Leandro Salvatti Piscke

FRAMEWORK EM JAVA PARA GERAÇÃO DE TELAS NO MODELO CRUD BASEADO EM XML E OBJETOS REMOTOS UTILIZANDO A ARQUITETURA MVC E PADRÕES

Orientador:
Prof. Adilson Vahldick

Roteiro de Apresentação

- Introdução
- Fundamentação Teórica
- Desenvolvimento do Sistema
- Implementação
- Conclusão

Introdução

- Problemas
  - Arquiteturas de software complexas
  - Sistemas mal arquitetados e sem uso de padrões
  - Sistemas que misturam camadas trazem limites impostos na reutilização
  - Repetição do modelo CRUD em sistemas de informação

Introdução

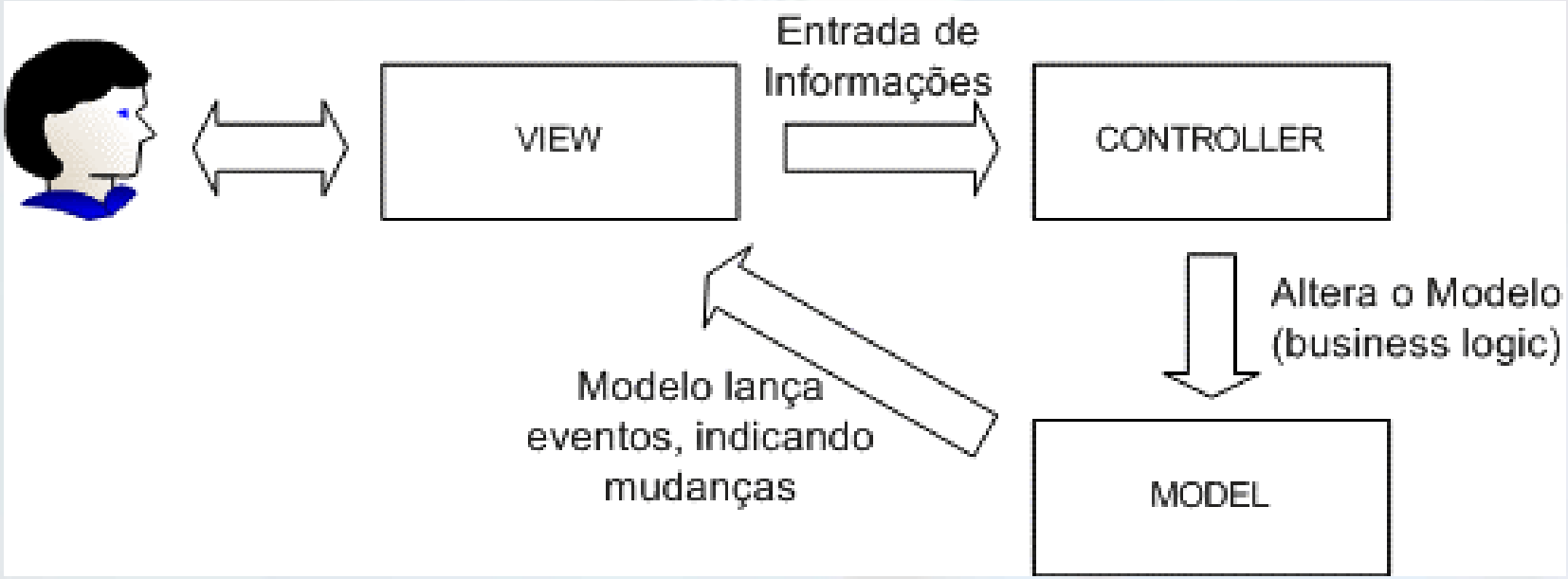
- Soluções
  - *Framework*
  - Padrões de projetos
  - Arquitetura MVC
  - Geração de telas no modelo CRUD
  - Tecnologias: EJB3, Thinlet, HttpClient, Custom Tags e BeanUtils

Introdução

- Objetivos
  - Gerar uma tela a partir de um XML e um DTO seguindo o modelo CRUD
  - Implementar a camada servidora do *framework*
  - Implementar a camada cliente do *framework*
  - Disponibilizar recursos para desenvolvedor acessar componentes visuais e o DTO
  - Desenvolver protótipo de um sistema de locação de vídeos utilizando o *framework* desenvolvido

Fundamentação Teórica

- MVC
  - Separa as camadas modelo, visualização e controle para aumentar a flexibilidade e a reutilização



Fundamentação Teórica

- CRUD
  - O modelo CRUD sugere as operações básicas para a persistência de objetos:
    - Criação
    - Leitura
    - Alteração
    - Exclusão

Fundamentação Teórica

- Padrões no servidor
  - Transfer Object
  - Data Access Object
  - Business Object
  - Session Facade
  - Composite View
- Padrões no cliente
  - Service Locator
  - Business Delegate

Fundamentação Teórica

- EJB3
  - Resolve problemas como escalabilidade, aplicações distribuídas e gerenciamento de ciclo de vida dos objetos
  - É uma simplificação profunda da especificação EJB
  - Focaliza mais no desenvolvimento de POJO

Fundamentação Teórica

- Thinlet
  - *Framework* para construção de interfaces gráficas AWT representadas através de um XML

```
<panel gap="4" top="4" left="4">
      <textfield name="number1" columns="4" />
      <label text="+" />
      <textfield name="number2" columns="4" />
      <button text="=" action="calculate(number1.text,
            number2.text, result)" />
      <textfield name="result" editable="false" />
</panel>
```

Fundamentação Teórica

- BeanUtils
  - Acesso dinâmico de objetos que implementam o padrão getters e setters

```
public class Employee {
      private String firstName;
      private String lastName;
      public String getFirstName();
      public void setFirstName(String firstName);
      public String getLastName();
      public void setLastName(String lastName);
}
```

```
// Resgatar atributos da classe
String firstName = (String)
      PropertyUtils.getSimpleProperty(employee, "firstName");
String lastName = (String)
      PropertyUtils.getSimpleProperty(employee, "lastName");

// Atribuir atributos da classe
PropertyUtils.setSimpleProperty(employee, "firstName", firstName);
PropertyUtils.setSimpleProperty(employee, "lastName", lastName);
```

Fundamentação Teórica

- Custom Tags
  - É uma tag definida pelo usuário

```
public class LabelEditTag extends TagSupport {
          public int doStartTag() {
             String res = "<label alignment=\"right\" text=\""+
                          getText() +"\"/>";
             res +="<textfield weightx=\"1\" name=\""+
                          getName() +"\"/>";
          try {
                  pageContext.getOut().print(res);
             } catch (IOException e) {
                  e.printStackTrace();
             }
          return SKIP_BODY;
      }
}
```

Fundamentação Teórica

Trabalhos Correlatos

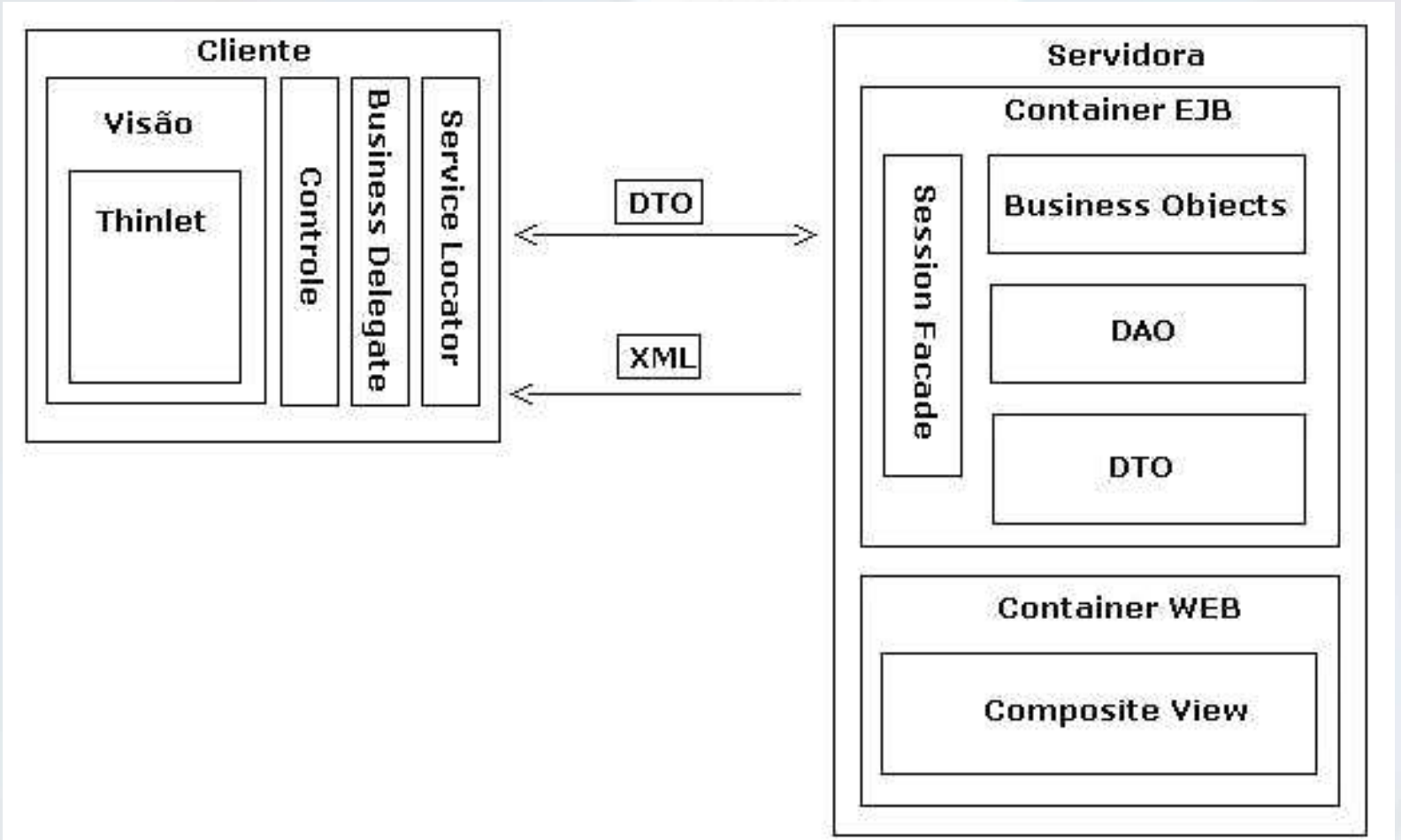
- Genesis
  - *É um framework* que tem como objetivo simplificar e tornar produtivo o desenvolvimento de aplicações corporativas
- PATI–MVC
  - Padrões no desenvolvimento de aplicações de sistema de informação baseados na arquitetura MVC

Desenvolvimento do Trabalho

- Requisitos principais
  - Implementar funcionalidades do modelo CRUD na camada de controle
  - Implementar a camada de controle para ser extensível pelo usuário
  - Instanciar e gerenciar a persistência dos DTO na camada de modelo
  - Ser implementado utilizando a especificação EJB3
  - Utilizar a arquitetura MVC e Padrões de Projeto

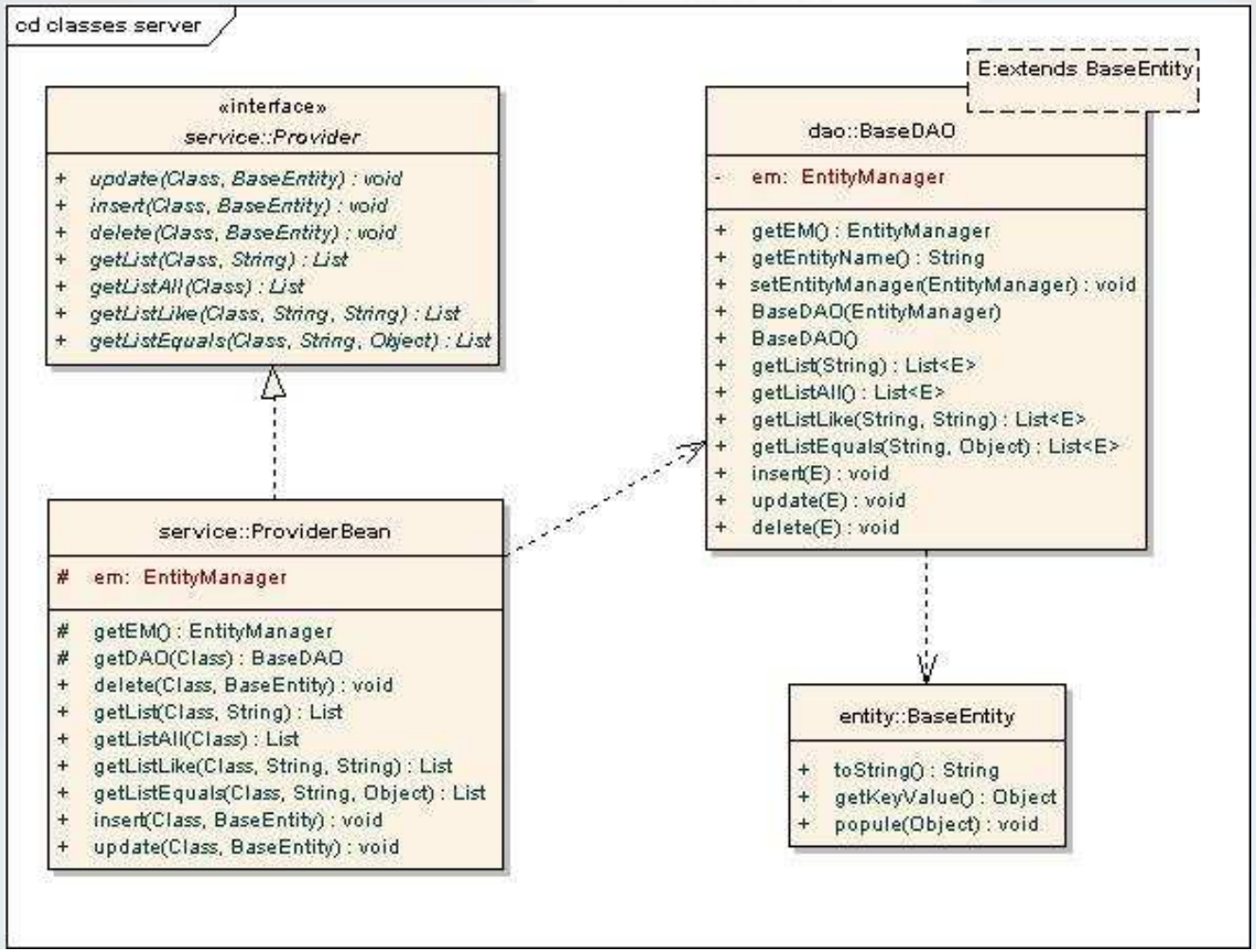
Desenvolvimento do Trabalho

Arquitetura



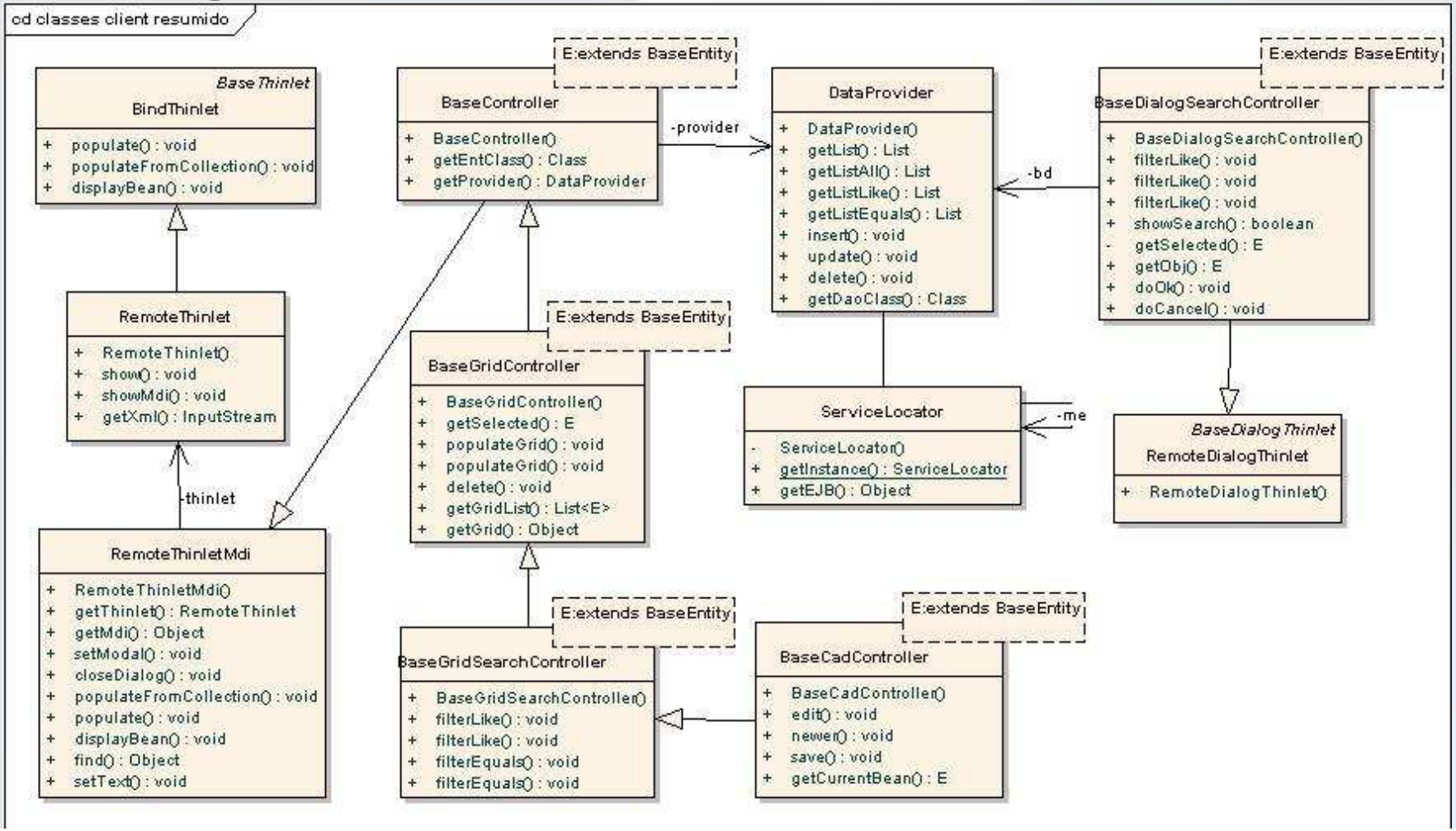
Desenvolvimento do Trabalho

Diagrama de classes – servidor *framework*



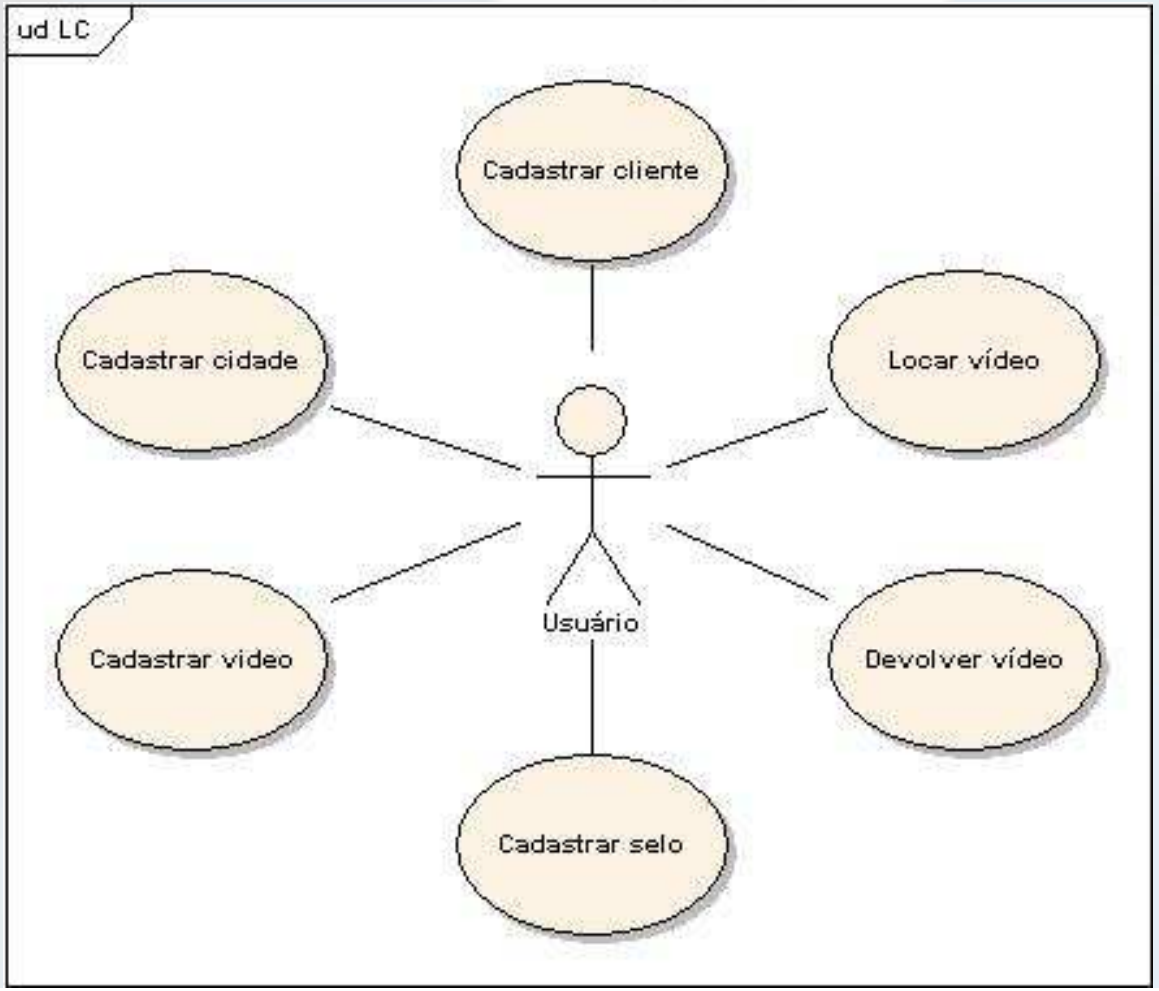
Desenvolvimento do Trabalho

Diagrama de classes – cliente *framework*



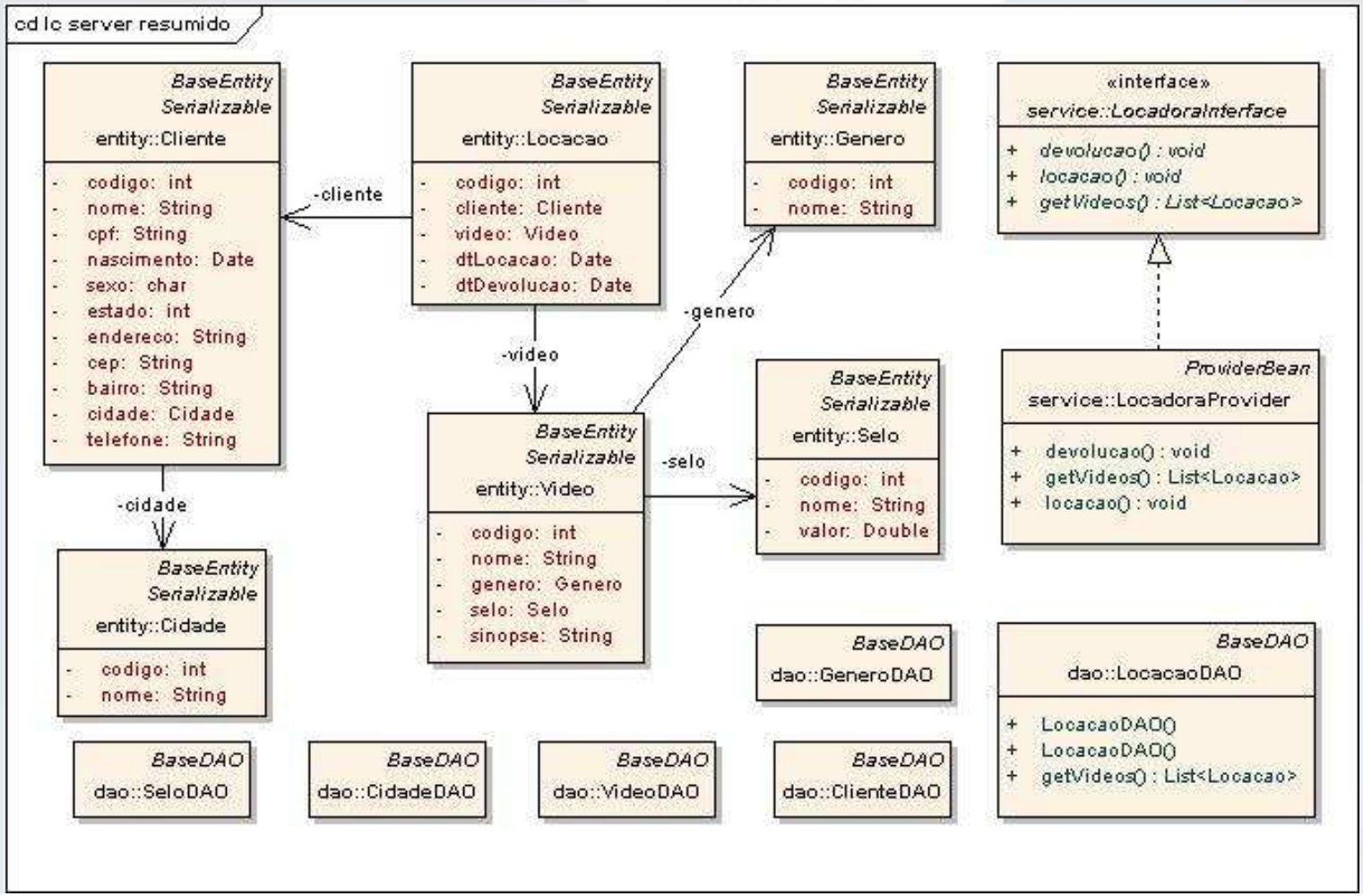
Desenvolvimento do Trabalho

Diagrama de caso de uso – protótipo



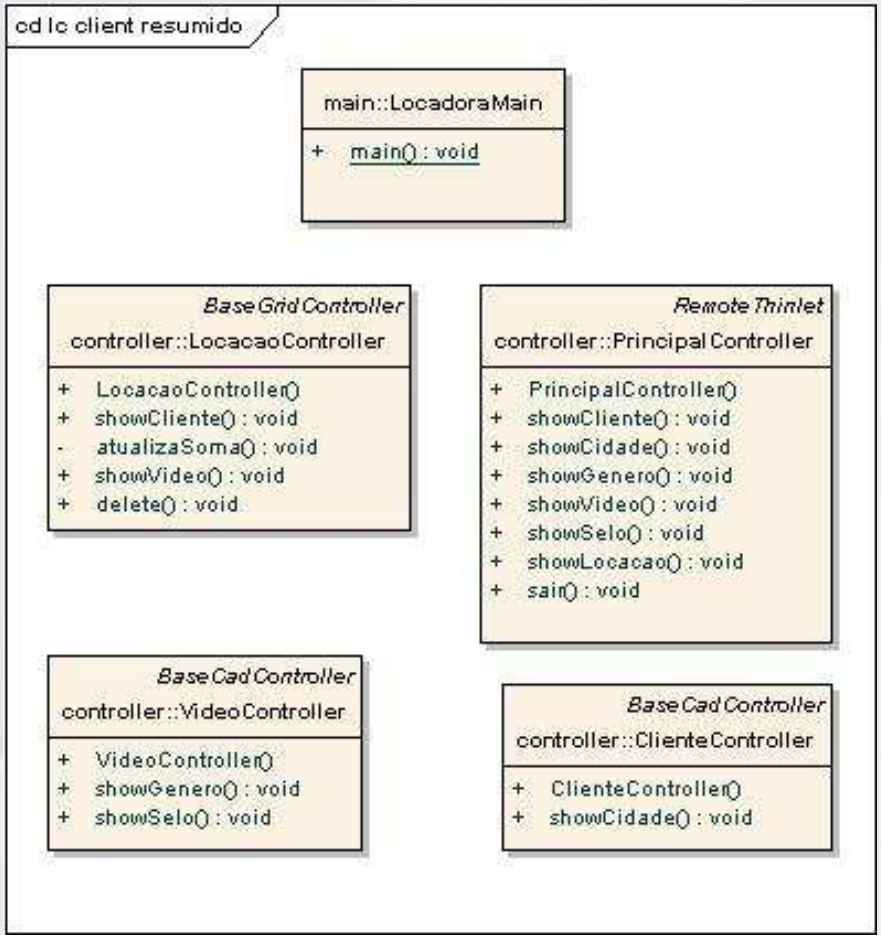
Desenvolvimento do Trabalho

Diagrama de classes – servidor protótipo



Desenvolvimento do Trabalho

Diagrama de classes – cliente protótipo



Implementação

- Técnicas e Ferramentas
  - JBoss IDE para Eclipse
  - ThinG
  - Servidor de aplicação JBoss
  - EJB3

Implementação

- *Framework* para a camada servidora
  - Utilizado o objeto EntityManager que atua no contexto de persistência dos EJB de entidade
  - *O framework* utiliza objetos DAO implementados pelo usuário

Implementação

- *Framework* para a camada cliente
  - Implementado RemoteThinlet utilizando componente HttpClient
  - Faz a ligação do DTO e os componentes visuais do Thinlet, utilizando binding do *framework* Genesis

Implementação

Implementando objeto relacional (DTO)

```
@Entity
@Table(name = "cliente")
public class Cliente extends BaseEntity implements Serializable {
	private int codigo;
	private Cidade cidade;
	@Id
	@GeneratedValue
	public int getCodigo() {
		return codigo;
	}
	public void setCodigo(int codigo) {
		this.codigo = codigo;
	}
	@ManyToOne(optional = false)
	@JoinColumn(name="CIDADE", nullable=false)
	public Cidade getCidade() {
		return cidade;
	}
	public void setCidade(Cidade cidade) {
		this.cidade = cidade;
	}
}
```

Implementação

Implementando classe DAO

```
import locadora.server.entity.Cliente;
import locadora.server.entity.Locacao;
import piscke.server.dao.BaseDAO;

public class LocacaoDAO extends BaseDAO<Locacao> {

	public LocacaoDAO(EntityManager entityMnger) {
		super(entityMnger);
	}

	public LocacaoDAO() {
		super();
	}

	public List<Locacao> getVideos(Cliente cli){
		Query q = getEM().createQuery("from Locacao where "+
		    "cliente.codigo=:codigo and dtDevolucao is null");
		q.setParameter("codigo", cli.getCodigo());
		return q.getResultList();
	}
}
```

Implementação

Implementando regra de negócio (BO)

```
public @Stateless
class LocadoraProvider extends ProviderBean implements Provider,
            LocadoraInterface {

      public void devolucao(Locacao loc) {
            LocacaoDAO dao = new LocacaoDAO(getEM());
            loc.setDtDevolucao(new Date());
            dao.update(loc);
      }

      public List<Locacao> getVideos(Cliente cli) {
            LocacaoDAO dao = new LocacaoDAO(getEM());
            return dao.getVideos(cli);
      }

      public void locacao(Locacao loc) {
            loc.setDtLocacao(new Date());
            LocacaoDAO dao = new LocacaoDAO(getEM());
            dao.insert(loc);
      }
}
```

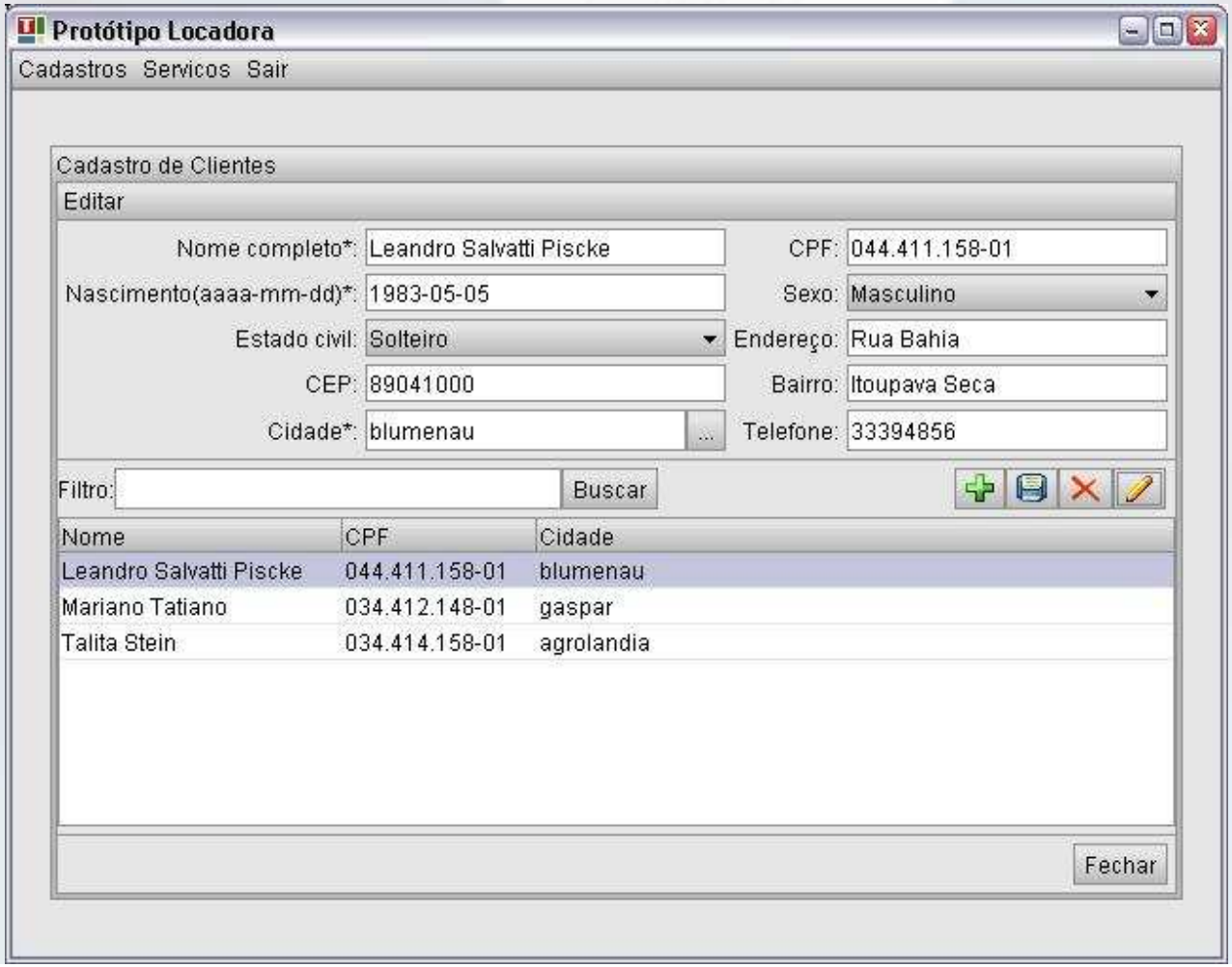
Implementação

Criando XML da tela

```
<%@ taglib uri="http://piscke.server.org/tags" prefix="fk"%>
<dialog columns="1" gap="4" height="400" resizable="true" scrollable="true"
width="500">
    <jsp:include page="inc/menu.jsp"></jsp:include>
    <panel columns="4" gap="4" left="4" right="4">
            <fk:labeledit name="nome" text="Nome:"></fk:labeledit>
            <fk:labellookup name="cidade" text="Cidade:" buttonText="..."
                  action="showCidade"></fk:labellookup>
    </panel>
    <panel columns="2">
      <fk:editsearch name="filter" fieldSearch="nome"
            buttonText="Buscar" text="Filtro:"
            action="filterLike(filter)"></fk:editsearch>
      <jsp:include page="inc/navegador.jsp"></jsp:include>
    </panel>
    <table name="grid" weightx="1" weighty="1">
        <header>
            <column name="nome" text="Nome" width="93"/>
            <column name="cpf" text="CPF" width="103"/>
            <column name="cidade" text="Cidade"/>
        </header>
    </table>
    <jsp:include page="inc/fechar.jsp"></jsp:include>
</dialog>
```

Implementação

Tela do cadastro de clientes



Implementação

Implementando classe controladora

```
import locadora.server.dao.CidadeDAO;
import locadora.server.dao.ClienteDAO;
import piscke.client.controller.BaseCadController;
import piscke.client.controller.BaseDialogSearchController;

public class ClienteController extends BaseCadController<Cliente> {

	public ClienteController(RemoteThinlet thin) throws Exception {
		super(thin, "cad_cliente.jsp", new
			DataProvider(ClienteDAO.class), "grid");
	}

	public void showCidade() throws Exception {
			BaseDialogSearchController<Cidade> cli = new
		BaseDialogSearchController<Cidade>(
			getThinlet(), "lookup.jsp", new
			DataProvider(CidadeDAO.class), "grid", "nome");
		if (cli.showSearch()) {
			getCurrentBean().setCidade(cli.getObj());
			setText(find("cidade"), cli.getObj().getNome());
		}
	}
}
```

Conclusão

- Conclusão
  - Projetar um software reutilizável e orientado a objetos
  - Reutilização de *frameworks* e componentes de terceiros
  - O *framework* foca na implementação da lógica de negócio da aplicação

Conclusão

- Limitações
  - Formatação de campos
  - Notificação das telas Thinlet quando o modelo é alterado

Conclusão

- Extensões
  - Desenvolver novas Custom Tags
  - Para as Custom Tags desenvolvidas também será necessário criar classes controladoras
  - Criar novas classes controladoras para telas que represente dados de forma diferentes
  - Implementar notificação das telas Thinlet quando o modelo é alterado