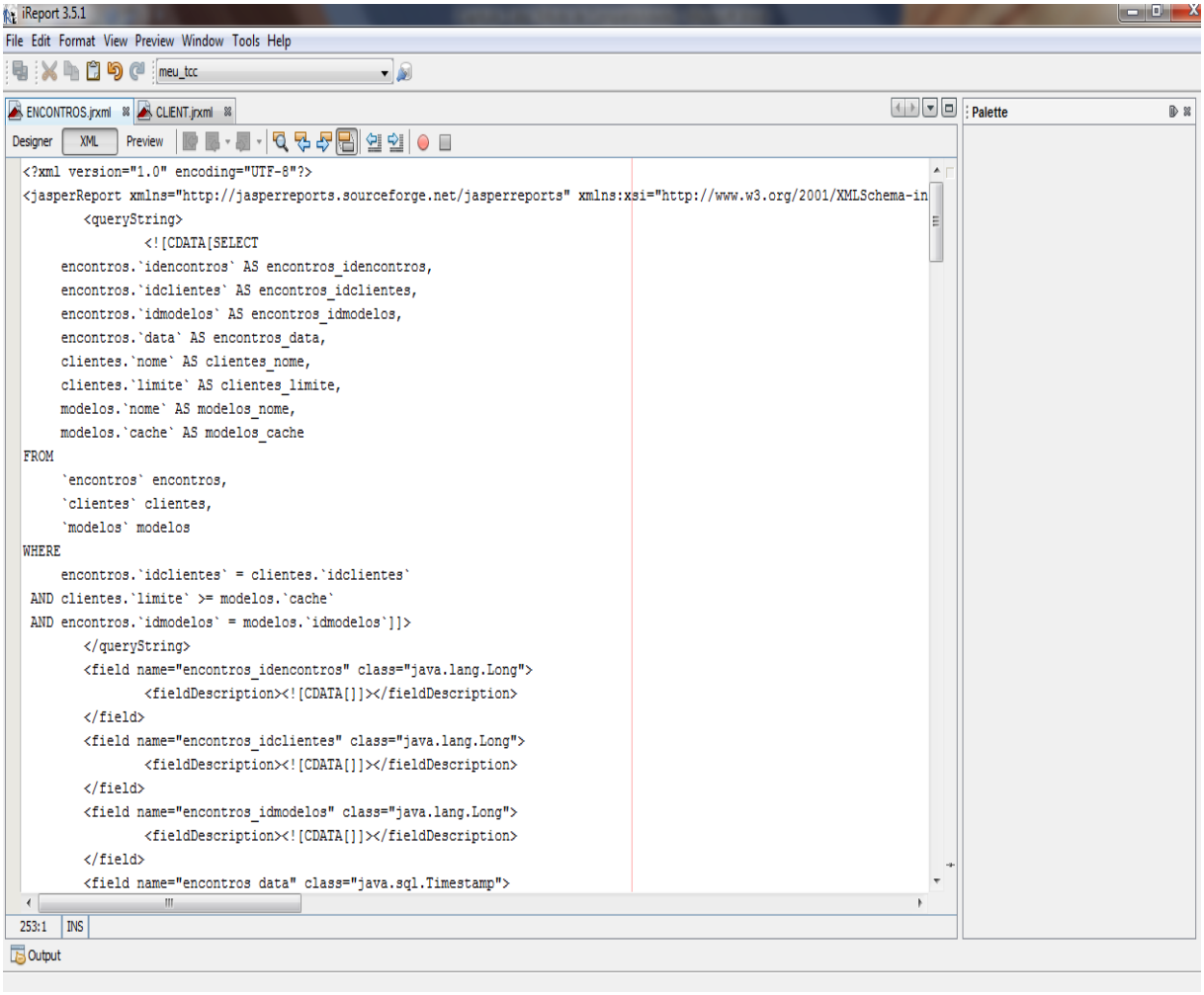


Figura 11. Ilustra a facilidade de editar o código XML do processo de construção de relatórios.

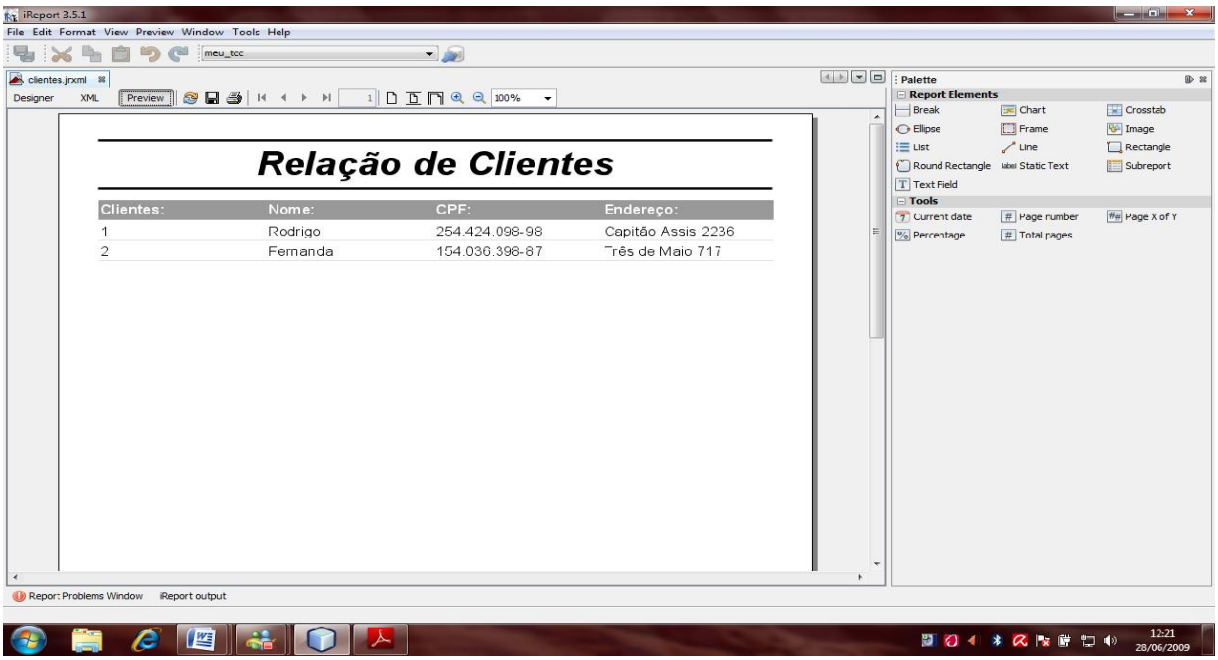


Figura 12. Ilustra a facilidade de visualização do relatório que está sendo construído.

4. MÓDULO PARA GERAR LISTAGENS E RELATÓRIOS

4.1. INTEGRAÇÃO COM O SISTEMA

Através da interface gráfica o usuário tem acesso a componentes, que disparam eventos a fim de buscar na base de dados informações para gerar relatórios, o Módulo contem os métodos responsáveis para gerenciar os processos que atendem a esses eventos, de forma a oferecer um recurso responsável pela geração de listagens e relatórios em sistemas *Desktop/Web.*

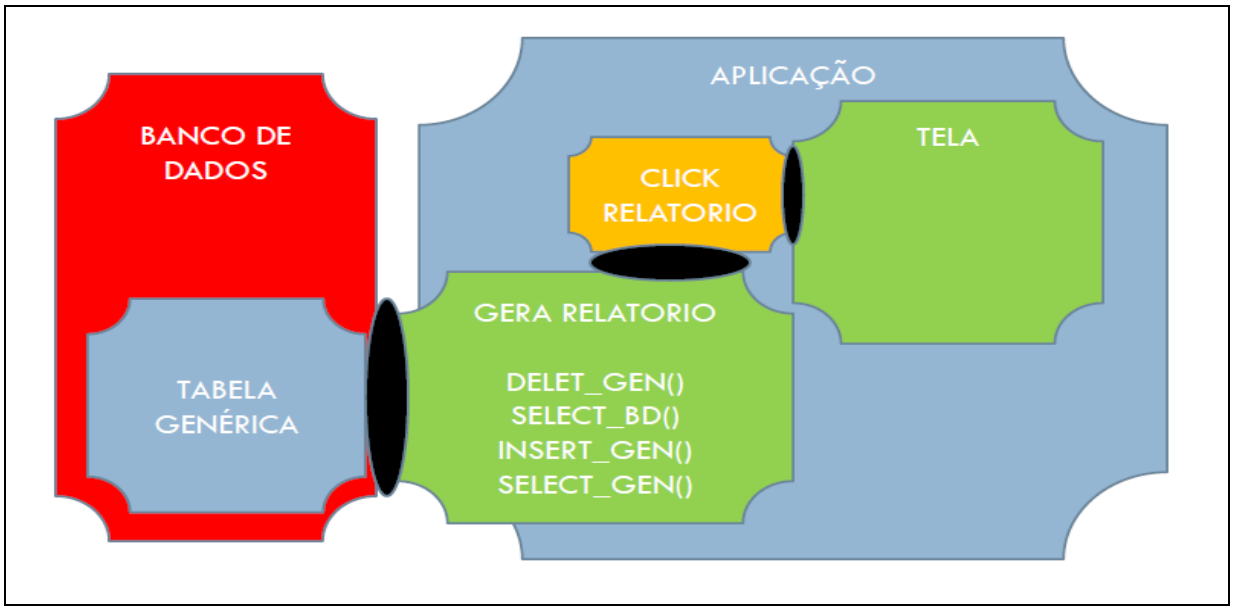


Figura 13. Ilustra a como o Módulo vai interagir com a aplicação e a base de dados para gerar os relatórios.

4.2. OBJETIVOS DO MÓDULO

O Módulo de gerar listagens e relatórios tem como objetivos facilitar o desenvolvimento de relatórios, na etapa de criação do visual e estilo, o fato de utilizar-se uma única tabela para processar as informações, elimina dessa fase de construção de relatórios todos os comandos SQL necessário para definir os relacionamentos entre tabelas, dessa forma a manipulação das infinitas tabelas que poderão ser utilizadas na criação do relatório fica na responsabilidade dos analistas,

assim há o favorecimento de manter a integridade dos registros gravados na base de dados.

4.3. ARQUITETURA DO MÓDULO

Integrado à aplicação, o Módulo é acionado através de um componente da tela que chama alguns métodos que vão habilitar o processo que acessa o banco de dados, e exclui os registros da tabela genérica. Concluída esta fase outros métodos habilitam um processo que seleciona informações nas tabelas do banco de dados. Vencida mais esta etapa, outros métodos preparados habilitam o processo que vai inserir as informações na tabela genérica na base de dados.Por fim outros métodos habilitam o processo que retornara à aplicação as informações solicitadas pelo usuário.

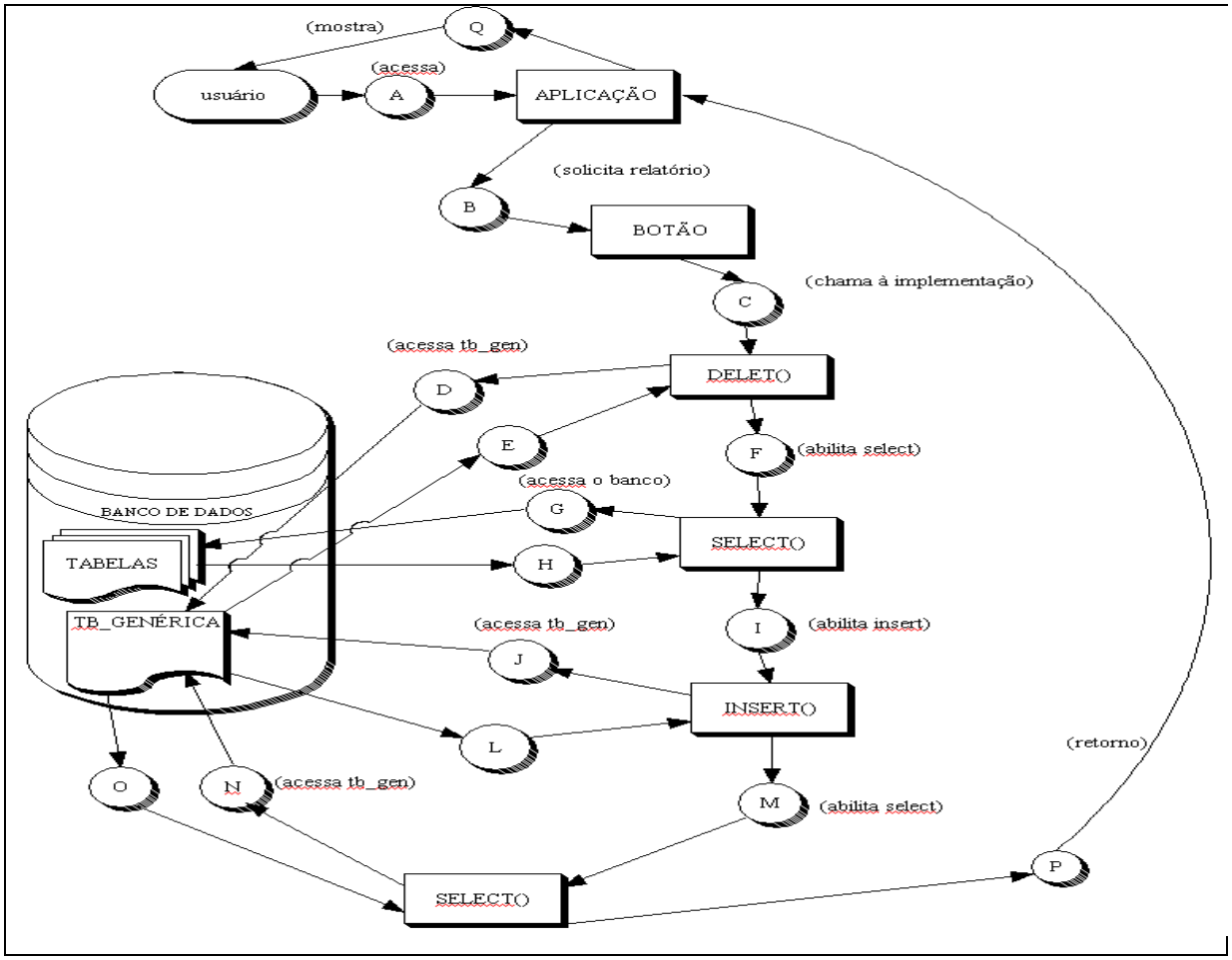


Figura 14. Ilustra a sequência de funcionamento do Módulo para alcançar o objetivo.

4.4. UML

4.4.1. Caso de uso

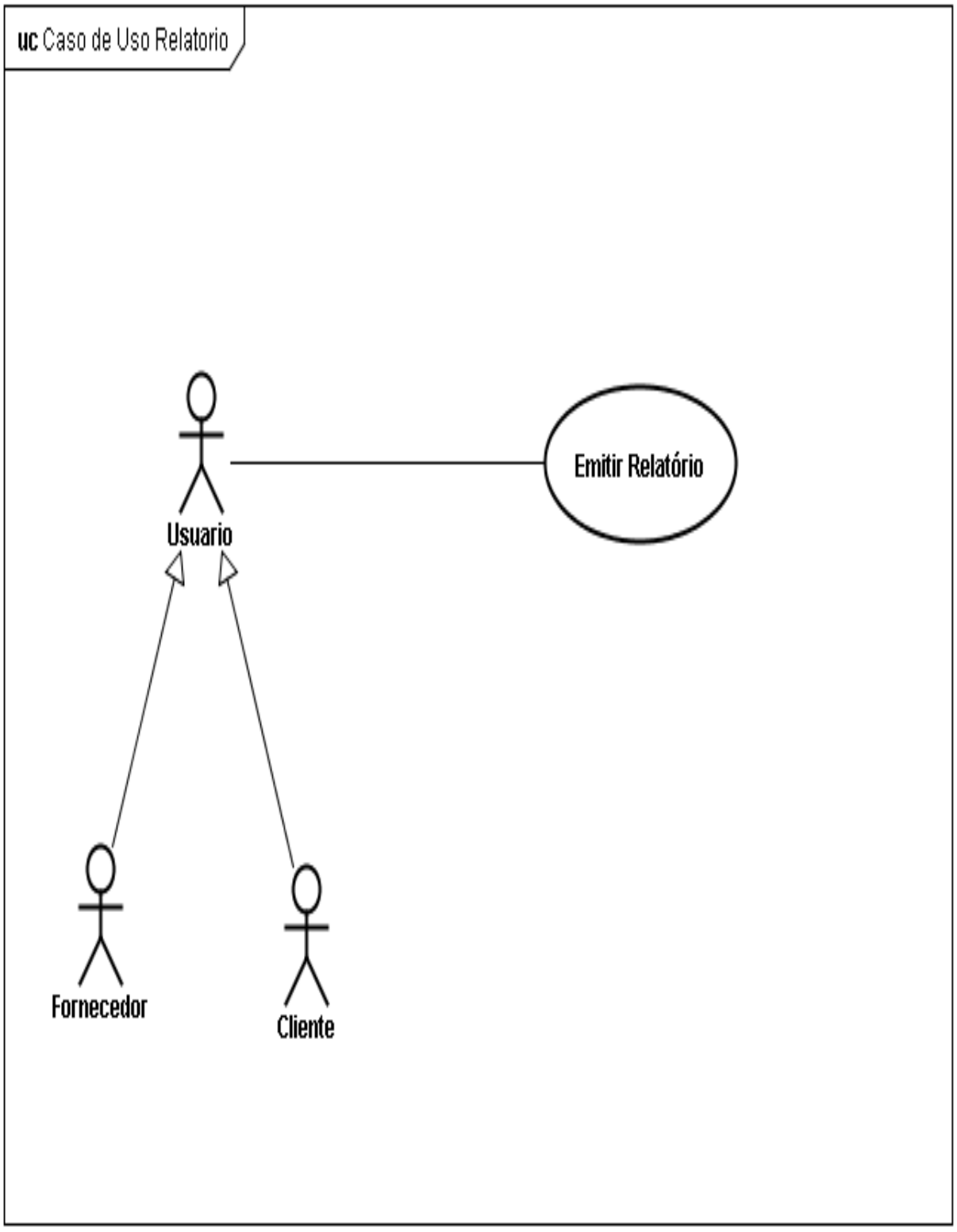


Figura 15. Ilustra o diagrama de caso de uso do Módulo de Gerar Relatórios.

4.4.2. Diagrama de Classes

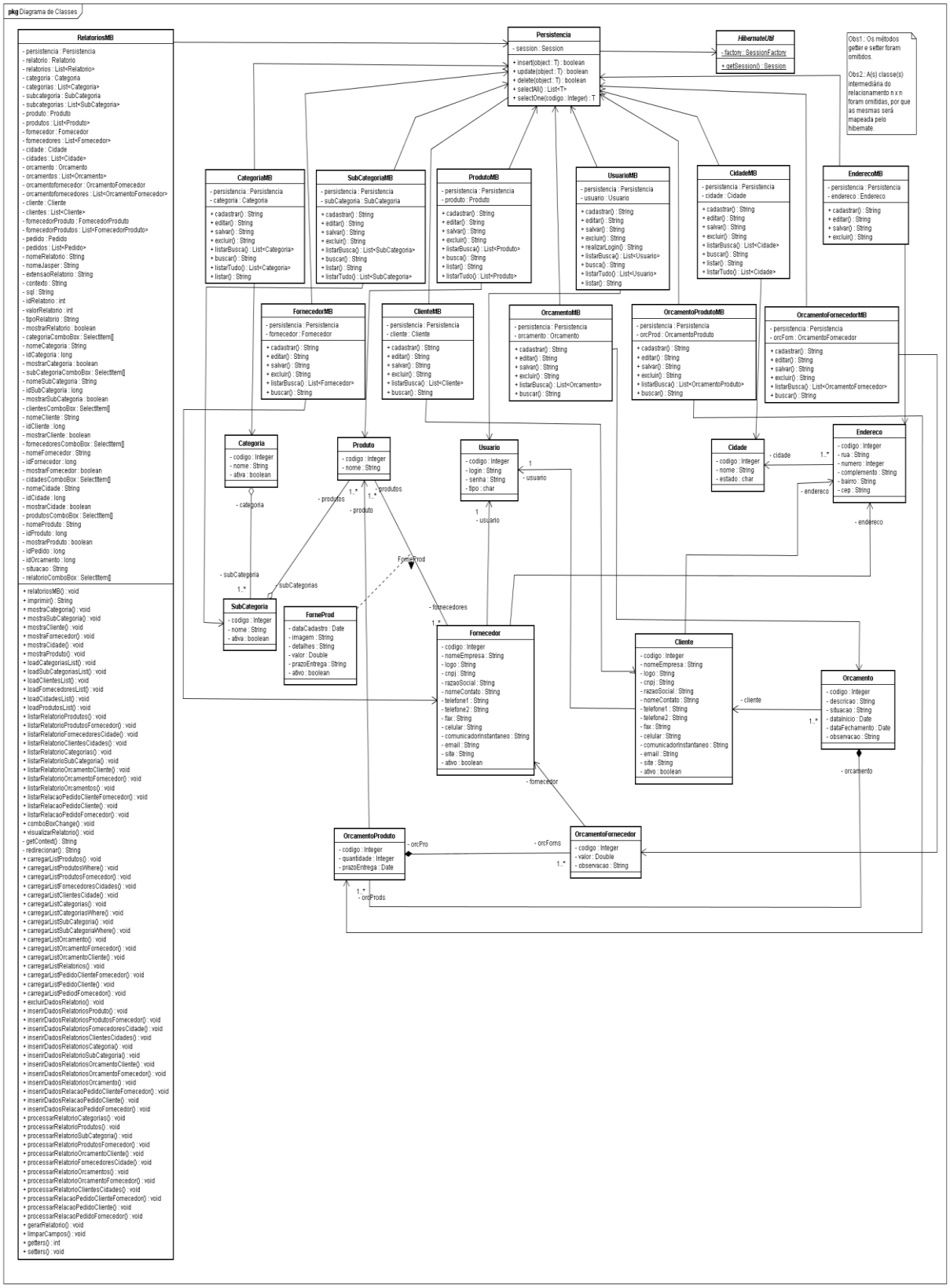


Figura 16. Ilustra o diagrama de classes do sistema onde foi usado o Módulo de Gerar Relatórios.

Figura 17. Ilustra de forma ampliada a definição dos objetos da classe principal do Módulo de Gerar Relatórios.

Figura 18. Ilustra de forma ampliada a definição dos métodos da classe principal do Módulo de Gerar Relatórios.

4.4.3. Diagramas de Sequência

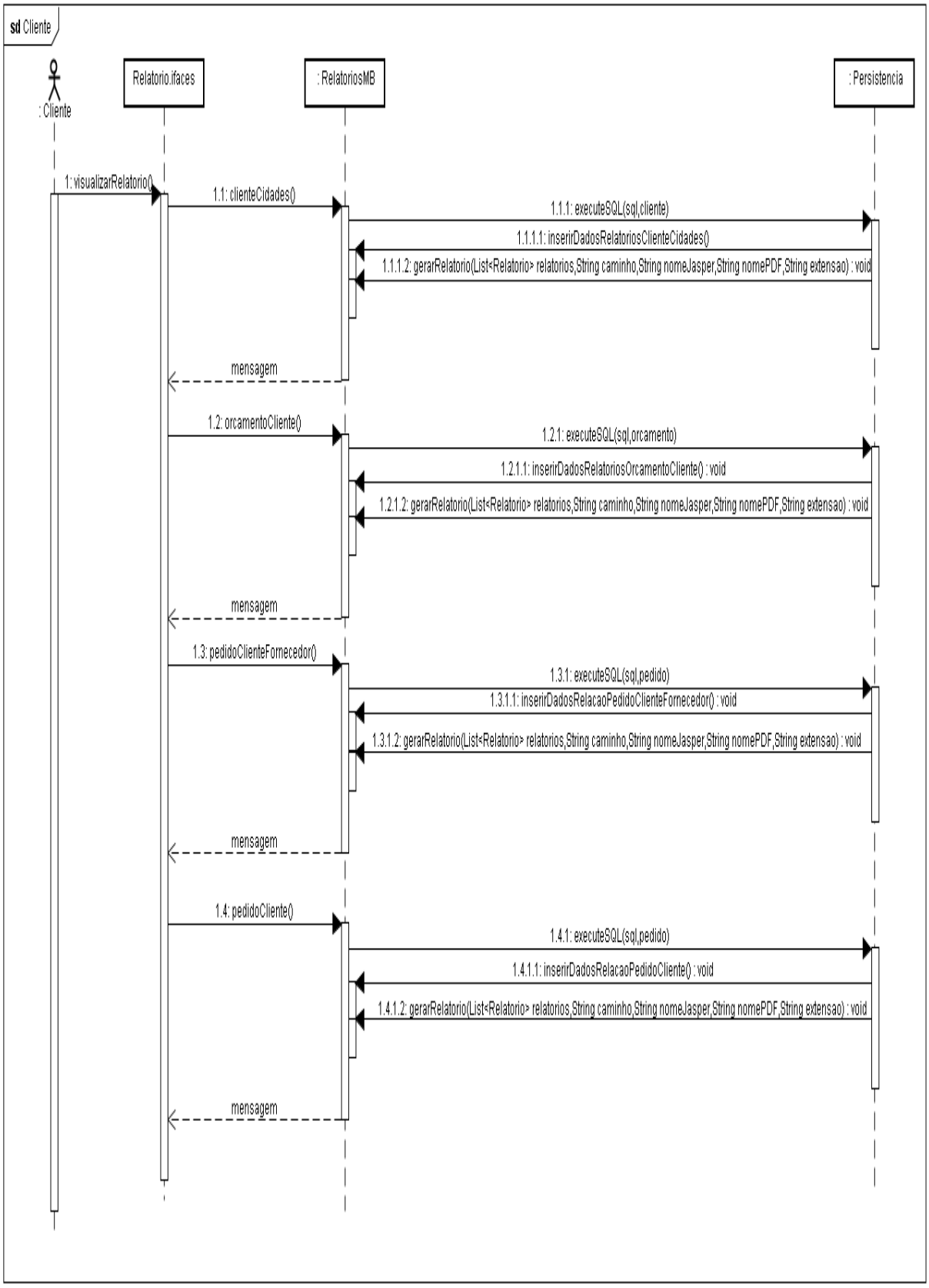


Figura 19. Ilustra o diagrama de sequência dos relatórios sobre clientes.

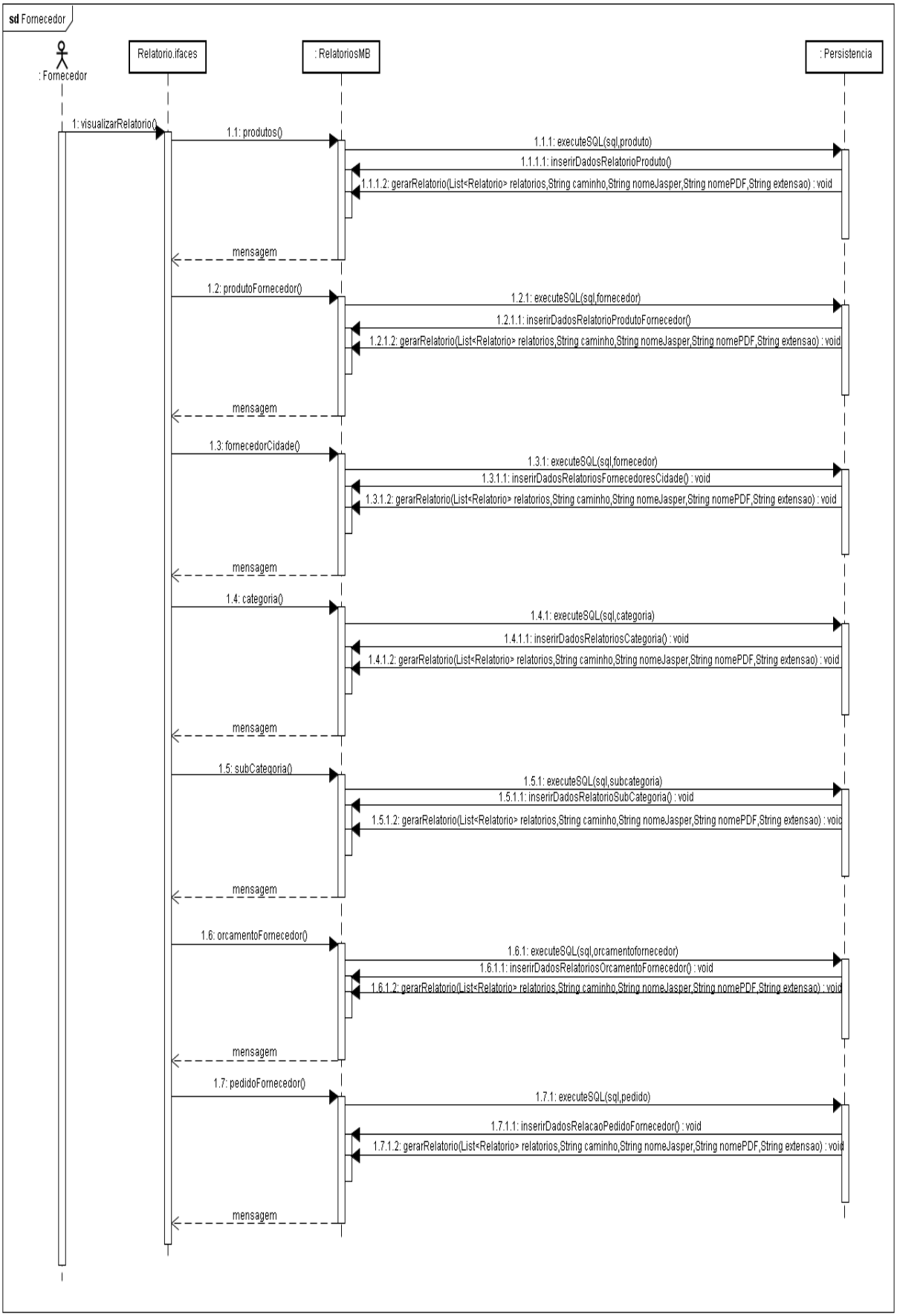


Figura 20. Ilustra o diagrama de sequência dos relatórios sobre fornecedores.

4.5. IMPLEMENTAÇÃO DOS RELATÓRIOS

Para a implementação das rotinas de geração de relatórios, existem alguns passos em comum. Na utilização do Módulo, é necessário preparar a tabela genérica para receber os dados do novo relatório. Essa preparação se resume nos seguintes passos:

1. Apagar os dados antigos da tabela genérica;
2. Realizar a busca dos dados nas demais tabelas do banco;
3. Inserir os dados da busca na tabela genérica;
4. Selecionar os novos dados da tabela genérica e enviar para o relatório.

4.5.1. Estrutura da Aplicação Desenvolvida

A aplicação seguiu o modelo MVC (Modelo-Visão-Controle), assim separando as classes por responsabilidade. Tal organização foi realizada através da criação de pastas:

- *Beans:* Nesta pasta residem às classes responsáveis por mapear os objetos da aplicação, um deles é a tabela genérica;
- Persistência: Esta pasta contém a classe que interage com o banco de dados;
- *ManagedBeans:* Esta contém a classe responsável pelo gerenciamento do visual e relatórios.

4.5.2. Execução da Aplicação Desenvolvida

Foi criada uma página JSP, que fez uso de componentes *ICE FACES* para gerenciar as ações do *view* com a classe *managedbeans*, essa página oferece todas as opções de chamadas a relatórios e seus possíveis filtros. Quando o usuário faz sua escolha através da caixa de opções e manda imprimir, é chamado um método que vai testar se a escolha contém um valor maior que zero, se for, é habilitado um componente na tela onde será feita a chamada aos métodos responsáveis pela execução dos processos que tornam possível o resultado proposto pelo Módulo.

Para melhor visualização de como é feito o filtro de escolha e a geração dos relatórios, temos o exemplo das rotinas que nos retorna o relatório de produtos por fornecedor.

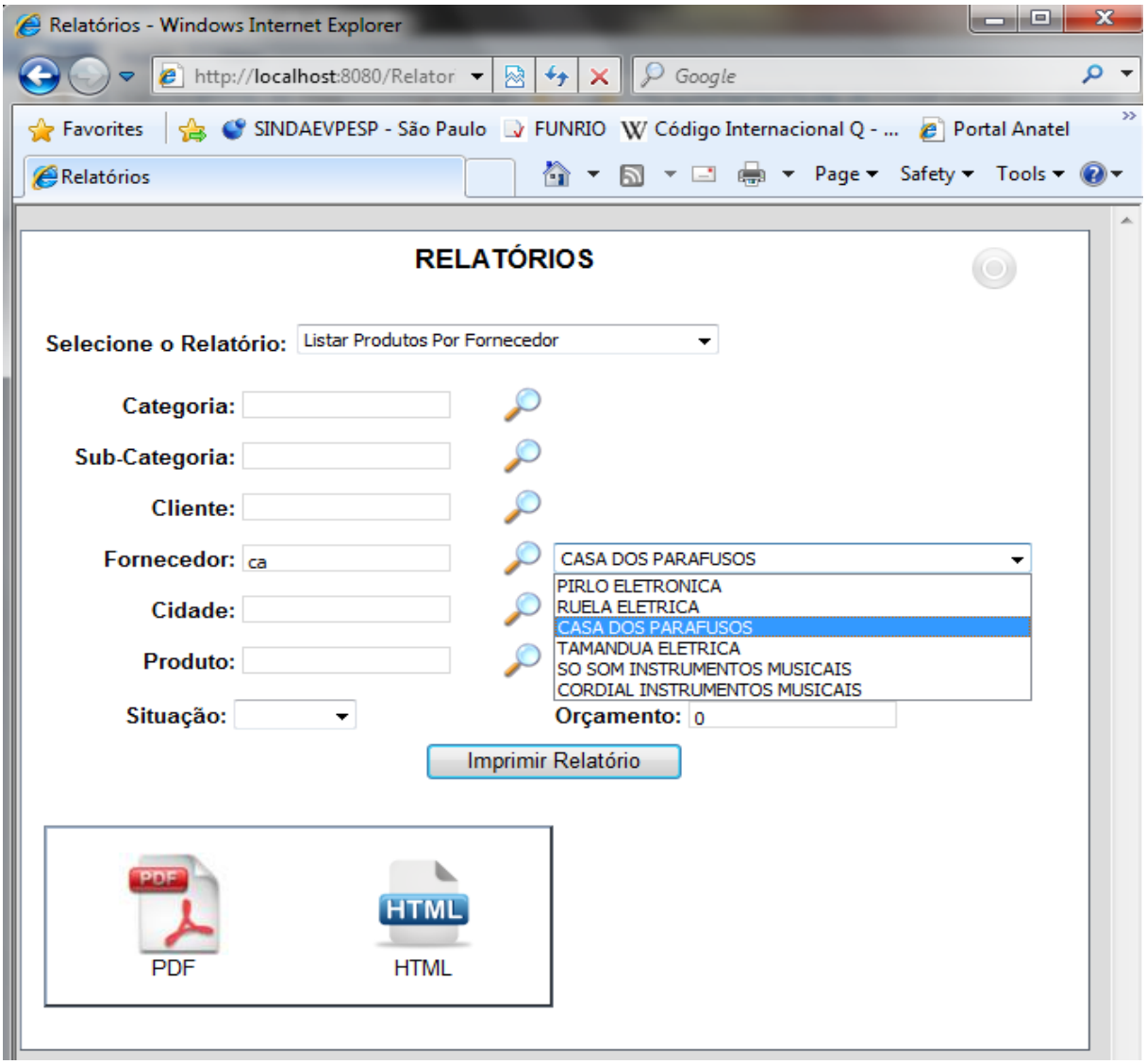


Figura 21. Ilustra as opções de filtro do relatório produtos por fornecedor.

A aplicação tem alguns métodos que testam se o usuário informou alguma opção de busca, confirmada essa opção, é habilitada uma rotina que chama um método que através da persistência genérica acessa a base de dados e carrega uma estrutura do tipo lista com os registros que contenham os caracteres informados. Carregada essa lista cria-se um objeto do tipo *SelectItem* do tamanho da lista retornada, e para cada posição desse *SelectItem* é criado um novo *SelectItem* para armazenar as informações da lista que serão extraídas através de uma estrutura de repetição do

tipo *for.* Assim é retornada para o usuário uma caixa que contém as opções de chamadas a relatórios específicos.

```
<td class="relatorio">
    <ice:outputText value="Fornecedor: " />
</td>
<td class="esquerda">
    <ice:inputText value="#{RelatorioMB.nomeFornecedor}" />
</td>
<td class="relatorio">
    <ice:commandLink action="#{RelatorioMB.mostraFornecedor}" >
        <ice:graphicImage value="./search.png" />
    </ice:commandLink>
</td>
<td class="combo">
    <ice:selectOneMenu id="fornecedor" value="#{RelatorioMB.idFornecedor}"
                       rendered="#{RelatorioMB.mostrarFornecedor}" style="width: 280px">
        <f:selectItems value="#{RelatorioMB.fornecedoresComboBox}" />
    </ice:selectOneMenu>
</td>
```

Figura 22. Ilustra trechos do código que chamam os métodos responsáveis pelo filtro do relatório produtos por fornecedor.

```
210    public SelectItem[] getFornecedoresComboBox() {
211        loadFornecedoresList();
212        fornecedoresComboBox = new SelectItem[fornecedores.size()];
213        for (int i = 0; i < fornecedores.size(); i++) {
214            fornecedoresComboBox[i] = new SelectItem(fornecedores.get(i).getCodigo(), fornecedores.get(i).getNomeEmpresa());
215        }
216        return fornecedoresComboBox;
217    }
```

Figura 23. Ilustra trechos do código que contém os métodos responsáveis pela opção de filtrar o relatório produtos por fornecedor.

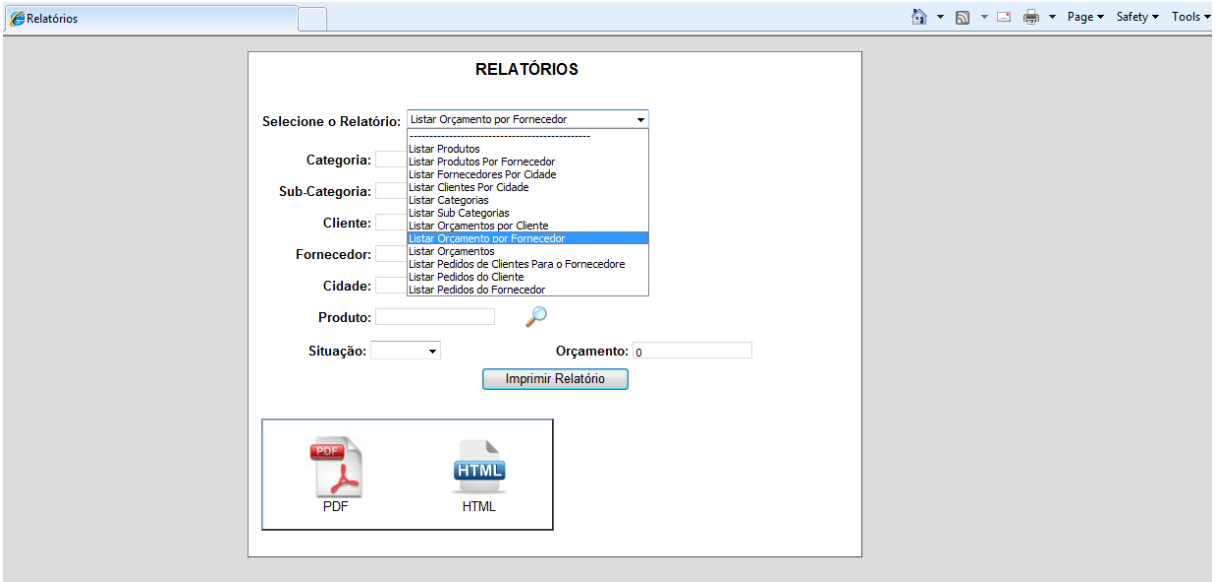


Figura 24. Ilustra a tela que interage com o usuário.

Uma vez feita à escolha do relatório, alguns métodos são requisitados através do botão imprimir relatório, o primeiro a ser executado é o método que testa o valor do relatório escolhido, se for válido, habilita-se uma rotina que contém as chamadas dos métodos necessários para geração dos relatórios correspondentes. Para um melhor entendimento da execução dos processos responsáveis pela geração dos relatórios temos a seguir os métodos e a sequência de execução.

Exemplo da geração do relatório de produtos por fornecedor:

Dentre as opções da rotina VisualizarRelatorio() foi escolhido o método ListarRelatorioProdutosFornecedor(), nesse método é atribuído a alguns objetos, valores (definidos pelo usuário) necessários para geração do relatório, e são feitas chamadas a outros métodos, como o processarRelatorioProdutoFornecedor(), que por sua vez faz chamada a três rotinas:

- excluirDadosRelatorio(), que chama um método que através da persistência genérica, carrega uma lista com os dados da tabela genérica, mediante a uma estrutura de repetição do tipo *for*, percorre-se essa lista e os valores iguais aos procurados são excluídos da tabela genérica. Após o fim da busca, um método limpa a lista deixando-a pronta para uma nova busca.
- inserirDadosRelatorio(), que chama um método que aceita instruções SQL que através da persistência genérica acessam as tabelas da base de dados e

carregam uma lista com as informações solicitadas, uma vez carregada à lista define-se uma estrutura de repetição que vai percorrer a lista e pra cada posição, será criado um objeto, que será carregado com as informações escolhidas para formar o relatório, cada objeto criado é inserido na tabela genérica e é adicionado em outra lista.

- gerarRelatorio(), esse método recebe entre seus parâmetros a lista com os objetos criados dentro da estrutura de repetição do método inserirDadosRelatório, dentro do gerarRelatorio são instanciadas algumas interfaces específicas do *Jasper,* como o *JRBeanCollectionDataSource,* que é uma coleção de fonte de dados que vai receber como parâmetro a lista de relatórios; uma *collection* simples do tipo *HashMap*, que recebe como parâmetros, a lista e a coleção de fonte de dados; um *JasperReport* que recebe o caminho e o nome do arquivo *.jasper* ; um *JasperPrint* que recebe como parâmetro o caminho, a *collection* simples e a *collectionDataSource.*O método contém um teste pra definir a geração do relatório conforme sua extensão.

Executadas essas rotinas é chamado um método que redireciona o contexto da aplicação para a geração do relatório, ou seja, traz o relatório na tela.

Por fim é chamado um método que limpa todos os campos da tela.

```
                                <td colspan="4" style="text-align: center; margin-top: 1em;">
                                    <ice:commandButton action="#{RelatorioMB.imprimir}" value="Imprimir Relatório" />
                                </td>
                            </tr>
                        </table>
                    </ice:panelGroup>

                    <ice:panelGroup rendered="#{RelatorioMB.mostrarRelatorio}" styleClass="crudFormRel">
                        <ice:panelGrid columns="2" styleClass="crudFormRelVis">
                            <ice:panelGroup styleClass="relItem">
                                <ice:commandLink action="#{RelatorioMB.visualizarRelatorio}">
                                    <ice:graphicImage value="./pdf.jpg" /><br />
                                    <ice:outputText value="PDF" />
                                    <f:setPropertyActionListener value="pdf" target="#{RelatorioMB.tipoRelatorio}" />
                                </ice:commandLink>
                            </ice:panelGroup>
                            <ice:panelGroup styleClass="relItem">
                                <ice:commandLink action="#{RelatorioMB.visualizarRelatorio}" >
                                    <ice:graphicImage value="./html.png" /><br />
                                    <ice:outputText value="HTML" />
                                    <f:setPropertyActionListener value="html" target="#{RelatorioMB.tipoRelatorio}" />
                                </ice:commandLink>
                            </ice:panelGroup>
                        </ice:panelGrid>
                    </ice:panelGroup>
                </ice:panelGroup>
```

Figura 25. Ilustra o trecho do código da página que habilitam os processos para gerar os relatórios.

```
129      public void visualizarRelatorio() {
130          switch (this.valorRelatorio) {
131              case 1: {
132                  listarRelatorioProdutos();
133                  break;
134              }
135
136              case 2: {
137                  listarRelatorioProdutosFornecedor();
138                  break;
139              }
140
141              case 3: {
142                  listarRelatorioFornecedoresCidade();
143                  break;
144              }
145
146              case 4: {
147                  listarRelatorioClientesCidades();
148                  break;
149              }
150
151              case 5: {
152                  listarRelatorioCategorias();
153                  break;
154              }
155
156              case 6: {
157                  listarRelatorioSubCategoria();
158                  break;
159              }
```

Figura 26. Ilustra o trecho do código que contém todas as chamadas aos métodos iniciais pra gerar os relatórios.

```
274     /**
275      * Método para listar o Relatório de Produtos por Fornecedor na View
276      */
277     public void listarRelatorioProdutosFornecedor() {
278         idRelatorio = this.valorRelatorio;
279         nomeJasper = "produtosFornecedor.jasper";
280         nomeRelatorio = "produtosFornecedor";
281         extensaoRelatorio = this.tipoRelatorio;
282         processarRelatorioProdutosFornecedor();
283         this.redirecionar(nomeRelatorio);
284         limparCampos();
285     }
```

Figura 27. Ilustra o trecho do código que contém o método pra listar o relatório de produtos por fornecedor.

```
971     public void processarRelatorioProdutosFornecedor() {
972         this.excluirDadosRelatorio();
973         this.inserirDadosRelatoriosProdutosFornecedor();
974         this.gerarRelatorio(relatorios, this.contexto, this.nomeJasper, this.nomeRelatorio, this.extensaoRelatorio);
975     }
```

Figura 28. Ilustra o trecho do código que contém o método pra processar o relatório de produtos por fornecedor.

```java
    /**
     * Método para fazer a exclusão dos Relatórios de acordo com seu ID do Relatório
     */
    public void excluirDadosRelatorio() {
        carregarListRelatorios();

        try {
            if (relatorios != null) {
                for (Relatorio rel : relatorios) {
                    if (rel.getId_relatorio().equals(this.idRelatorio)) {
                        persistencia.delete(rel);
                    }
                }
            }
        } catch (NullPointerException e) {
            e.printStackTrace();
        }
        relatorios.clear();
    }
```

Figura 29. Ilustra o trecho do código que contém o método pra excluir dados da tabela genérica.

```java
    /**
     * Método para fazer a insersão dos Relatórios de acordo com seu ID do Relatório
     */
    public void inserirDadosRelatoriosProdutosFornecedor() {
        carregaListProdutosFornecedor(idFornecedor);
        for (int i = 0; i < fornecedoreProdutos.size(); i++) {
            relatorio = new Relatorio();
            relatorio.setId_relatorio(2);
            relatorio.setCampo3(String.valueOf(fornecedoreProdutos.get(i).getFornecedorProdutoFK().getFornecedor().getNomeEmpresa()));
            relatorio.setCampo4(fornecedoreProdutos.get(i).getFornecedorProdutoFK().getProduto().getNome());
            relatorio.setCampo5(fornecedoreProdutos.get(i).getPrazoEntrega());
            relatorio.setValor2(fornecedoreProdutos.get(i).getValor());
            persistencia.insert(relatorio);
            relatorios.add(relatorio);
        }
    }

```

Figura 30. Ilustra o trecho do código que contém o método pra inserir dados no relatório.

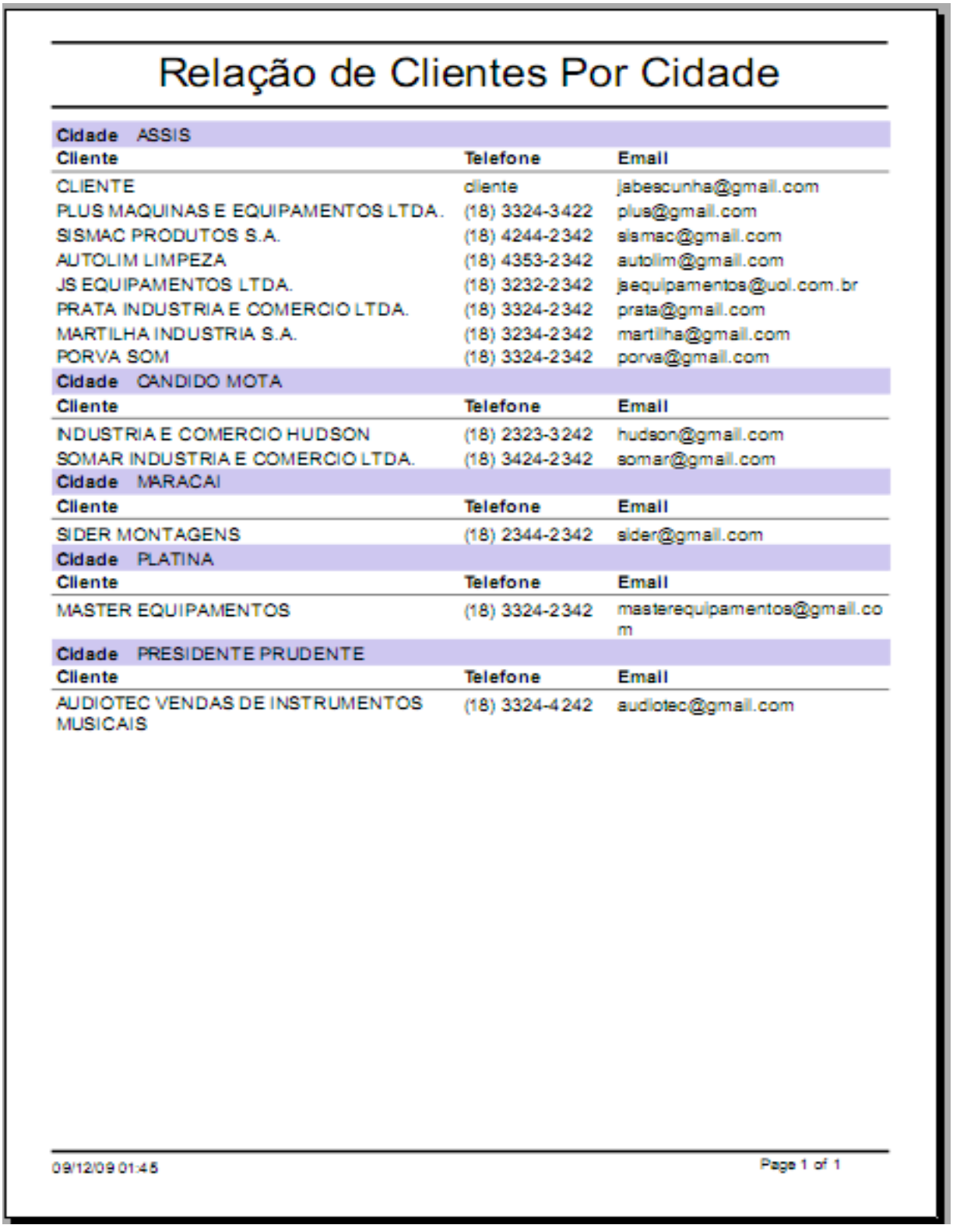
```
1034    public void gerarRelatorio(List<Relatorio> relatorios, String caminho,
1035            String nomeJasper, String nomeRel, String extensao) {
1036        try {
1037            JRBeanCollectionDataSource jrRS = new JRBeanCollectionDataSource(relatorios);
1038            Map parameters = new HashMap();
1039            parameters.put(relatorios, jrRS);
1040            JasperReport jr = (JasperReport) JRLoader.loadObject(caminho + "/" + nomeJasper);
1041            JasperPrint impressao = JasperFillManager.fillReport(jr, parameters, jrRS);
1042            if (extensao.equalsIgnoreCase("pdf")) {
1043                JasperExportManager.exportReportToPdfFile(impressao, caminho +
1044                        "/arquivos/" + nomeRel + "." + extensao);
1045            } else {
1046                JasperExportManager.exportReportToHtmlFile(impressao, caminho +
1047                        "/arquivos/" + nomeRel + "." + extensao);
1048            }
1049            System.out.println("Relatorio foi gerado!!!");
1050        } catch (JRException ex) {
1051            Logger.getLogger(RelatorioMB.class.getName()).log(Level.SEVERE, null, ex);
1052        }
1053    }
```

Figura 31. Ilustra o trecho do código que contém o método pra gerar os relatórios.

```
504     /**
505      * Método que redireciona o contexto para onde o Relatório foi gerado
506      * @param nomeArquivo String
507      */
508     private void redirecionar(String nomeArquivo) {
509         nomeArquivo = nomeArquivo + "." + extensaoRelatorio;
510         try {
511             FacesContext.getCurrentInstance().getExternalContext().
512                     redirect("../../RelatorioOrcamento/relatorios/arquivos/" + nomeArquivo);
513         } catch (IOException ex) {
514             Logger.getLogger(RelatorioMB.class.getName()).log(Level.SEVERE, null, ex);
515         }
516     }
```

Figura 32. Ilustra o trecho do código que contém o método que redireciona os relatórios para a tela.

A seguir alguns relatórios que foram gerados através do Módulo.



Relação de Clientes Por Cidade

Cidade ASSIS		
Cliente	**Telefone**	**Email**
CLIENTE	cliente	jabescunha@gmail.com
PLUS MAQUINAS E EQUIPAMENTOS LTDA.	(18) 3324-3422	plus@gmail.com
SISMAC PRODUTOS S.A.	(18) 4244-2342	sismac@gmail.com
AUTOLIM LIMPEZA	(18) 4353-2342	autolim@gmail.com
JS EQUIPAMENTOS LTDA.	(18) 3232-2342	jsequipamentos@uol.com.br
PRATA INDUSTRIA E COMERCIO LTDA.	(18) 3324-2342	prata@gmail.com
MARTILHA INDUSTRIA S.A.	(18) 3234-2342	martilha@gmail.com
PORVA SOM	(18) 3324-2342	porva@gmail.com
Cidade CANDIDO MOTA		
Cliente	**Telefone**	**Email**
INDUSTRIA E COMERCIO HUDSON	(18) 2323-3242	hudson@gmail.com
SOMAR INDUSTRIA E COMERCIO LTDA.	(18) 3424-2342	somar@gmail.com
Cidade MARACAI		
Cliente	**Telefone**	**Email**
SIDER MONTAGENS	(18) 2344-2342	sider@gmail.com
Cidade PLATINA		
Cliente	**Telefone**	**Email**
MASTER EQUIPAMENTOS	(18) 3324-2342	masterequipamentos@gmail.com
Cidade PRESIDENTE PRUDENTE		
Cliente	**Telefone**	**Email**
AUDIOTEC VENDAS DE INSTRUMENTOS MUSICAIS	(18) 3324-4242	audiotec@gmail.com

09/12/09 01:45 Page 1 of 1

Figura 33. Ilustra o relatório clientes por cidade.

Listagem de Pedidos Cliente à Fornecedor

Cliente: AUDIOTEC VENDAS DE INSTRUMENTOS MUSICAIS

Fornecedor:	Emissão:	Valor:	Prazo de Entrega:	Condição de Pagamento:
SO SOM INSTRUMENTOS MUSICAIS	16/10/09 09:22	4921.875	Imediato	A vista
CORDIAL INSTRUMENTOS MUSICAIS	16/10/09 09:23	5130.0	1 dia	7 dias
PLAYMUSIC INSTRUMENTOS LTDA.	16/10/09 09:24	4725.0	Imediato	A vista

Cliente: AUTOLIM LIMPEZA

Fornecedor:	Emissão:	Valor:	Prazo de Entrega:	Condição de Pagamento:
PIRLO ELETRONICA	18/09/09 09:55	23666.0	18 dias	30/60/90 DDL
TREVO MATERIAIS ELETRICOS	18/09/09 09:56	21100.0	19 dias	a prazo (30/60 dias)
TAMANDUA ELETRICA	20/09/09 00:51	659.0	10 dias	10 DDL

Cliente: INDUSTRIA E COMERCIO HUDSON

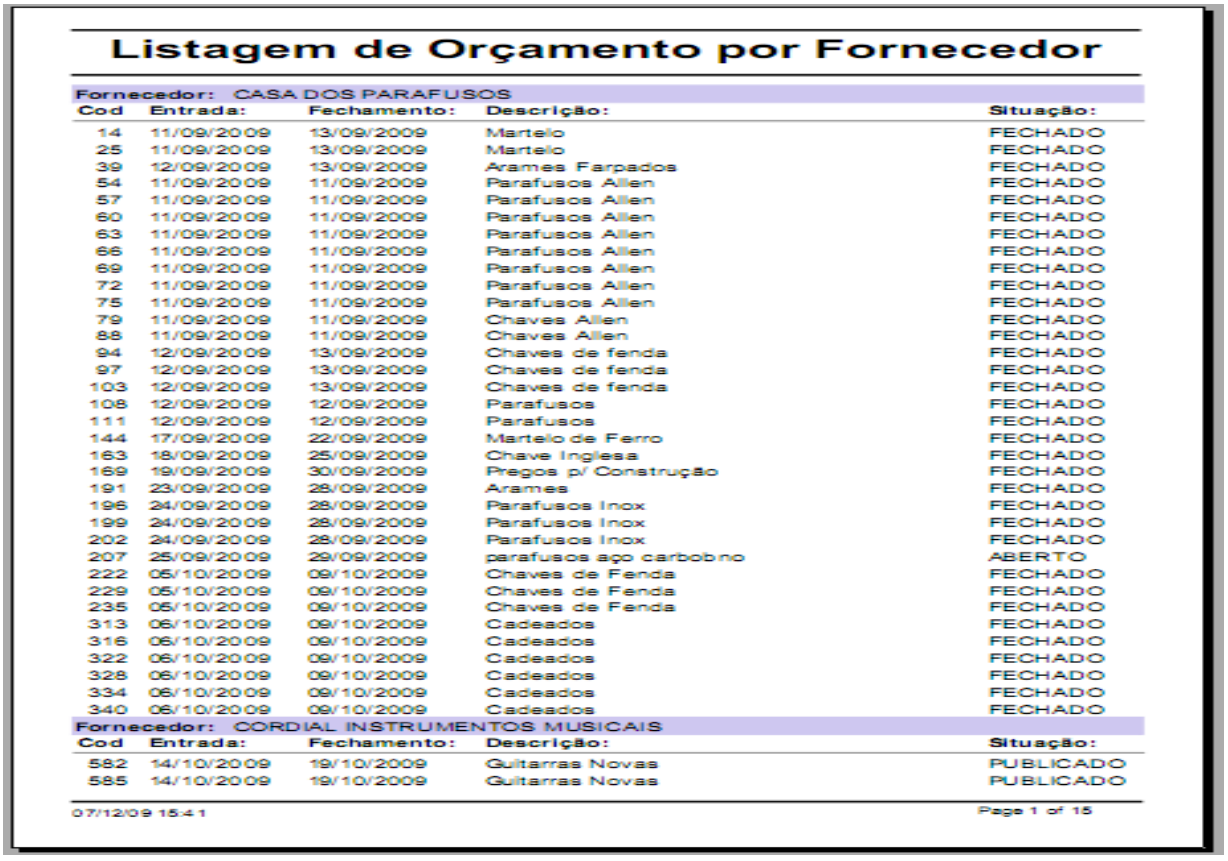
Fornecedor:	Emissão:	Valor:	Prazo de Entrega:	Condição de Pagamento:
PIRLO ELETRONICA	18/09/09 08:10	18106.0	18 dias	30 dias
SPI FERRAMENTAS	18/09/09 08:12	60.0	18 dias	cheque
TREVO MATERIAIS ELETRICOS	18/09/09 08:13	1699.0	11 dias	a vista
TAMANDUA ELETRICA	18/09/09 08:14	1508.5	8 dias	a prazo
PIRLO ELETRONICA	19/09/09 22:25	1249.0	11 dias	15 DDL
SPI FERRAMENTAS	30/09/09 08:13	60.0	a vista	10 dias

Cliente: JS EQUIPAMENTOS LTDA.

Fornecedor:	Emissão:	Valor:	Prazo de Entrega:	Condição de Pagamento:
MAX FERRAMENTAS	05/10/09 11:46	1295.811	1 dia útil	10/15/20 DDL
MITO FERRAGENS	05/10/09 11:47	1275.0	Imediato	30 dias

07/12/09 15:24 Page 1 of 3

Figura 34. Ilustra a listagem de pedidos de clientes a fornecedor.



Listagem de Orçamento por Fornecedor

Fornecedor: CASA DOS PARAFUSOS

Cod	Entrada:	Fechamento:	Descrição:	Situação:
14	11/09/2009	13/09/2009	Martelo	FECHADO
25	11/09/2009	13/09/2009	Martelo	FECHADO
39	12/09/2009	13/09/2009	Arames Farpados	FECHADO
54	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
57	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
60	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
63	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
66	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
69	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
72	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
75	11/09/2009	11/09/2009	Parafusos Allen	FECHADO
79	11/09/2009	11/09/2009	Chaves Allen	FECHADO
88	11/09/2009	11/09/2009	Chaves Allen	FECHADO
94	12/09/2009	13/09/2009	Chaves de fenda	FECHADO
97	12/09/2009	13/09/2009	Chaves de fenda	FECHADO
103	12/09/2009	13/09/2009	Chaves de fenda	FECHADO
108	12/09/2009	12/09/2009	Parafusos	FECHADO
111	12/09/2009	12/09/2009	Parafusos	FECHADO
144	17/09/2009	22/09/2009	Martelo de Ferro	FECHADO
163	18/09/2009	25/09/2009	Chave Inglesa	FECHADO
169	19/09/2009	30/09/2009	Pregos p/ Construção	FECHADO
191	23/09/2009	28/09/2009	Arames	FECHADO
196	24/09/2009	28/09/2009	Parafusos Inox	FECHADO
199	24/09/2009	28/09/2009	Parafusos Inox	FECHADO
202	24/09/2009	28/09/2009	Parafusos Inox	FECHADO
207	25/09/2009	29/09/2009	parafusos aço carbobno	ABERTO
222	05/10/2009	09/10/2009	Chaves de Fenda	FECHADO
229	05/10/2009	09/10/2009	Chaves de Fenda	FECHADO
235	05/10/2009	09/10/2009	Chaves de Fenda	FECHADO
313	06/10/2009	09/10/2009	Cadeados	FECHADO
316	06/10/2009	09/10/2009	Cadeados	FECHADO
322	06/10/2009	09/10/2009	Cadeados	FECHADO
328	06/10/2009	09/10/2009	Cadeados	FECHADO
334	06/10/2009	09/10/2009	Cadeados	FECHADO
340	06/10/2009	09/10/2009	Cadeados	FECHADO

Fornecedor: CORDIAL INSTRUMENTOS MUSICAIS

Cod	Entrada:	Fechamento:	Descrição:	Situação:
582	14/10/2009	19/10/2009	Guitarras Novas	PUBLICADO
585	14/10/2009	19/10/2009	Guitarras Novas	PUBLICADO

07/12/09 15:41 Page 1 of 15

Figura 35. Ilustra a listagem de orçamento por fornecedor.

Listagem de Orçamentos

Situação: ABERTO

Código:	Nome do Contato:	Descrição:	Início:	Fechamento:	Observação:
1	Brunna Luzia da Matta	Cabos p/ Instalação Elétrica	11/09/09 00:00	17/09/09 00:00	

Situação: FECHADO

Código:	Nome do Contato:	Descrição:	Início:	Fechamento:	Observação:
2	Brunna Luzia da Matta	Martelo	11/09/09 00:00	13/09/09 00:00	
3	Jéssica Parke	Temporizadores	11/09/09 00:00	11/09/09 00:00	
4	Jéssica Parke	Arames Farpados	12/09/09 00:00	13/09/09 00:00	
5	Flávia Neopina da Silvia de Souza	Luminárias Decorativas	11/09/09 00:00	12/09/09 00:00	
6	Marta Martina	Parafusos Allen	11/09/09 00:00	11/09/09 00:00	
7	Flávia Neopina da Silvia de Souza	Chaves Allen	11/09/09 00:00	11/09/09 00:00	
8	Dercy Alves de Miranda	Chaves de fenda	12/09/09 00:00	13/09/09 00:00	
9	Dercy Alves de Miranda	Parafusos	12/09/09 00:00	12/09/09 00:00	czdasd
10	Jéssica Parke	sensores	12/09/09 00:00	15/09/09 00:00	sdffds
11	Marta Martina	Aquecedores	12/09/09 00:00	12/09/09 00:00	
12	Flávia Neopina da Silvia de Souza	Martelo de Ferro	17/09/09 00:00	22/09/09 00:00	
13	Flávia Neopina da Silvia de Souza	Motores Elétrico	18/09/09 00:00	18/09/09 00:00	
14	Brunna Luzia da Matta	Chave Inglesa	18/09/09 00:00	25/09/09 00:00	Necessito urgente deste orçamento, o mais rapido possivel.
15	Brunna Luzia da Matta	Pregos p/ Construção	19/09/09 00:00	30/09/09 00:00	
16	Dercy Alves de Miranda	Timer	21/09/09 00:00	30/09/09 00:00	
17	Jéssica Parke	Sensores Digitais	22/09/09 00:00	28/09/09 00:00	
18	Dercy Alves de Miranda	Arames	23/09/09 00:00	28/09/09 00:00	
19	Marta Martina	Parafusos Inox	24/09/09 00:00	28/09/09 00:00	dsfsdfsdfsdfsd

07/12/09 22:33 Page 1 of 4

Figura 36. Ilustra a listagem de orçamentos.

5. CONCLUSÕES

5.1. RESULTADOS ALCANÇADOS

Como resultado dos estudos voltados à implementação do Módulo de Gerar Relatórios, pode-se dizer que mais uma facilidade pode ser oferecida ao programador incumbido do processo de criação de relatórios. Sabe-se que a biblioteca *Jasper* possui inúmeros recursos pra facilitar o processo de criação de relatórios, porem exige que o desenvolvedor tenha um bom conhecimento da linguagem XML, através do aplicativo *iReport* esse conhecimento em XML torna-se facultativo, pois ele oferece um ambiente visual para criar os relatórios tornando o código XML transparente para o desenvolvedor. Dentre as funcionalidades do *iReport* pode-se destacar a *Design query,* que nos oferece acesso as tabelas do banco de dados para selecionarmos as que serão usadas; uma vez selecionadas, inserimos a codificação de SQL para realizar os relacionamentos entre elas, para se processar os dados em busca das informações necessárias para gerar os relatórios. Através do Módulo de Gerar Relatórios, o desenvolvedor terá a opção de trabalhar com a biblioteca *Jasper* e o Aplicativo *iReport* com mais facilidade, pois atendendo o que foi proposto o Módulo de Gerar Relatórios torna capaz o uso de uma única tabela para processar e gerar qualquer relatório independente da quantidade de tabelas a ser acessadas para extrair as informações requisitadas, com isso o programador fará uso do *iReport* para desenvolver o estilo dos relatórios em cima de uma só tabela, sendo assim, não terá mais a preocupação com os relacionamentos entre elas e nem a função de escrever os comandos em SQL, porque Módulo de Gerar Relatórios trouxe para dentro da aplicação toda parte de codificação. Desde já, merece destaque o fato do desenvolvedor ter acesso somente a essa tabela, que é utilizada para processar os dados de forma genérica, dessa forma tem se como resultado a garantia da integridade dos registros gravados na base de dados.

5.2. TRABALHOS FUTUROS

Como trabalho futuro pretende-se explorar mais os recursos da persistência genérica a fim de aperfeiçoar alguns de seus métodos para que não seja mais

necessário o uso de instruções SQL dentro do Módulo de Gerar Relatórios como as que têm nos métodos que carregam os dados.

REFERÊNCIAS BIBLIOGRÁFICAS

LARMAN, Craig. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientado a objetos /** Craig Larman; trad. Luiz A. Meirelles Salgado. – Porto Alegre: Bookman, 2000.

MELLO, Ana Cristina. **Exercitando Modelagem em UML /** Ana Cristina Melo. – Rio de Janeiro: Brasport, 2006.

GRAVES, Mark. **Projeto de Banco de Dados com XML.** Tradução de Aldair José Coelho da Silva. São Paulo: Person Education do Brasil, 2003.

GONÇALVES, Edson. **Dominando Relatórios JasperReport com iReport**. 1ª edição-Rio de Janeiro: Editora Ciência Moderna, 2008.

SILVA, Simone Cardoso. **Vantagens e Desvantagens de se usar o Framework Hibernate.** 2009. Ciência da Computação – Fundação Educacional do Município de Assis, Assis, São Paulo, 2009.