# GERADOR DE CÓDIGO JSP BASEADO EM PROJETO DE SGBD

Acadêmico: Maicon Klug
Orientadora: Joyce Martins


www.furb.br

# Roteiro

- Introdução
- Objetivos do trabalho
- Fundamentação teórica
- Desenvolvimento do trabalho
- Conclusão
- Extensões

# Introdução

- Processo de desenvolvimento de software demanda tempo e esforço
- Trabalho repetitivo e dispendioso na implementação de interfaces e acesso ao banco de dados
- Possível solução: uso de geradores de código
- Ferramenta para a geração de páginas JSP a partir da definição de uma base de dados em um banco de dados relacional

# Objetivos do trabalho

- Desenvolver uma ferramenta para automatizar o processo de desenvolvimento de software, gerando páginas JSP a partir do dicionário de dados de um SGBD

- Efetuar a leitura do dicionário de dados dos SGBDs Oracle, Microsoft SQL Server e MySQL, através da API JDBC

- Utilizar *templates* na definição do código de saída

- Gerar páginas JSP com funcionalidades de inclusão, alteração, exclusão e consulta

- Gerar código no padrão MVC

# Fundamentação teórica

- Conceitos
  - ✓ Geradores de código
  - ✓ API JDBC
  - ✓ *Templates*
  - ✓ Motor de *templates* Velocity
  - ✓ Arquitetura MVC
  - ✓ JSP

- Trabalhos correlatos
  - ✓ CodGer
  - ✓ EasyCase
  - ✓ TechCodeGenerator
  - ✓ ClassGenerator

# Geradores de código

*Geração de código é a técnica pela qual se constrói código utilizando programas* (Herrington, 2003).

- Auxiliam no processo de desenvolvimento de software
- Podem gerar um sistema completo ou somente rotinas específicas
- Etapas para o desenvolvimento de um gerador de código
  - ✓ Identificação da saída
  - ✓ Definição da entrada e sua análise
  - ✓ Interpretação da entrada e formatação da saída
  - ✓ Geração da saída a partir das informações de entrada

# API JDBC

- Facilita e padroniza a comunicação de aplicações Java com o banco de dados

- Possibilita a manipulação das informações no banco de dados

- *Drivers* JDBC implementam as interfaces da API JDBC

- A API JDBC suporta quatro tipos de *drivers*:
  - ✓ *driver* JDBC-ODBC (*driver* tipo 1)
  - ✓ *driver* de acesso nativo (*driver* tipo 2)
  - ✓ *driver* de acesso por *middleware* (*driver* tipo 3)
  - ✓ *driver* de acesso direto ao servidor (*driver* tipo 4)

# *Templates*

- São arquivos utilizados para a geração de outros arquivos
- Utilizados em geração de código para permitir a padronização do código gerado
- Permitem que a formatação do código gerado esteja externa ao código da aplicação que o gera
- São formados por códigos estáticos e códigos dinâmicos

# Motor de *Templates* Velocity

- Não é uma aplicação, mas um conjunto de classes Java

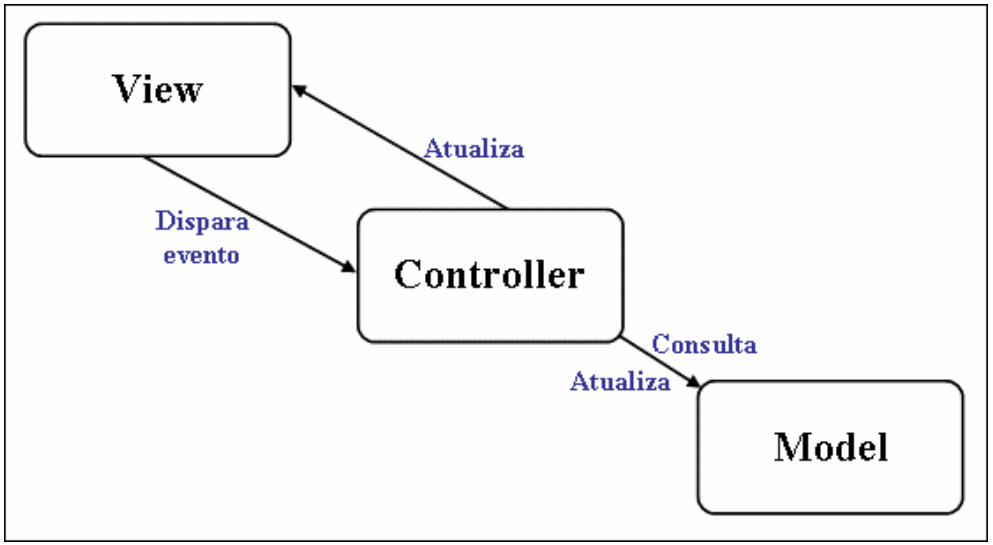
- Através da VTL, permite a inserção de informações de maneira dinâmica dentro do *template*

- A VTL possui recursos que permitem:
  - ✓ Referenciar variáveis dentro do *template*
  - ✓ Utilizar controles de fluxo de execução
  - ✓ Definir e invocar macros
  - ✓ Efetuar a chamada de métodos de objetos Java

# Arquitetura MVC

↳ Separa os dados, o fluxo da aplicação e a interface com usuário em camadas distintas

↳ Possibilita a substituição inteira de uma camada sem que as demais sejam afetas




Fonte: adaptado de Sun Microsystems (2002).

# JSP

- Tecnologia para o desenvolvimento de aplicações web
- Permite criar conteúdo dinâmico reutilizando componentes predefinidos
- Multiplataforma

# Desenvolvimento do trabalho

- Especificação dos requisitos funcionais e não funcionais
- Definição dos bancos de dados
- Estudo da API JDBC
- Interpretação das informações de entrada
- Especificação da saída
- Definição do Tomcat como servidor de aplicações
- Especificação da ferramenta através da UML
- Implementação
- Elaboração dos *templates*
- Definição de uma base de dados para testes

# Requisitos funcionais

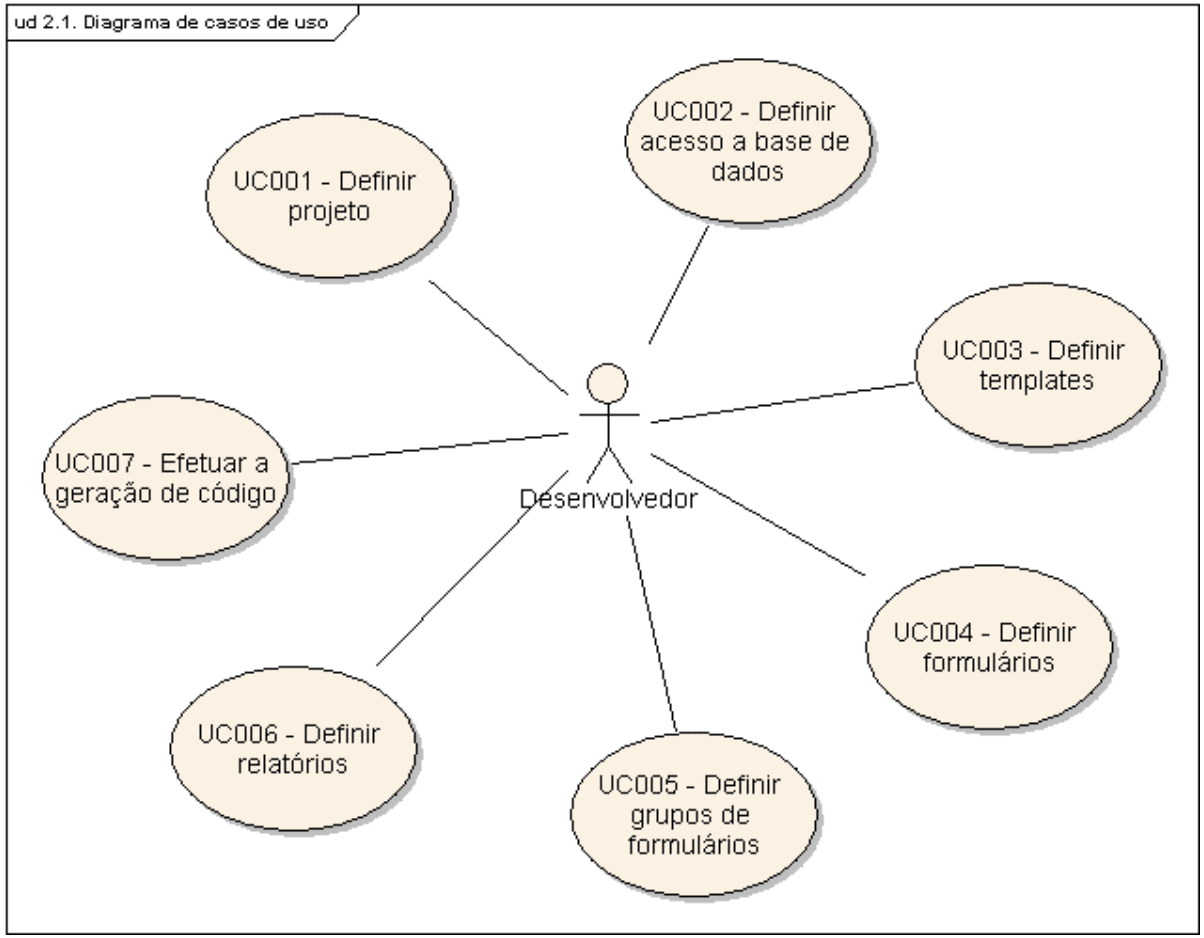
- Manter configurações do projeto
- Manter configurações de acesso ao banco de dados
- Manter configurações dos *templates*
- Manter configurações de formulários
- Manter configurações de grupos de formulários
- Manter configurações de relatórios
- Efetuar a geração de código

# Requisitos não funcionais

- Gerar páginas na linguagem JSP e no padrão MVC
- Utilizar a linguagem Java 5.0
- Utilizar o ambiente de desenvolvimento Eclipse 3.2
- Utilizar *templates* para a geração do código de saída
- Utilizar o motor de *templates* Velocity para formatação dos *templates*
- Utilizar a API JDBC para a extração das informações do metadados dos SGBDs
- Utilizar *drivers* JDBCs específicos para cada SGDB definido
- Gerar os arquivos de configuração no formato XML

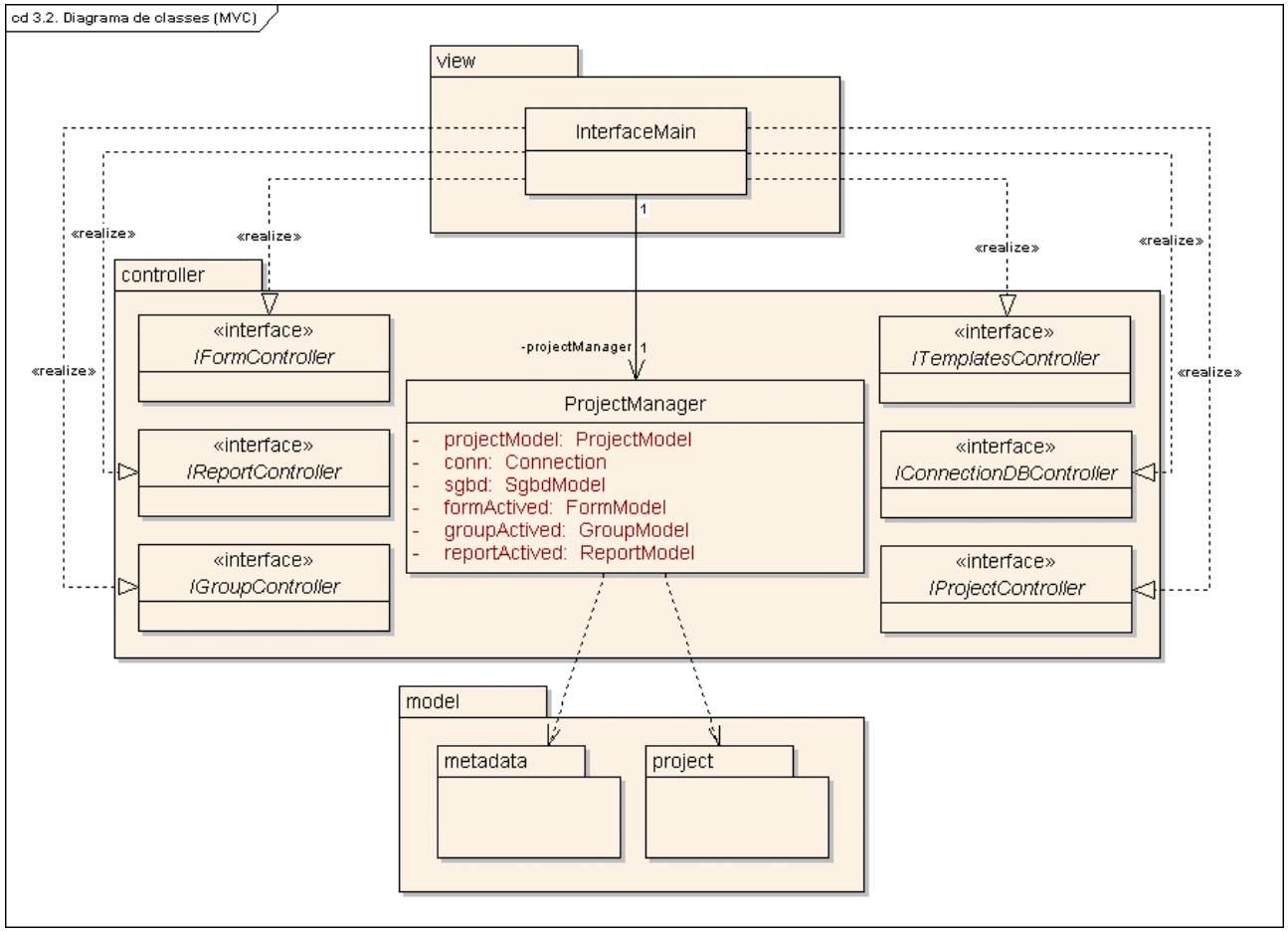
# Especificação

- Casos de uso:



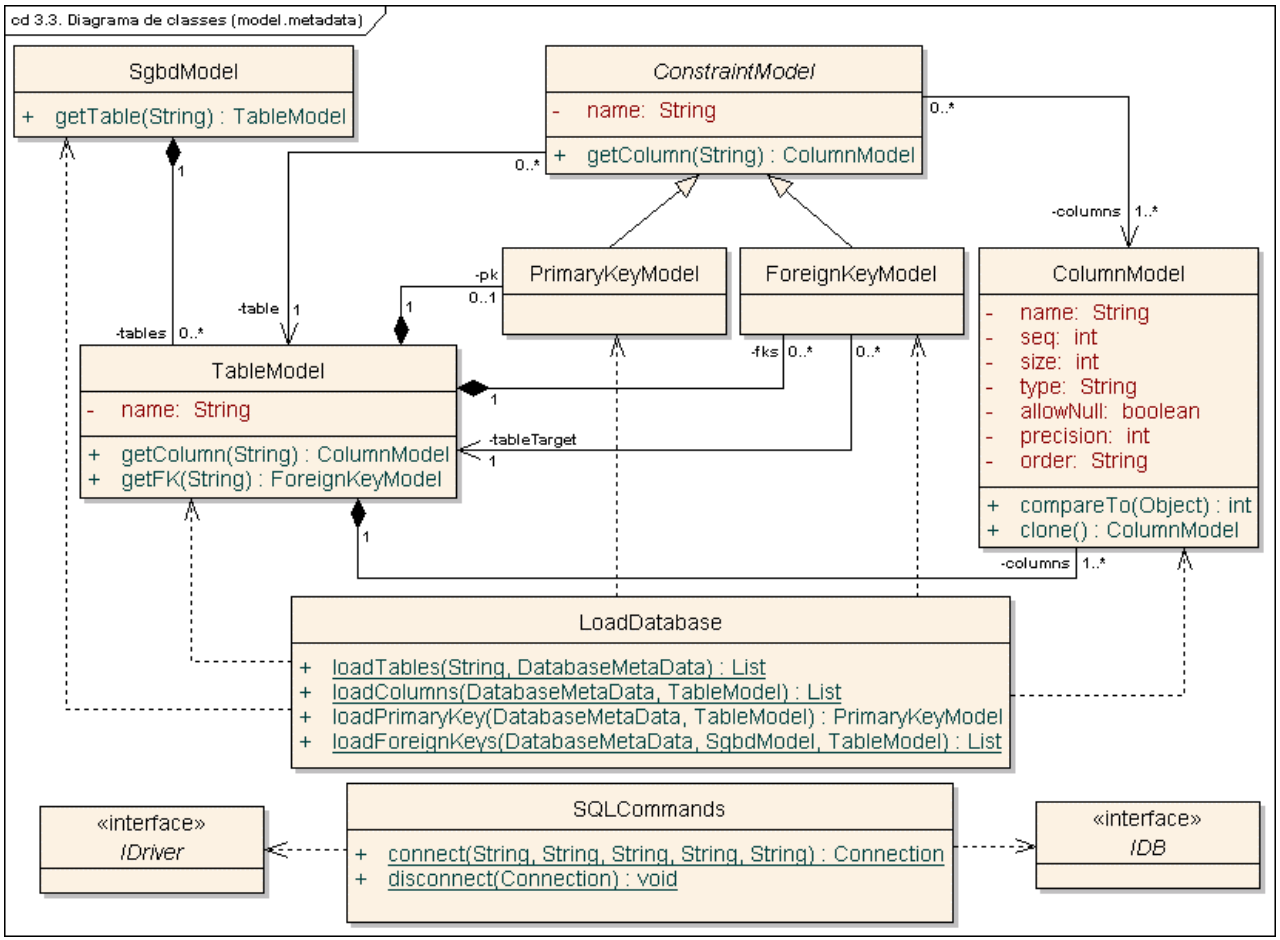
# Especificação

- Diagrama de pacotes mostrando a aplicação do padrão MVC:



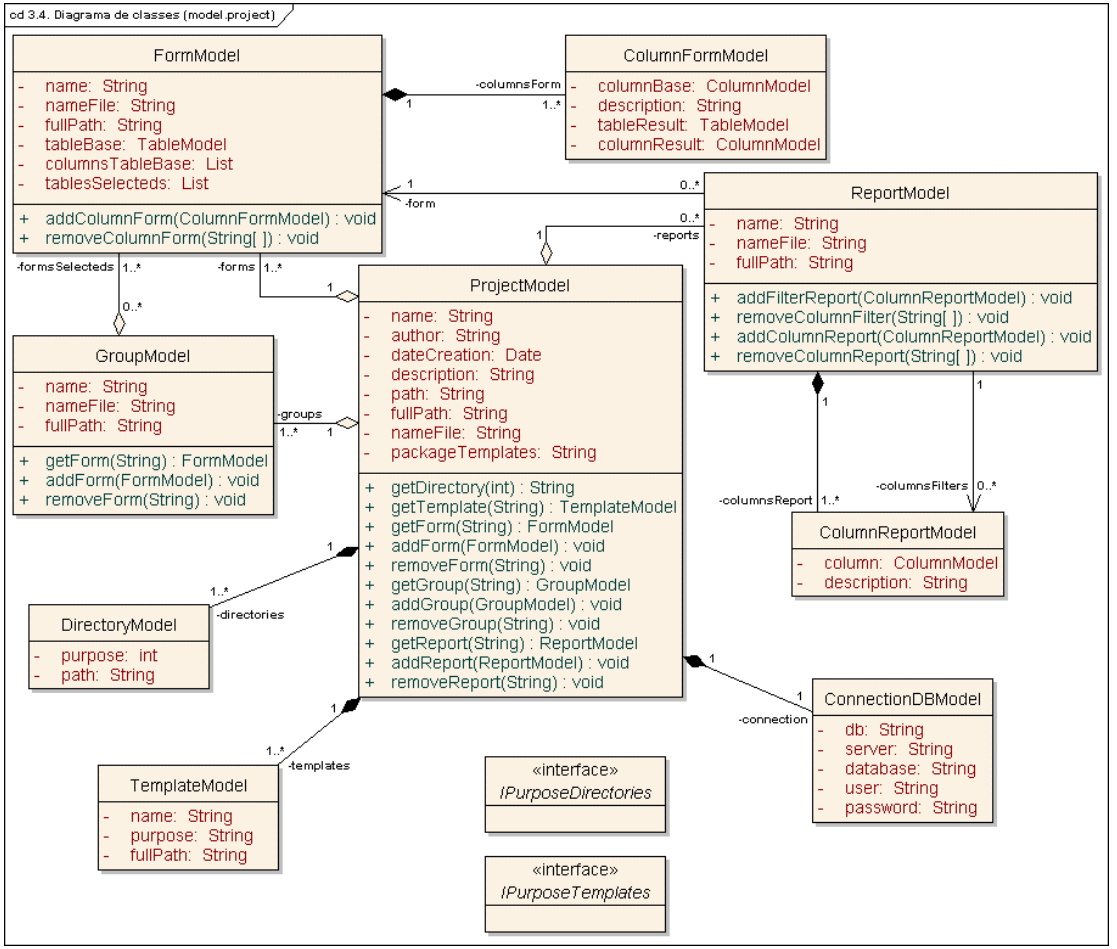
# Especificação

- Diagrama de classes do pacote `model.metadata`:



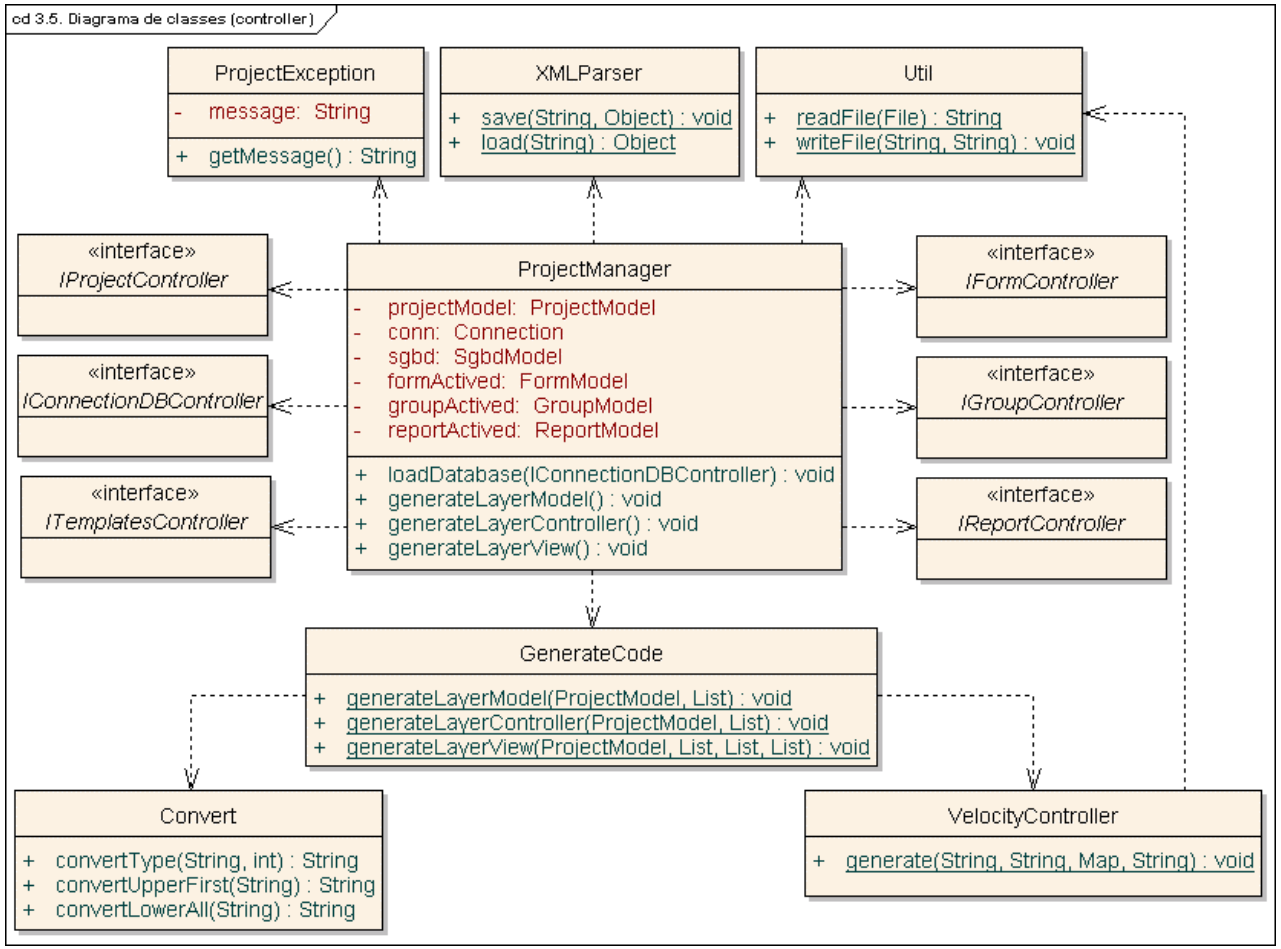
# Especificação

- Diagrama de classes do pacote `model.project`:



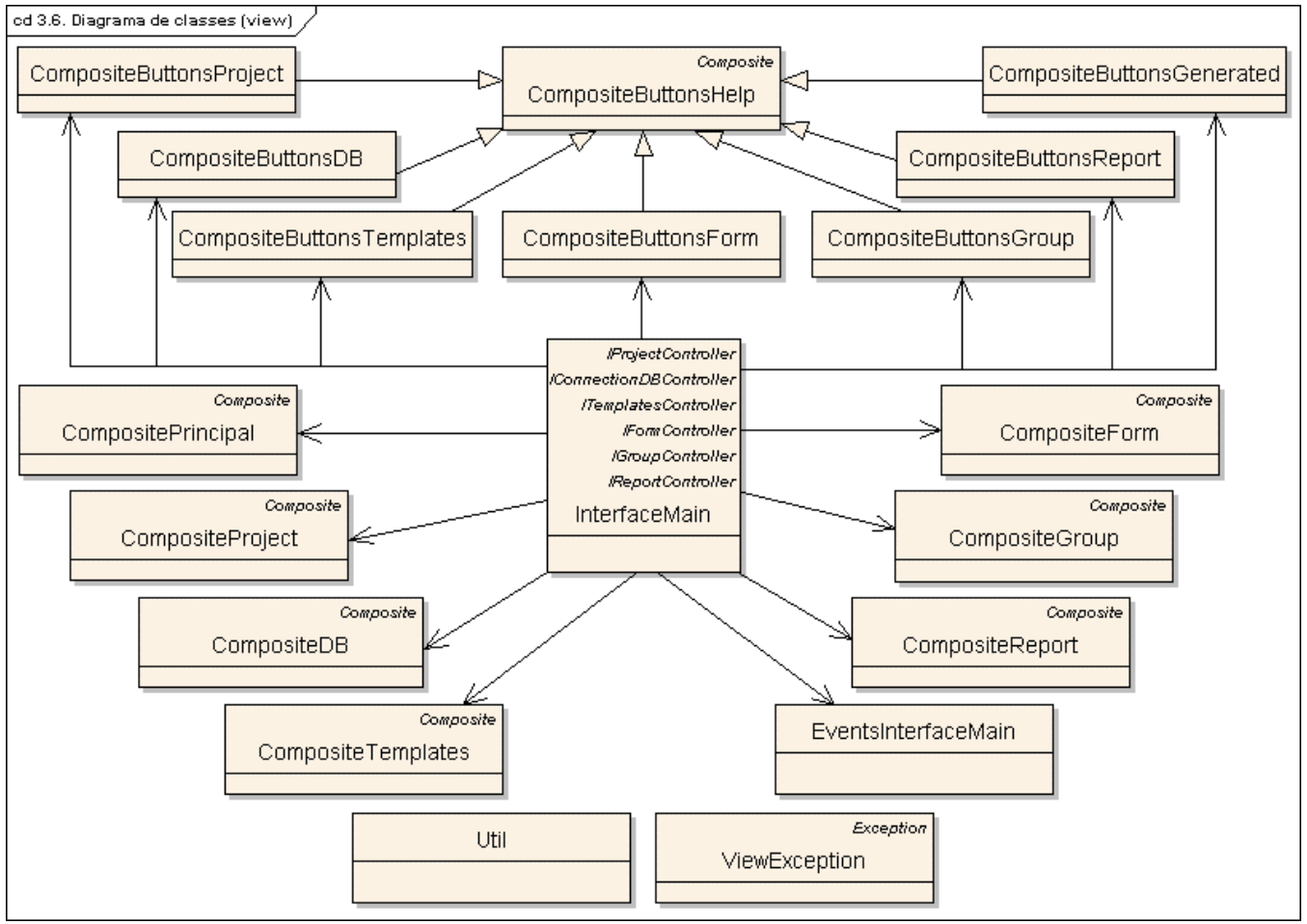
# Especificação

- Diagrama de classes do pacote `controller`:



# Especificação

- Diagrama de classes do pacote `view`:



# Implementação

- Técnicas e ferramentas utilizadas:
  - ✓ Arquitetura MVC
  - ✓ Java 5.0
  - ✓ Eclipse 3.2
  - ✓ API JDBC
  - ✓ *Drivers* JDBC apropriados
  - ✓ SWT
  - ✓ Velocity
  - ✓ *Templates*
  - ✓ JSP

# Implementação

- Método que obtém as colunas de uma tabela do banco de dados:

```
package model.metadata;

public class LoadDatabase {
   ...
   public static List<ColumnModel> loadColumns(DatabaseMetaData metadata,
                                    TableModel table) throws SQLException {

      // retorna um ResultSet contendo as colunas da tabela passada como
         parâmetro
      ResultSet rs = metadata.getColumns(null, null, table.getName(), null);
      List<ColumnModel> columns = new ArrayList<ColumnModel>();

      while ( rs.next() ) {

         // obtém as informações das colunas
         boolean nul = false;
         String name = rs.getString( "COLUMN_NAME" );
         String type = rs.getString( "TYPE_NAME" );
         int size = rs.getInt( "COLUMN_SIZE" );
         int precision = rs.getInt( "DECIMAL_DIGITS" );
         int seq = rs.getInt( "ORDINAL_POSITION" );
         String allowNull = rs.getString( "IS_NULLABLE" );

         if ( allowNull.equalsIgnoreCase( "YES" ) ) {
            nul = true;
         }

         // cria um objeto de ColumnModel
         ColumnModel column = new ColumnModel(seq, name, type, size,
                                              precision, nul, null);

         // adiciona o objeto de SeqColumnModel nas lista de colunas
         columns.add( column );

      }

      // retorna a lista de colunas da tabela
      return columns;
   }
   ...
}
```

# Implementação

- Método que efetua a geração de código:

```
package controller.generate;

public class VelocityController {

   public static void generate( String packageTemplates,
               String nameTemplate, Map context, String fileName ) throws
                     ResourceNotFoundException, ParseErrorException,
                     MethodInvocationException, IOException, Exception {

      // instancia os objetos necessários
      VelocityEngine velociteEngine = new VelocityEngine();
      Template template = new Template();
      VelocityContext velociteContext = new VelocityContext();
      Properties pathTemplates = new Properties();

      // seta o properties com o path onde encontram-se os templates
      pathTemplates.put( "file.resource.loader.path", packageTemplates );

      // inicia a engine do Velocity
      velociteEngine.init( pathTemplates );

      // obtém o template
      template = velociteEngine.getTemplate( nameTemplate );

      // seta o contexto dos objetos utilizados no merge
      for ( Iterator it = context.keySet().iterator(); it.hasNext(); ) {
         String key = (String) it.next();
         velociteContext.put( key, context.get( key ) );
      }

      // faz o merge
      StringWriter writer = new StringWriter();
      template.merge( velociteContext, writer );

      // grava o arquivo em disco
      Util.writeFile( fileName, writer.toString() );
   }
}
```

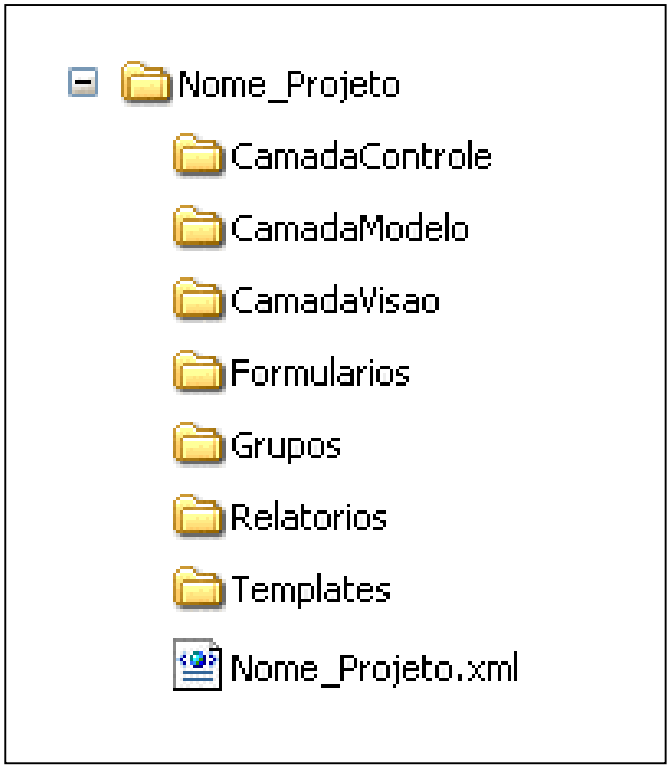
# Implementação

- *Template* utilizado na geração da tela de menu da aplicação JSP:

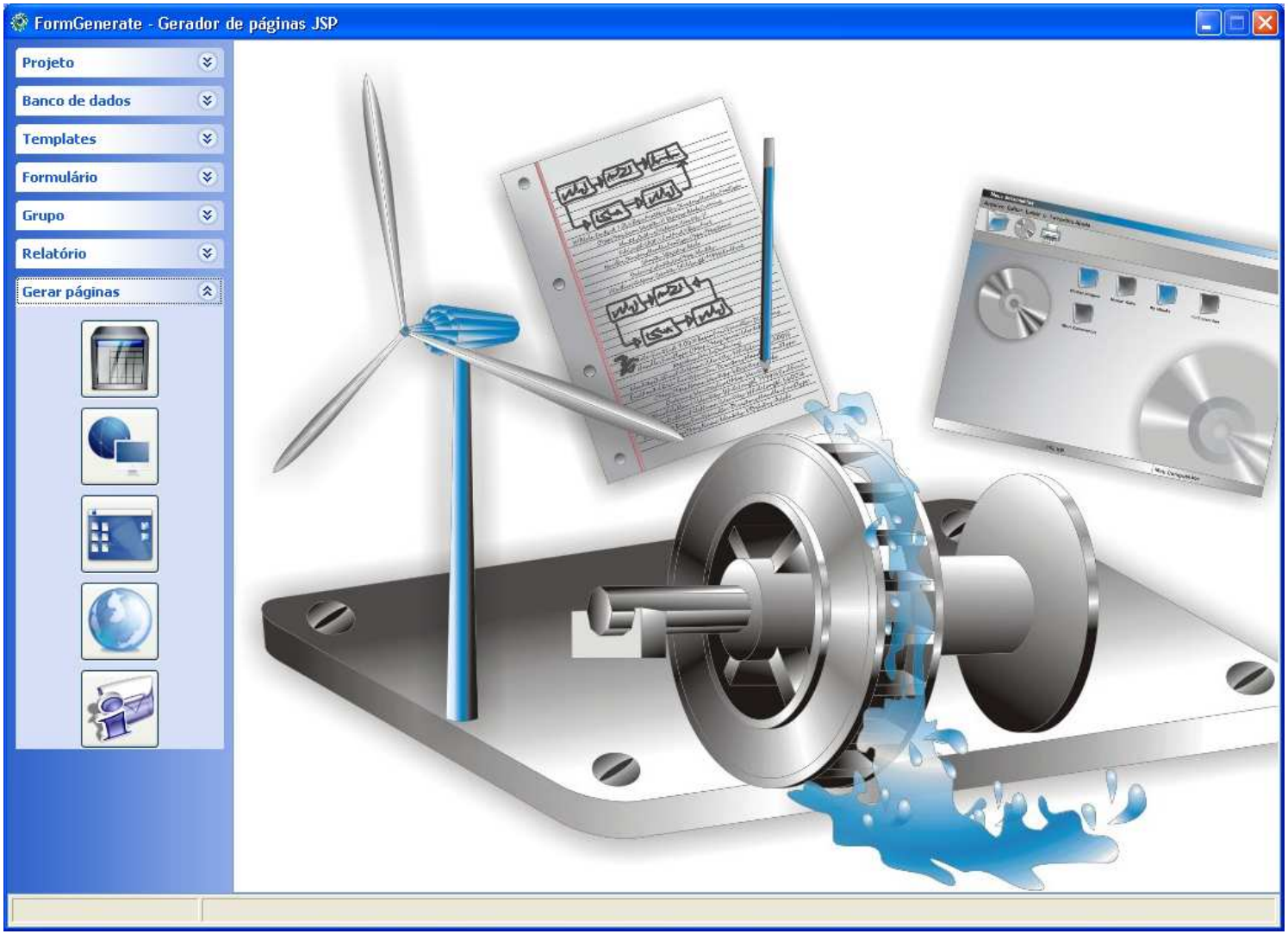
```
#macro(criaMenu $group)
  <tr><td align="center" class="tituloMenu">::$group.getName().toUpperCase()::</td></tr>
#end
#macro(criaItensMenu $form)
  #set($extension = "Index.jsp")
  #set($file = "$form.getName()")
  <tr><td align="center" class="itensMenu"><a href="$file$extension" target="main">$form.getName()</a></td></tr>
#end
#macro(criaLinkRelatorio $report)
  #set($extension = "ExecReport.jsp")
  #set($file = "$report.getName()")
  <tr><td align="center" class="itensMenu"><a href="$file$extension" target="main">$report.getName()</a></td></tr>
#end
<html>
<head>
  <link rel="stylesheet" type="text/css" href="folhaEstilo.css">
  <title>SysVideo</title>
  <script src="scripts.js"></script>
</head>
<body class="bodyMenu">
<table align="center">
  <tr><td align="center"><a href="conectado.jsp" target="main">
    <img src="images/mConectar.gif" border="0" alt="Conectar"></a></td>
    <td align="center"><a href="main.jsp" target="main">
    <img src="images/mHome.gif" border="0" alt="Página Inicial"></a></td>
    <td align="center"><a href="desconectado.jsp" target="main">
    <img src="images/mSair.gif" border="0" alt="Sair"></a></td>
  </tr>
</table>
<table align="center">
  #foreach($group in $groups)
    #criaMenu($group)
    #foreach($form in $group.getFormsSelecteds())
      #criaItensMenu($form)
    #end
  #end
  #set($size = $reports.size())
  #if($size > 0)
    <tr><td align="center" class="tituloMenu">::RELATÓRIOS::</td></tr>
      #foreach($report in $reports)
        #criaLinkRelatorio($report)
      #end
  #end
</table>
</body>
</html>
```

# Implementação

- Estrutura de diretórios gerada pela ferramenta:



# Operacionalidade



# Resultados e discussão

- Atendimento dos requisitos propostos
- Mantidas funcionalidades do CodGer (Menin, 2005)
- Código gerado totalmente externo ao código da ferramenta
- Comparativo entre as ferramentas:

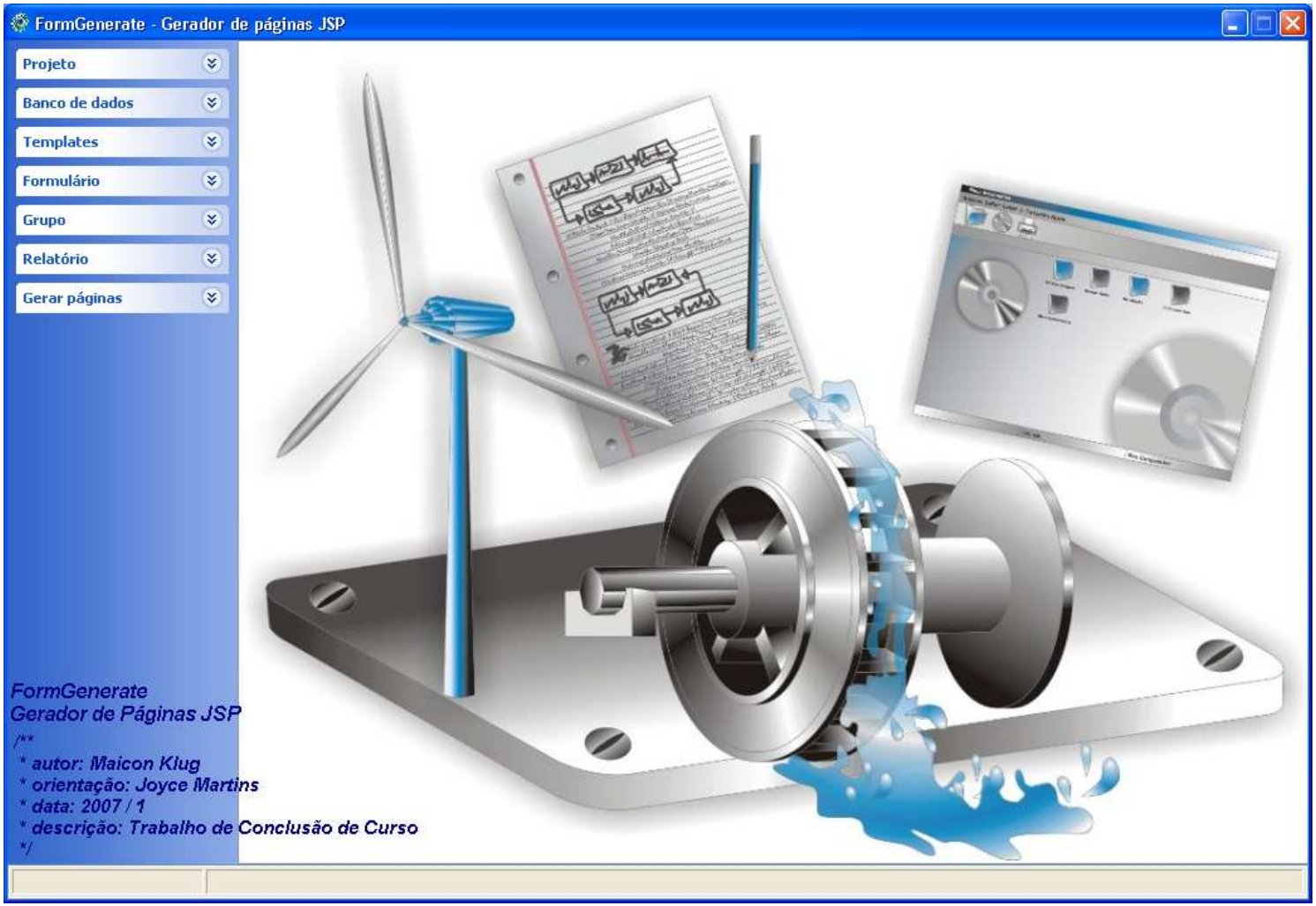
| CARACTERÍSTICAS | CodGer | FormGenerate |
|---|---|---|
| desenvolvimento utilizando o conceito de orientação a objetos | sim | sim |
| desenvolvimento utilizando o padrão MVC | não | sim |
| uso de *templates* para geração de código | não | sim |
| utilização de *driver* JDBC para a conexão com o banco de dados | sim | sim |
| permite conexão com vários SGDBs | não | sim |
| utilização do motor de *templates* Velocity | não | sim |
| geração de código utilizando o padrão MVC | sim | sim |

# Conclusão

- Acesso ao banco de dados:
  - ✓ Utilização de *drivers* JDBC
  - ✓ Forma de conexão padronizada
  - ✓ Conexão com vários bancos de dados
- Flexibilidade
  - ✓ Configuração dos formulários efetuada pelo desenvolvedor
  - ✓ Código gerado definido em *templates*
- Geração de código
  - ✓ Velocity
  - ✓ *Templates*
- Limitações
  - ✓ Tipos de dados do banco de dados
  - ✓ Formulários mestre-detalhe

# Extensões

- Disponibilizar acesso a mais bancos de dados
- Possibilitar informar o componente gráfico do campo do formulário no momento da configuração
- Gerar versão para outro sistema operacional


FormGenerate - Gerador de páginas JSP
Projeto
Banco de dados
Templates
Formulário
Grupo
Relatório
Gerar páginas
FormGenerate
Gerador de Páginas JSP
/**
* autor: Maicon Klug
* orientação: Joyce Martins
* data: 2007 / 1
* descrição: Trabalho de Conclusão de Curso
*/

Obrigado !!!