FERRAMENTA PARA CONSTRUÇÃO DE INTERFACES DE SOFTWARE A PARTIR DE DIAGRAMA DE CLASSES

Aluno: André Luis Becker

Orientador: Prof. Everaldo Artur Grahl.

Mestre – Orientador, FURB

Roteiro da Apresentação

- Introdução;
- Objetivos do trabalho;
- Fundamentação teórica;
- Desenvolvimento;
- Conclusão.

Introdução

- Muitas empresas estão utilizando diversas técnicas oferecidas pela Engenharia de Software para auxiliar na qualidade e produtividade;
- Criou-se a *Unified Modeling Language* (UML) que é uma notação padronizada;
- Diagrama de classes um dos mais importantes diagramas estruturais da UML;

Introdução

- EA é uma ferramenta CASE para modelagem e permite a exportação dos diagramas e automatização de alguns processos;

- No que diz a automatização de construção de interface o EA não da suporte;

- Construção de interfaces é um papel importante, cauteloso e demorado, com base nisso surgiu a idéia de gerar interfaces de software *standalone* a partir de um diagrama de classes, gerado para nova tecnologia WPF;

Objetivos do trabalho

1) Objetivo Geral:

- construir interfaces conforme critérios ergonômicos de usabilidade a partir de diagrama de classes;

2) Objetivos Específicos:

- importar o diagrama de classes gerado pela ferramenta EA;
- traduzir as informações obtidas para linguagem XAML;
- adptar as informações para gerar interfaces gráficas para aplicações *standalone*, conforme critérios ergonômicos de usabilidade;
- permitir configurar valores para atender os critérios.

Fundamentação Teórica

- Problemas de usabilidade:
  - Relacionam-se com as coisas que impedem o usuário de interagir com o sistema.

- Alguns problemas encontrados:
  - Aparência da interface;
  - Muitas funcionalidades;
  - Nome das funcionalidades em outra linguagem;
  - Mensagens informativas não fornecem clareza.

Fundamentação Teórica

- Critério Ergonômicos:
  - Qual finalidade?
    - **R**: Avaliar a usabilidade da interface do sistema com finalidade de identificar problemas de interação e apresentar soluções ergonômicas.
  - Lista dos principais critérios conforme Bastien e Scapin(1993):
    - condução;
    - carga de trabalho;
    - controle explícito;
    - adaptabilidade;
    - gestão de erros;
    - homogeneidade;
    - significado dos códigos e denominações;
    - compatibilidade.

Fundamentação Teórica

- Windows Presentation Foundation(WPF):
  - O que é?
    R: Um conjunto de classes que faz parte do *Framework 3.0.*
  - Função?
    R: Substituir o *Windows Forms* na criação de aplicativos *standalone.*
  - Características?
    R: flexibilidade da interface, reconhecimento de voz, *layouts* avançados, 3D, animações e separação entre design e código.
  - Pilar WPF?
    R: XAML. Conforme Farias(2005), XAML é a nova linguagem de marcação usada para criar interfaces de forma simples e rápida.

Fundamentação Teórica

- Diagrama de Classes:
  - Utilizado na construção do modelo de classes;
  - Níveis de abstração;
  - Ilustrativa;

Fundamentação Teórica

- Enterprise Architect(EA):
  - ferramenta para análise e desenvolvimento de aplicações usando UML;
  - diversas funcionalidades;

- XMI:
  - formato padrão;
  - objetivo de possibilitar o compartilhamento de modelos;
  - componente principais:
    - DTDs;
    - Regras de produção de documento XML.

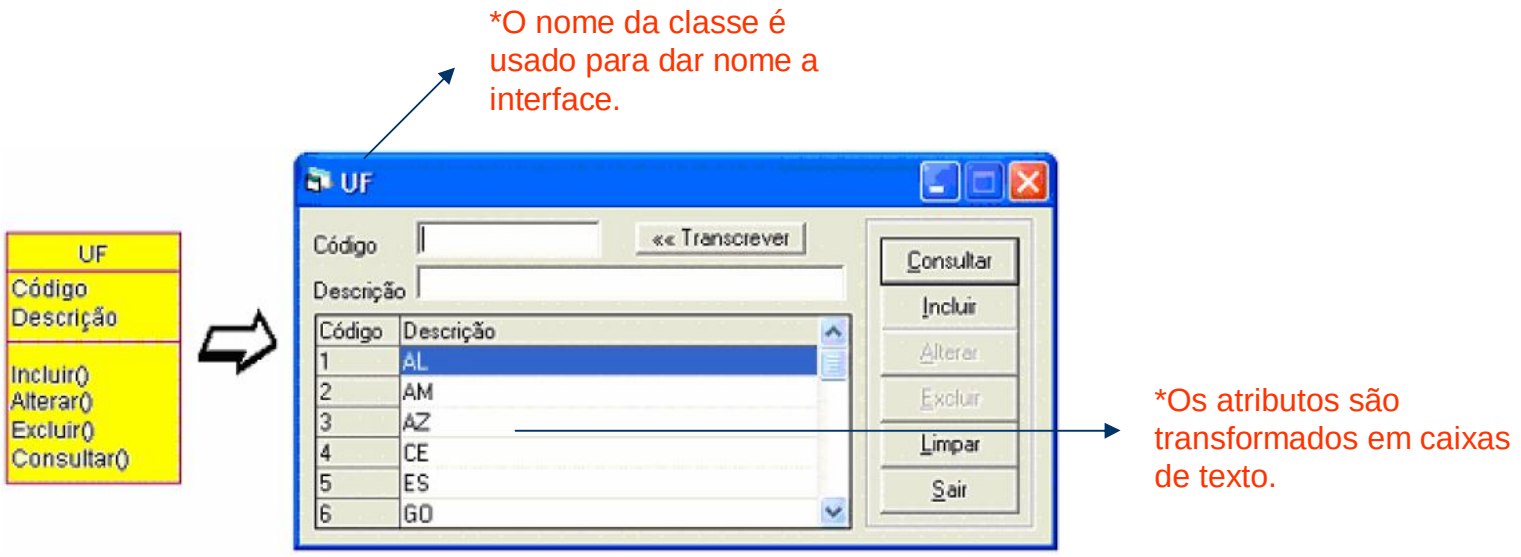
Fundamentação Teórica

Trabalho Correlato

- Ferramenta para padronização de interfaces de informação (PIRES e SIQUEIRA, 2004):
  - Idéia: gerar interfaces a partir de diagrama de classes;
  - adota *templates*;

Fundamentação Teórica

- Exemplo de interface gerada:



Especificação

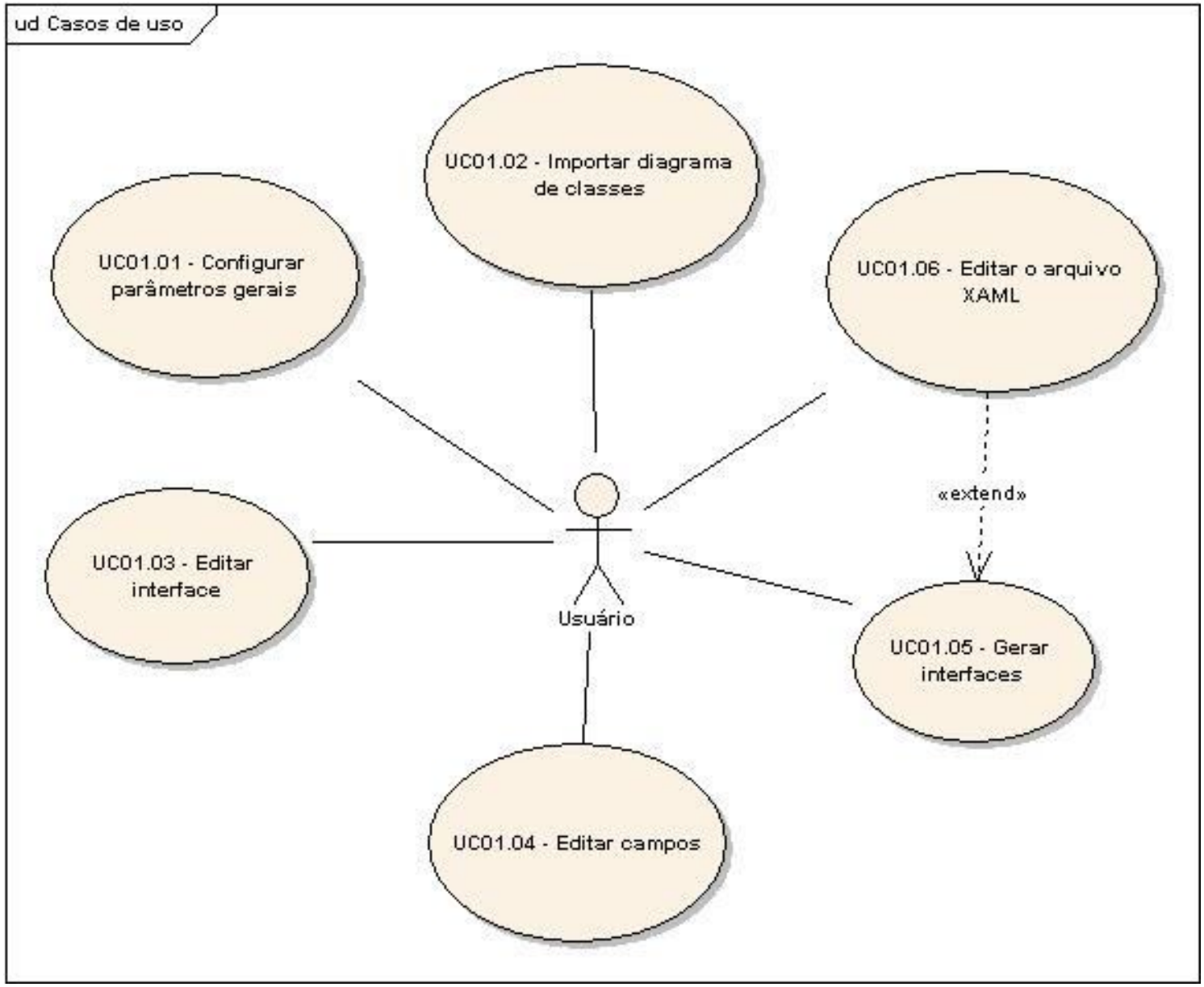
- Os principais Requisitos Funcionais:
  - Configurar as máscaras de data, hora, valor e telefone;
  - Definir as teclas de atalhos;
  - Definir os ícones dos botões de inserir, cancelar, gravar, excluir, fechar e procurar;
  - Selecionar o arquivo XMI a ser importado;
  - Editar a altura, largura e visibilidade de cada interface;
  - Escolher a pasta do projeto onde deseja salvar os *templates*, bibliotecas e as interfaces geradas;
  - Visualizar e editar os arquivos XAML gerados para cada interface.

Especificação

- Requisito Não-Funcionais:
  - A ferramenta deve utilizar .NET Framework 3.0 para ser compatível com a nova tecnologia WPF para geração de interfaces no formato XAML. Ser implementado usando o ambiente Microsoft Visual C# 2008 Express Edition;
  - Utilizar o arquivo XML para armazenar as configurações dos parâmetros gerais de usabilidade;
  - Ser compatível com o sistema operacional Windows.

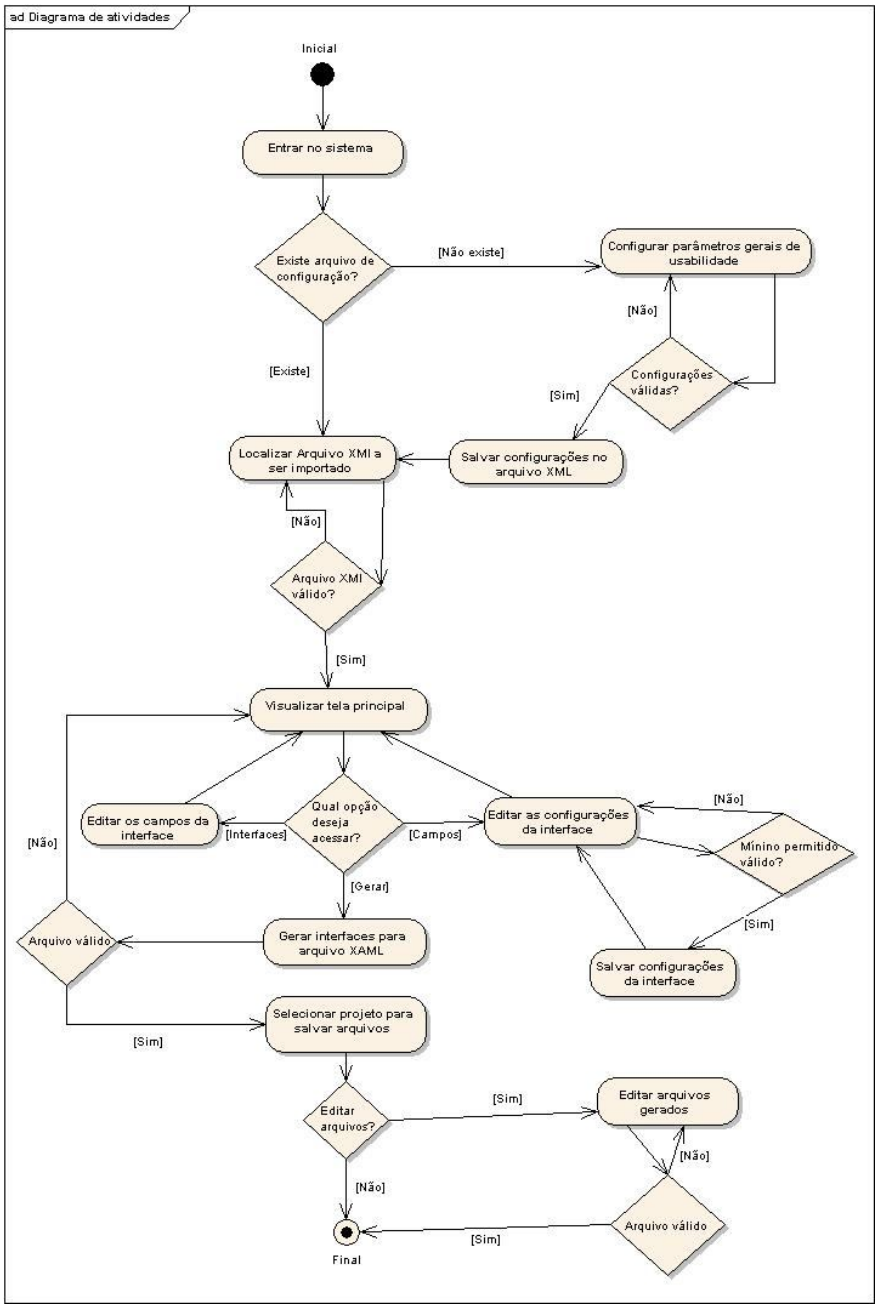
Especificação

- Diagrama de Casos de Uso:
  - É representado por um único ator que é o usuário do sistema;
  - A figura a seguir mostra o diagrama de casos de uso do aplicativo.


ud Casos de uso
UC01.02 - Importar diagrama de classes
UC01.01 - Configurar parâmetros gerais
UC01.06 - Editar o arquivo XAML
«extend»
UC01.03 - Editar interface
Usuário
UC01.05 - Gerar interfaces
UC01.04 - Editar campos

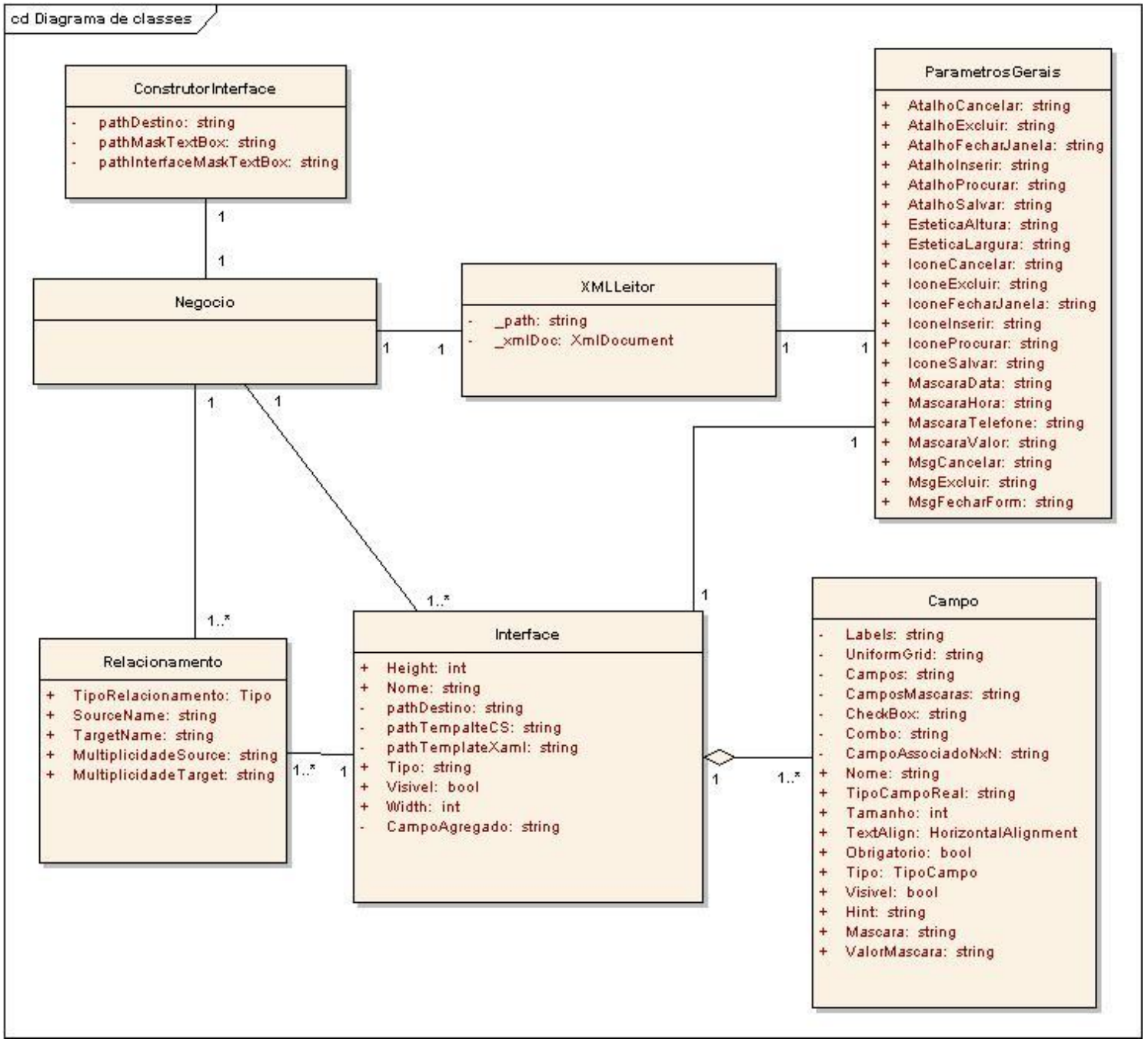
Especificação

- Diagrama de Atividades:
  - Na figura a seguir é ilustrado o processo para a geração das interfaces pela ferramenta, contemplando todos os casos de uso.


ad Diagrama de atividades
Inicial
Entrar no sistema
Existe arquivo de configuração?
[Não existe]
Configurar parâmetros gerais de usabilidade
[Não]
Configurações válidas?
[Sim]
[Existe]
Localizar Arquivo XMl a ser importado
Salvar configurações no arquivo XML
[Não]
Arquivo XMl válido?
[Sim]
Visualizar tela principal
Editar os campos da interface
[Interfaces]
Qual opção deseja acessar?
[Campos]
Editar as configurações da interface
[Não]
Mínino permitido válido?
[Não]
[Sim]
[Gerar]
Arquivo válido
Gerar interfaces para arquivo XAML
Salvar configurações da interface
[Não]
[Sim]
Selecionar projeto para salvar arquivos
Editar arquivos?
[Sim]
Editar arquivos gerados
[Não]
[Não]
Arquivo válido
[Sim]
Final

Especificação

- Diagrama de Classes:
  - Interface;
  - Campo;
  - ConstrutorInterface;
  - Negocio;
  - ParametrosGerais ;
  - Relacionamento;
  - XMLLeitor.


cd Diagrama de classes
ConstrutorInterface
- pathDestino: string
- pathMaskTextBox: string
- pathInterfaceMaskTextBox: string
ParametrosGerais
+ AtalhoCancelar: string
+ AtalhoExcluir: string
+ AtalhoFecharJanela: string
+ AtalhoInserir: string
+ AtalhoProcurar: string
+ AtalhoSalvar: string
+ EsteticaAltura: string
+ EsteticaLargura: string
+ IconeCancelar: string
+ IconeExcluir: string
+ IconeFecharJanela: string
+ IconeInserir: string
+ IconeProcurar: string
+ IconeSalvar: string
+ MascaraData: string
+ MascaraHora: string
+ MascaraTelefone: string
+ MascaraValor: string
+ MsgCancelar: string
+ MsgExcluir: string
+ MsgFecharForm: string
Negocio
XMLLeitor
- _path: string
- _xmlDoc: XmlDocument
Relacionamento
+ TipoRelacionamento: Tipo
+ SourceName: string
+ TargetName: string
+ MultiplicidadeSource: string
+ MultiplicidadeTarget: string
Interface
+ Height: int
+ Nome: string
- pathDestino: string
- pathTempalteCS: string
- pathTemplateXaml: string
+ Tipo: string
+ Visivel: bool
+ Width: int
- CampoAgregado: string
Campo
- Labels: string
- UniformGrid: string
- Campos: string
- CamposMascaras: string
- CheckBox: string
- Combo: string
- CampoAssociadoNxN: string
+ Nome: string
+ TipoCampoReal: string
+ Tamanho: int
+ TextAlign: HorizontalAlignment
+ Obrigatorio: bool
+ Tipo: TipoCampo
+ Visivel: bool
+ Hint: string
+ Mascara: string
+ ValorMascara: string
1
1..*

Implementação

- Ferramentas:
  - EA para exportação dos diagramas;
  - Microsoft Visual c# 2008 Express Edition como ambiente de desenvolvimento;
  - Linguagem C#;
  - *Framework 3.0.*

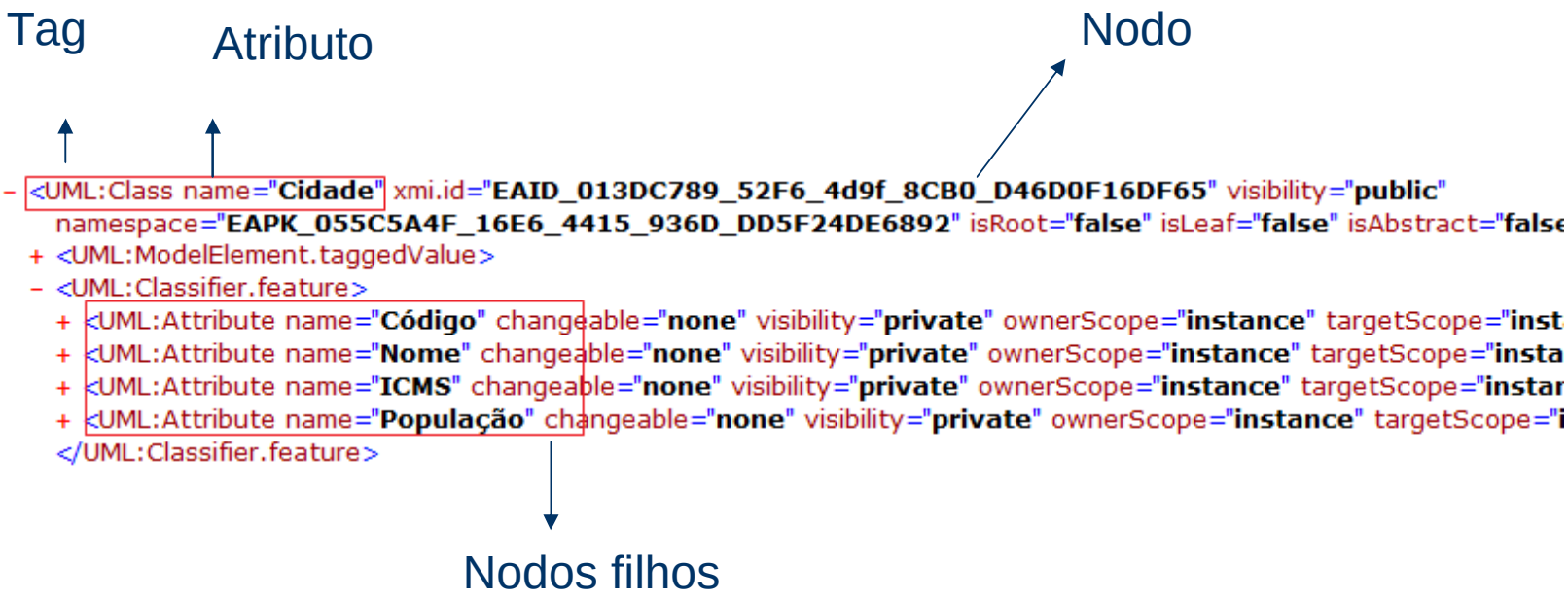
Implementação

- Técnicas:
  - Importação;
  - Definição das usabilidade;
  - Geração das interfaces;

Implementação(Importação)

- As principais bibliotecas e classes são:
  - Biblioteca *System.XML;*
  - XmlNodeList;
  - XmlNode;
  - XmlAttributeCollection;
- Os principais métodos são:
  - LerArquivoXML();
  - RetornarInterfacesPorNomeTag();
  - RetornarRelacionamentosPorNomeTags();
  - DefinirAtributosInterface();
  - DefinirAtributosCampo();
  - DefinirRelacionamentosInterface();
  - DefinirCampos();

Implementação(Importação)



Implementação(Importação)

- DefinirCampos()

```
// rotina que varre em pronfundidade os nodos e subnodos recursivo até achar o nodo com o nome do elemento
// UM:Atributes..esse elemento contem os campos da interface;
private void DefinirCampos(XmlNode iInterface, Interface iObjetoInterface)
{
    // Percorrer todos os elementos que estiverem dentro do elemento filho atual
    for (int b = 0; b < iInterface.ChildNodes.Count; b++)
    {
        // considerar somente os elementos...
        if (iInterface.ChildNodes.Item(b).NodeType == XmlNodeType.Element)
        {
            if (iInterface.ChildNodes.Item(b).Name != "#text")
            {
                if (iInterface.ChildNodes.Item(b).Name == "UML:Attribute")
                {
                    DefinirAtributosCampo(iInterface.ChildNodes.Item(b),
                                          iInterface.ChildNodes.Item(b).Attributes,
                                          iObjetoInterface);
                }

                // rotina é recursiva, então manda le novamente pois poderá haver mais nodos
                // ler somente os elementos que guardam os atributos
                // ler nodo por selecnodos....e selecionar os nodos de atributo
                DefinirCampos(iInterface.ChildNodes.Item(b), iObjetoInterface);
            }
        }
    }

}
```

Implementação(Importação)

- DefinirAtributosCampo()

```
// rotina que varrer o conjunto de atributos que contem neste campo conforme o nome definido na lista
private void DefinirAtributosCampo(XmlNode iNodoPaiAtributos, XmlAttributeCollection iAtributos,
                                   Interface iObjetoInterface)
{
    Campo campo = new Campo();

    XmlNode nodeAux = iAtributos.GetNamedItem("name");
    campo.Nome = RemoverCaracterObrigatorio(nodeAux.Value);
    campo.Visivel = true;
    campo.Hint = RetornarValorAtributo(iNodoPaiAtributos, "description");
    campo.DefinirTipoCampo(RetornarValorAtributo(iNodoPaiAtributos, "type"));
    campo.TipoCampoReal = RetornarValorAtributo(iNodoPaiAtributos, "type");
    campo.DefinirMascara();
    if (campo.Mascara == "Telefone")
        campo.ValorMascara = _parametrosGerais.MascaraTelefone;
    else if (campo.Mascara == "Valor")
        campo.ValorMascara = _parametrosGerais.MascaraValor;
    else if (campo.Mascara == "Data")
        campo.ValorMascara = _parametrosGerais.MascaraData;
    else if (campo.Mascara == "Hora")
        campo.ValorMascara = _parametrosGerais.MascaraHora;
    else
        campo.ValorMascara = "";

    if (campo.Tipo == Campo.TipoCampo.String)
        campo.Tamanho = 200;
    else if ((campo.Tipo == Campo.TipoCampo.Boolean) || (campo.Tipo == Campo.TipoCampo.Indefinido))
        campo.Tamanho = 200;
    else
```

Implementação(Definição das usabilidades)

- As principal classe é:
  - XMLTextWriter;
- Os principais métodos são:
  - GerarArquivoConfiguracaoGeral();
  - CarregarArquivoConfiguracaoGeral();

Implementação(Geração das interfaces)

- Os principais métodos são:
  - GerarInterfaces(string pathDestino);
  - _interface.Gerar() ;
- Sub-Métodos:
  - DefinirNomeInterface();
  - DefinirTamanhoInterface();
  - DefinirTeclasAtalhos();

Implementação(Geração das interfaces)

- DefinirTeclasAtalhos();

```
public void DefinirTeclasAtalho(string iPathXaml)
{
    XmlDocument doc = new XmlDocument();

    doc.Load(iPathXaml);

    XmlNodeList menuItem = doc.GetElementsByTagName("MenuItem");
    foreach (XmlNode node in menuItem)
    {
        XmlNode atributo = node.Attributes.GetNamedItem("Name");

        if (atributo != null)
        {
            if (atributo.Value == "mniInserir")
            {
                XmlNode atalho = node.Attributes.GetNamedItem("InputGestureText");
                atalho.Value = _parametrosGerais.AtalhoInserir;
            }
            else if (atributo.Value == "mniCancelar")
            {
                XmlNode atalho = node.Attributes.GetNamedItem("InputGestureText");
                atalho.Value = _parametrosGerais.AtalhoCancelar;
            }
            else if (atributo.Value == "mniSalvar")
            {
                XmlNode atalho = node.Attributes.GetNamedItem("InputGestureText");
                atalho.Value = _parametrosGerais.AtalhoSalvar;
            }
```

Localizando menuItem

Definindo um valor para o atributo do menu item

Implementação(Geração das interfaces)

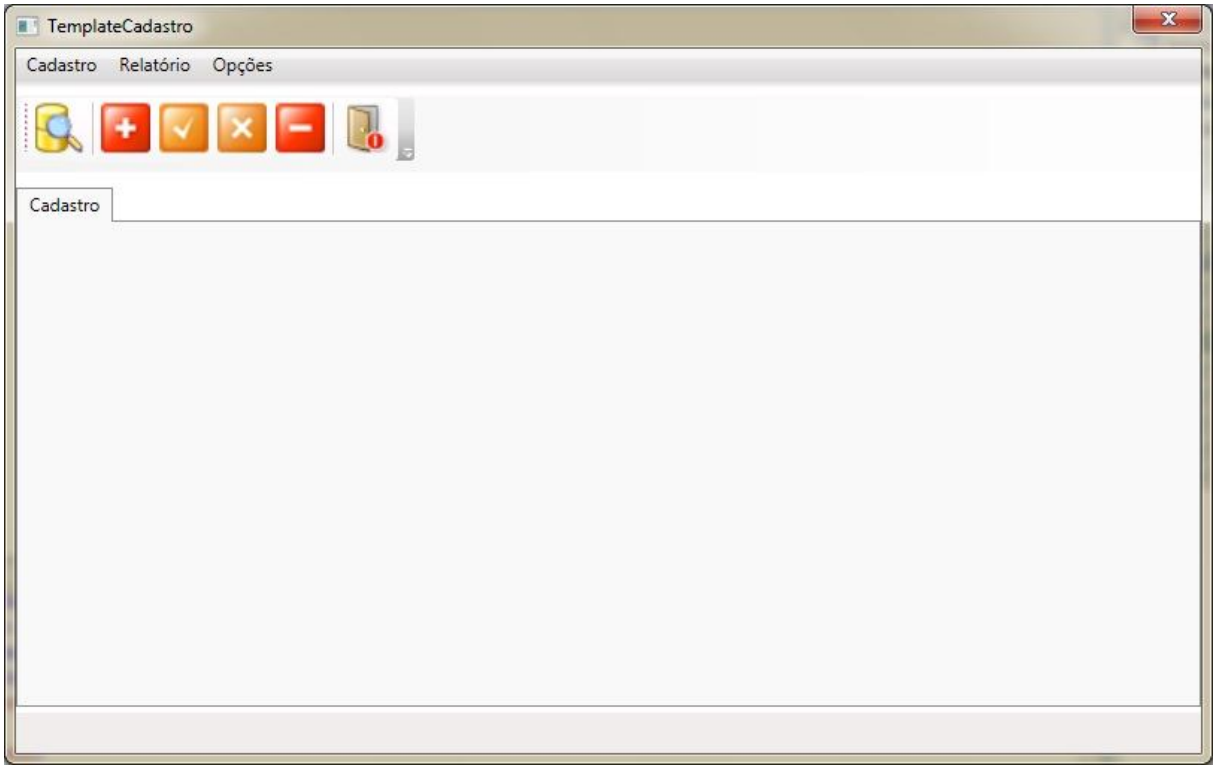
- Código XAML do template de cadastro

```
<Window x:Class="WpfApplication1.TemplateCadastro" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        WindowStartupLocation="CenterScreen" Closing="Cadastro_Closing" Keyboard.KeyDown="Window_KeyDown"
        xmlns:local="clr-namespace:MaskedTextBox">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="26*" />
      <RowDefinition Height="55*" />
      <RowDefinition Height="214*" />
      <RowDefinition Height="141*" />
      <RowDefinition Height="26*" />
    </Grid.RowDefinitions>
    <Menu Name="mnuCadastro">
      <MenuItem Header="Cadastro" Name="mniCadastro" ToolTip="Opções do Cadastro">
        <MenuItem Header="_Procurar" Name="mniProcurar" InputGestureText="F5" Click="mniProcurar_Click" AutomationPrc
        <Separator />
        <MenuItem Header="_Inserir" Name="mniInserir" InputGestureText="F1" Click="mniInserir_Click"/>
        <MenuItem Header="_Salvar" Name="mniSalvar" InputGestureText="F2" Click="mniSalvar_Click"/>
        <MenuItem Header="_Cancelar" Name="mniCancelar" InputGestureText="F3" Click="mniCancelar_Click"/>
        <MenuItem Header="_Excluir" Name="mniExcluir" InputGestureText="F4" Click="mniExcluir_Click"/>
        <Separator />
        <MenuItem Header="_Fechar" Name="mniFechar" InputGestureText="Escape" Click="mniFechar_Click"/>
      </MenuItem>
      <MenuItem Header="Relatório" Name="mniRelatorio" ToolTip="Relatórios referentes ao cadastro" />
      <MenuItem Header="Opções" Name="mniOpcoes" ToolTip="Opções Diversas" />
    </Menu>
    <StatusBar Grid.Row="4" Name="stbCadastro">
    </StatusBar>
    <ToolBarTray Name="tbtBotoesCadastro" AllowDrop="False" OpacityMask="BlanchedAlmond" Margin="0,4.943,0,0" Grid.Rc
      <ToolBar Band="1" BandIndex="1" Height="44" HorizontalAlignment="Stretch" VerticalContentAlignment="Stretch" Na
```

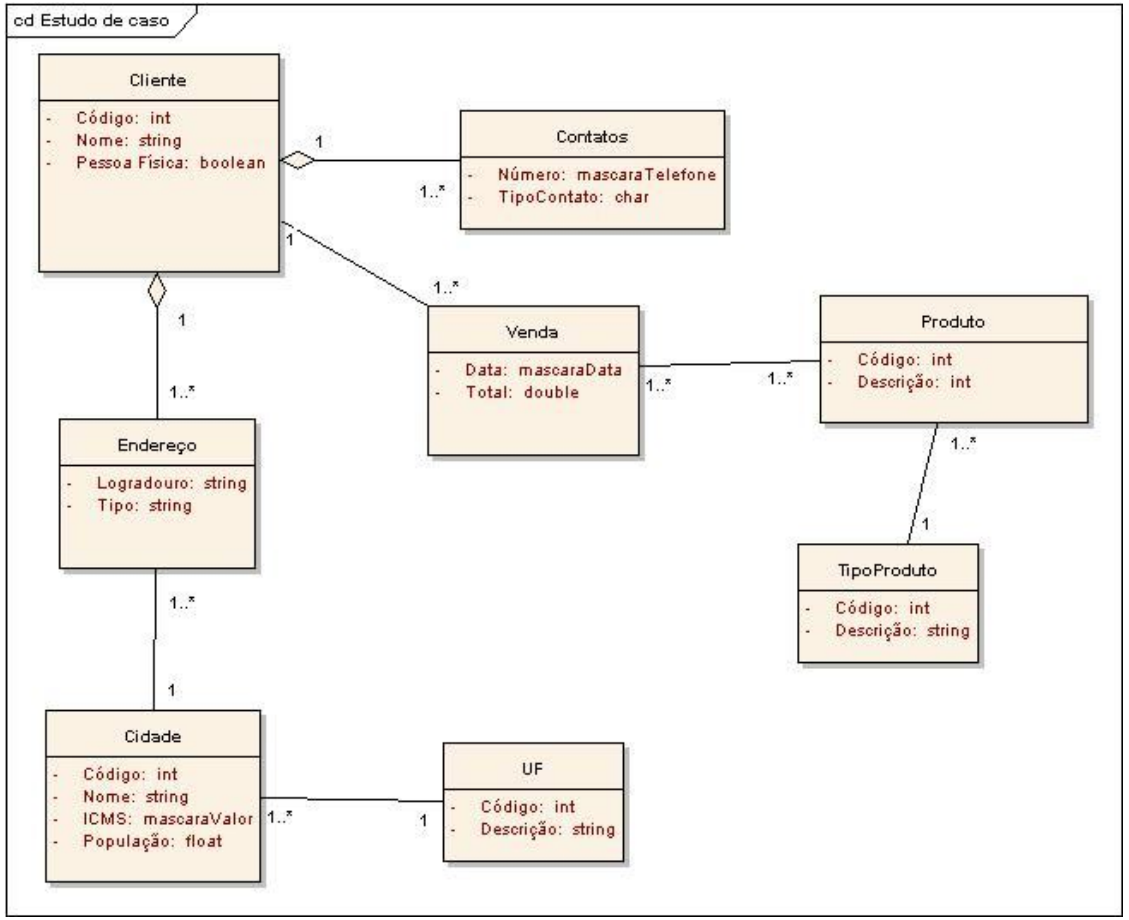
Definição do atalho

Implementação(Geração das interfaces)

- Template de cadastro

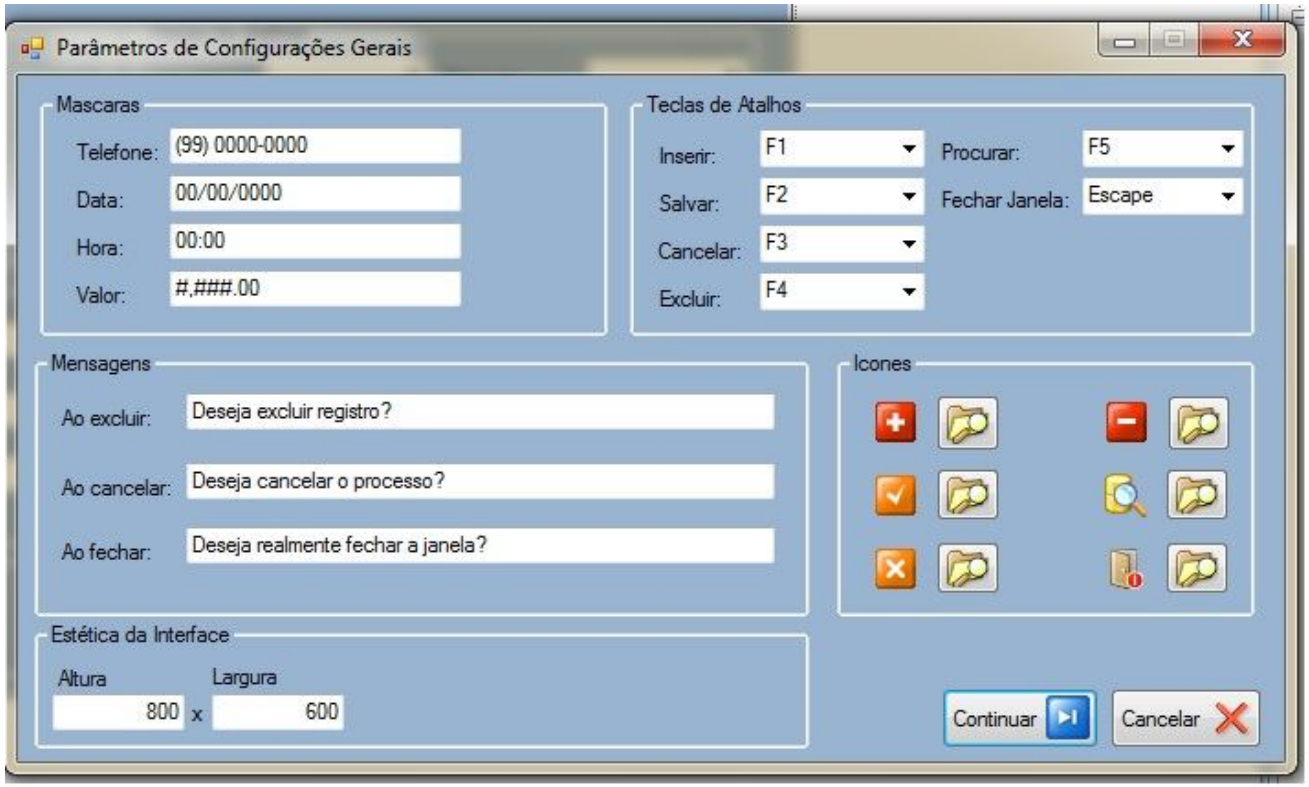


Operacionalidade(Estudo de caso)



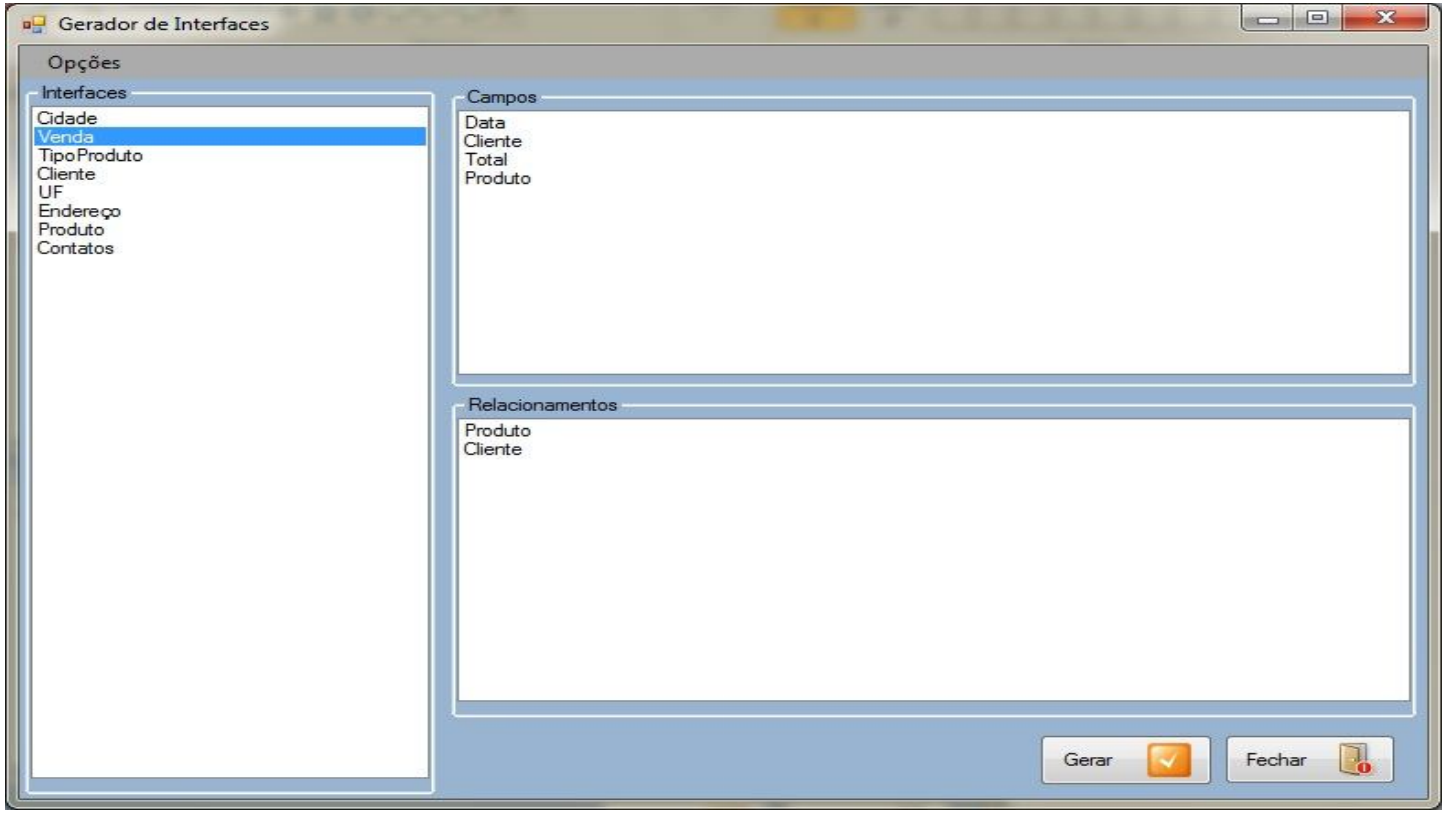
Operacionalidade

- Definir critérios ergonômicos de usabilidade



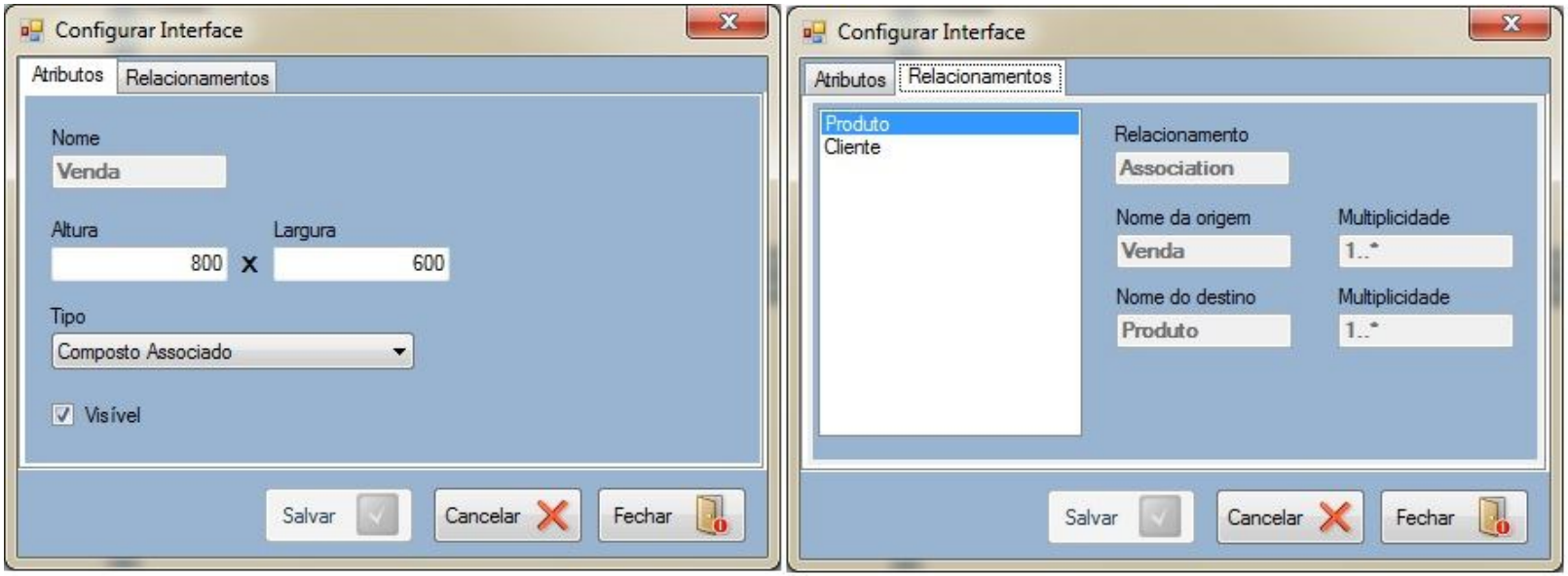
Operacionalidade

- Tela principal do aplicativo



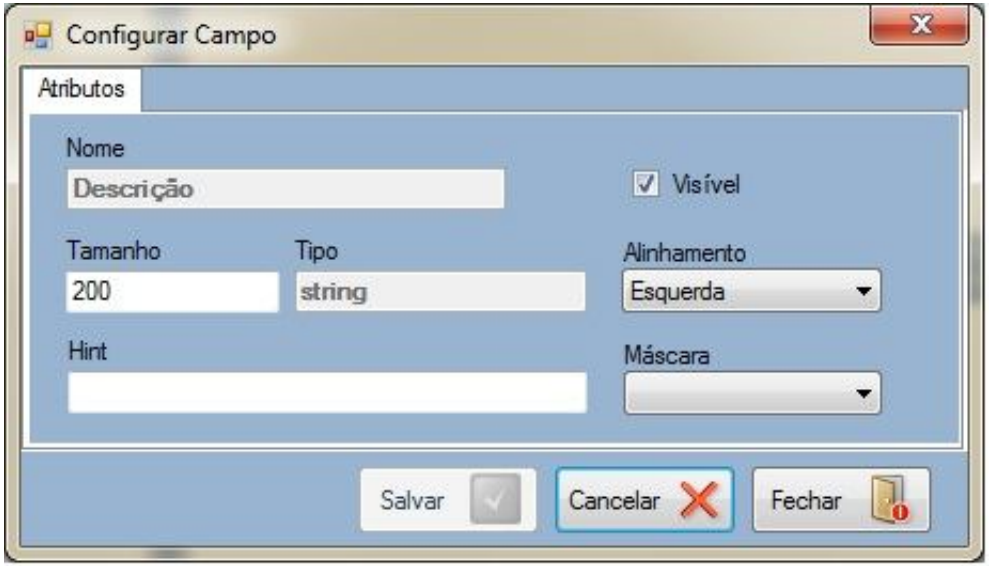
Operacionalidade

- Configurar interface



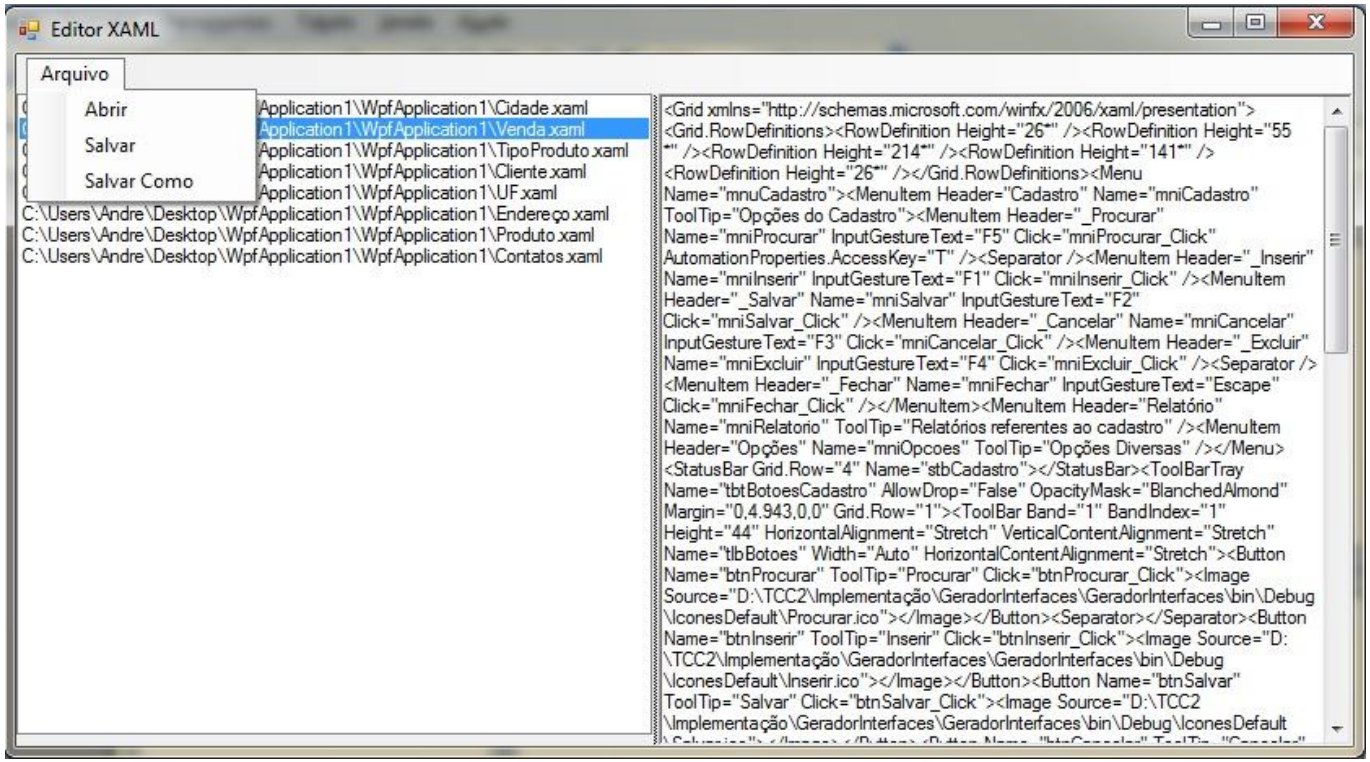
Operacionalidade

- Configurar Campo



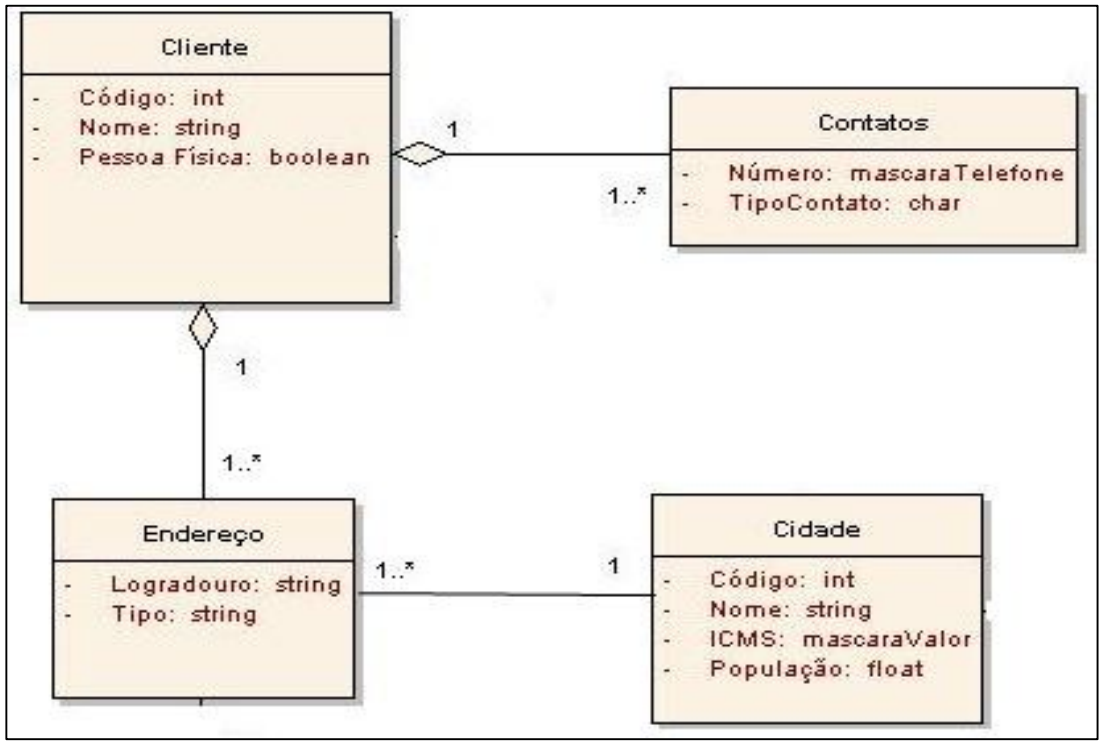
Operacionalidade

- Editor XAML



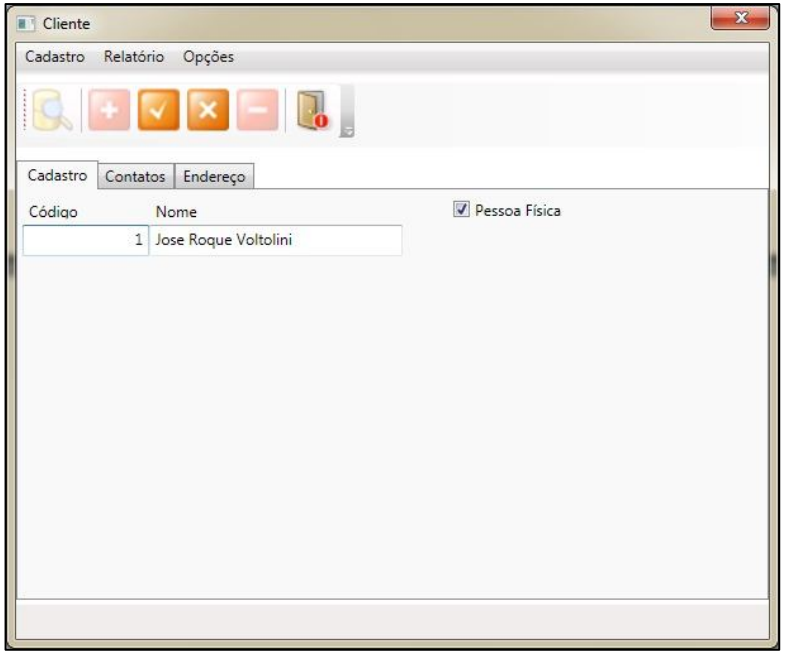
Operacionalidade

- Diagrama de classes (Cliente)

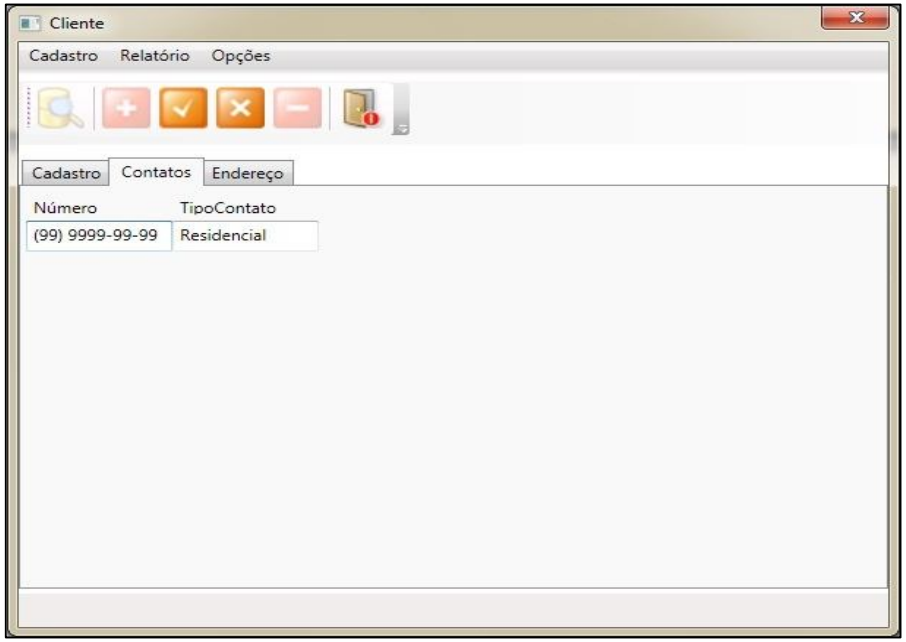


Operacionalidade

- Aba principal

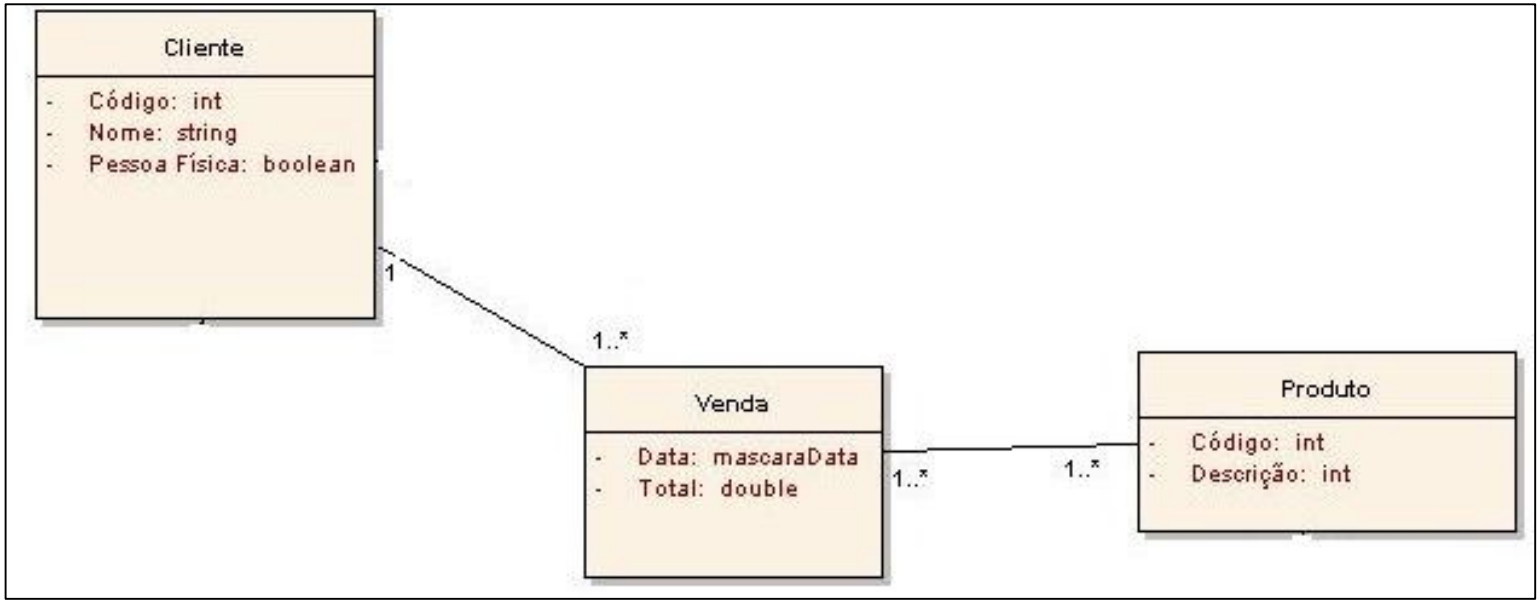


- Aba contatos



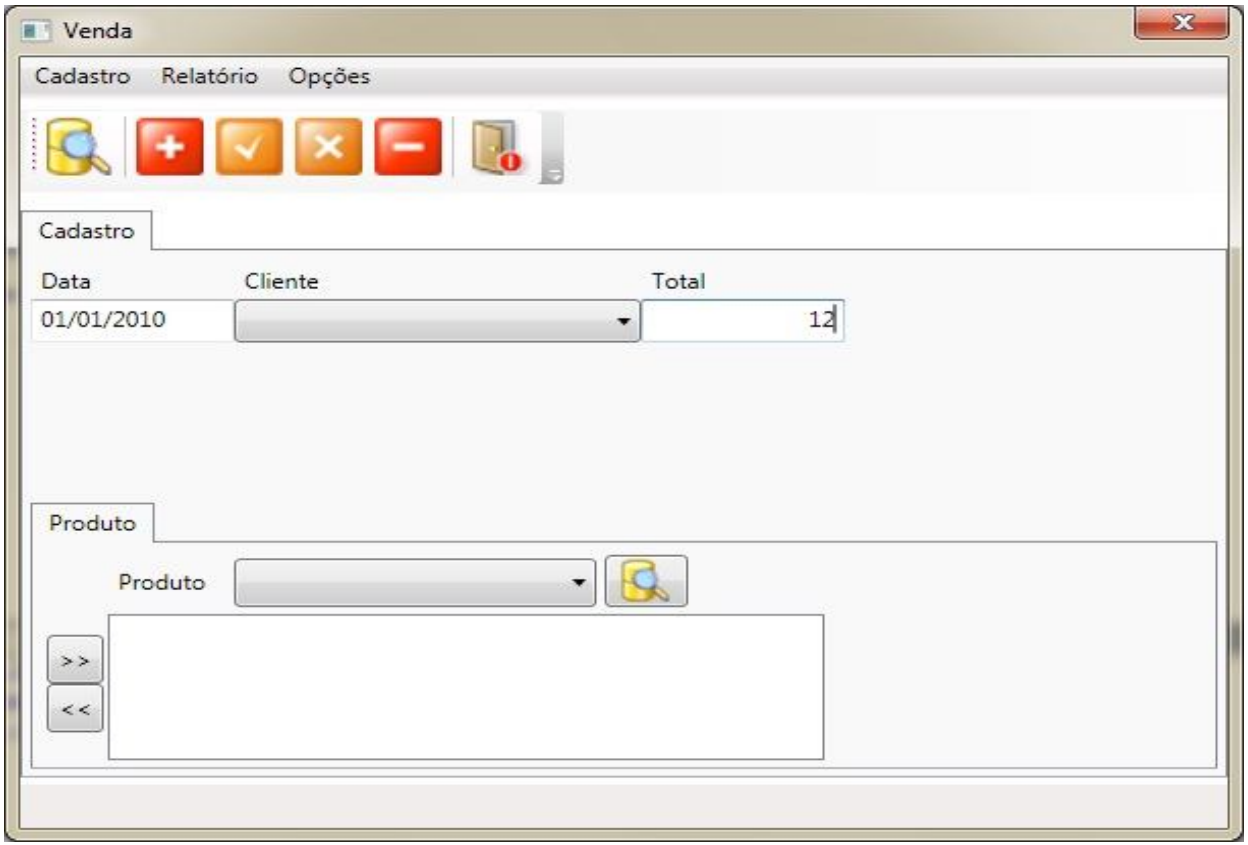
Operacionalidade

- Diagrama de classes (Venda)



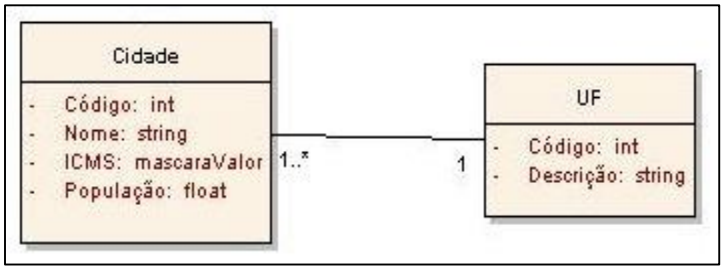
Operacionalidade

- Interface Gerada de “Vendas”

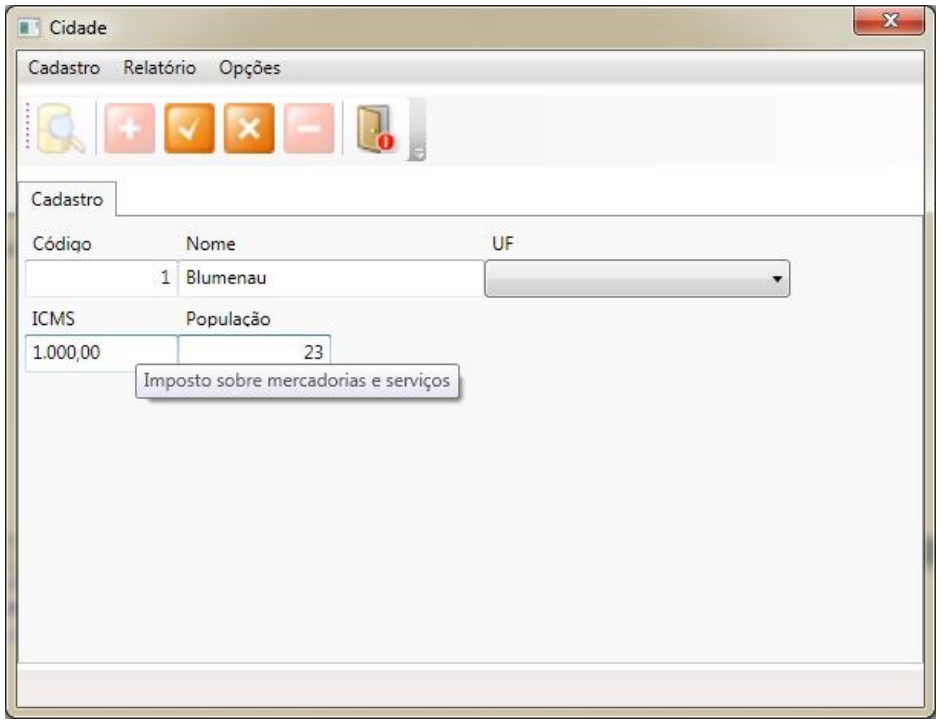


Operacionalidade

- Diagrama de classes (Cidade)



- Interface Gerada de “Cidade”



Operacionalidade(Considerações)

- Classes sem relacionamentos geram somente interfaces com uma aba principal, denominada cadastro;
- Classes com relacionamentos do tipo associação n x n, geram interfaces com uma aba principal denominada cadastro e dentro desta aba, é criado para cada relacionamento uma nova aba como o nome da classe relacionada;
- Classes com relacionamentos do tipo agregação 1 x n, geram interfaces com uma aba denominada cadastro e para cada tipo de relacionamento de agregação, cria-se uma aba, sendo estas uma ao lado da outra.

Operacionalidade(Considerações)

- Atributos do tipo *class*, exemplo UF:UF, são automaticamente trocado pelo sistema para, “Associado1x1”, “Associado1xN”, “AssociadoNxN” ou “Agregado1xN”;

- Os atributos do tipo “Associado1x1” e “Associado1xN” geram campos do tipo *combobox* dentro da aba pertencente a sua origem;

- Os atributos do tipo “AssociadoNxN” geram dentro da aba de sua origem um campo do tipo *combobox*, um *listbox* para adicionar os valores, um botão de adicionar, um botão para remover e um botão de procurar ;

- Os atributos do tipo “Agregado1xN”, busca todos os atributos que estão no relacionamento de destino e para cada atributo geram um campo conforme o seu tipo, dentro da sua aba de origem.

Operacionalidade(Considerações)

- Atributos do tipo int, float e double geram campos *textbox* alinhados a direita;
- Atributos do tipo string e char geram campos *texbox* alinhados a esquerda;
- Atributos do tipo mascaraValor, mascaraData, mascaraHora e mascaraTelefone geram campos do tipo “MyMaskedTextBox” com o valor da mascara definida pelo usuário no aplicativo.

Resultados e discussão

- O aplicativo atingiu a meta esperada. O estudo de caso comprova a quantidade de relacionamentos e campos possíveis gerados;
- Fontes de pesquisa sobre o assunto WPF e XAML foram limitadas;
- Houve necessidade de criar uma cama de negócio para cada camada visual.

Resultados e discussão

- As interface são geradas similar ao utilizado em Pires e Siqueira(2004):

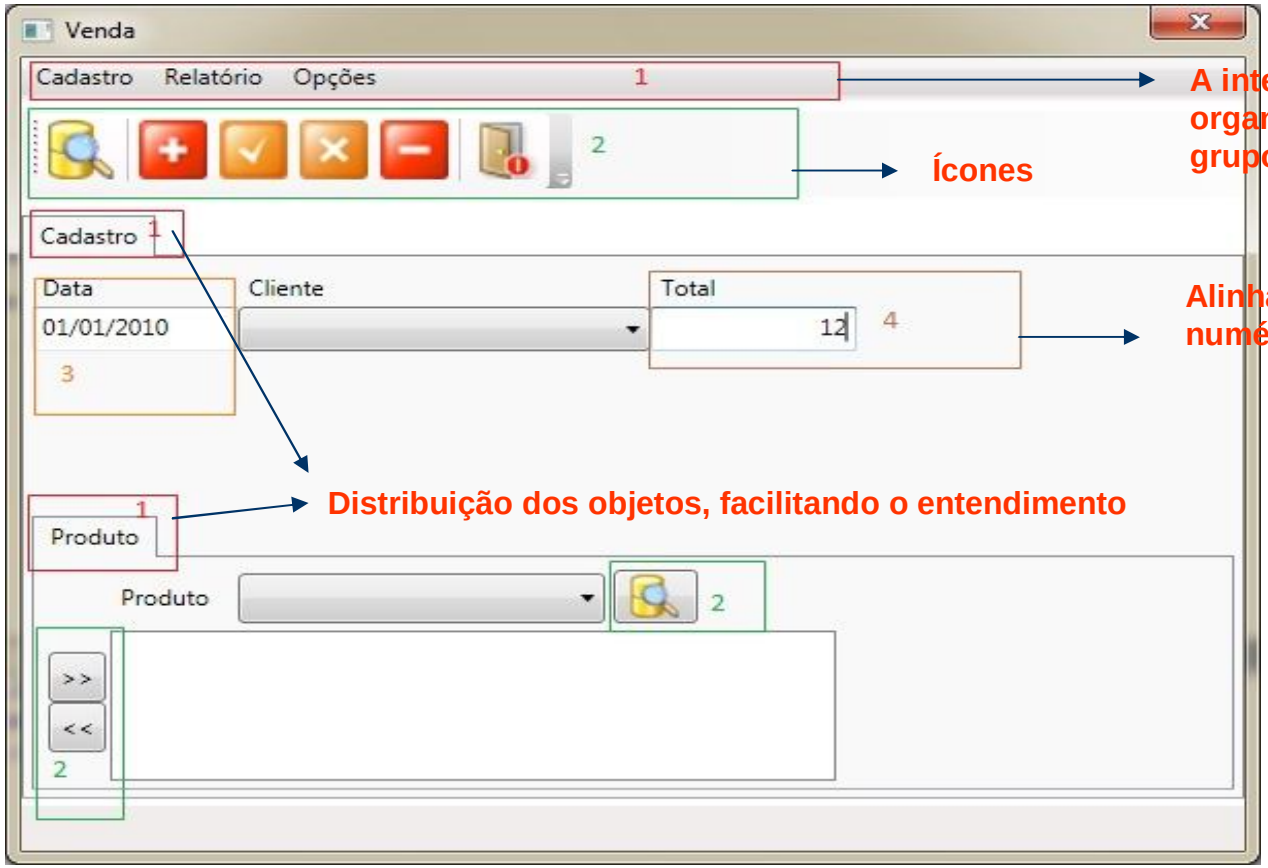
	Pires e Siqueira	Aplicativo Atual
Persistência	Sim	Não
Dependências	Sim	Não
Usabilidade	Não	Sim
Tecnologia WPF	Não	Sim

Resultados e discussão

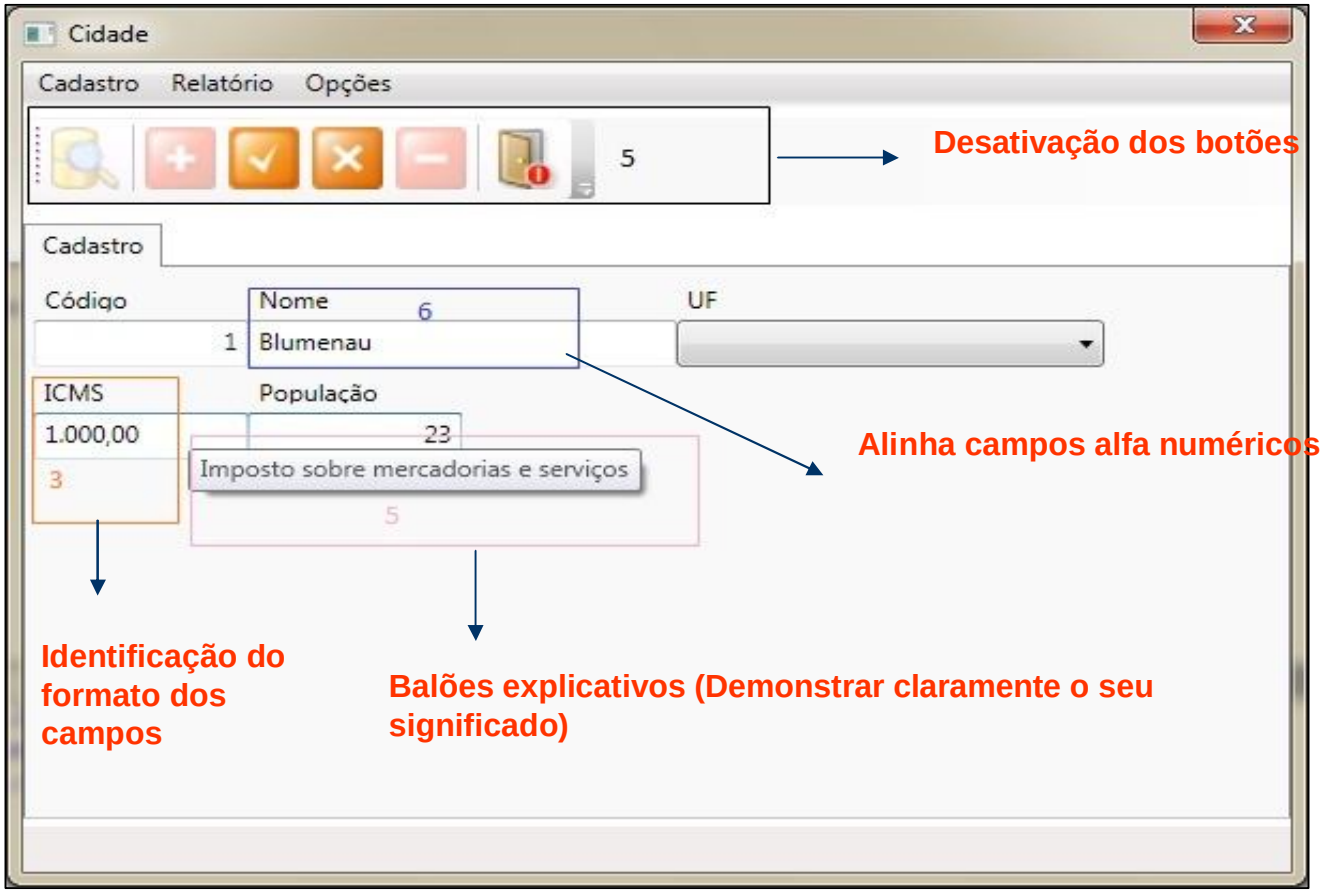
- Foi possível utilizar os seguintes critérios:

a) Usabilidade – inteligibilidade;
   - Ex: Usos de identificadores, informação através de balões
b) Aspectos visuais;
   - Ex: Alinha campos numéricos a direita e alfa a esquerda
c) Mensagem apresentadas;
   - Ex: Exibe mensagem de orientação ao usuário
d) Usabilidade – operacionalidade;
   - Ex: teclas de atalho
e) Prevenção de erros;
   - Ex: Impossibilita entrada de dados numéricos em alfa

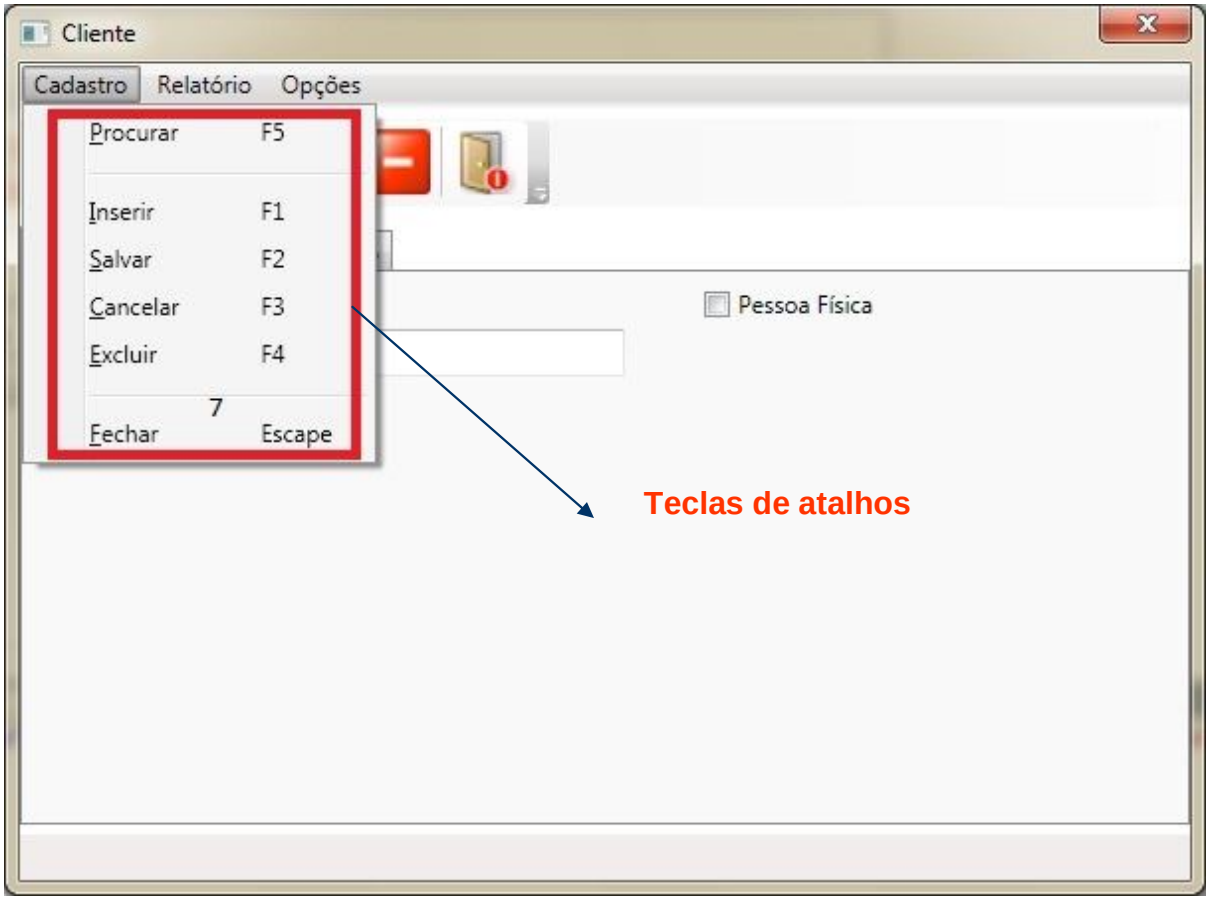
Resultados e discussão



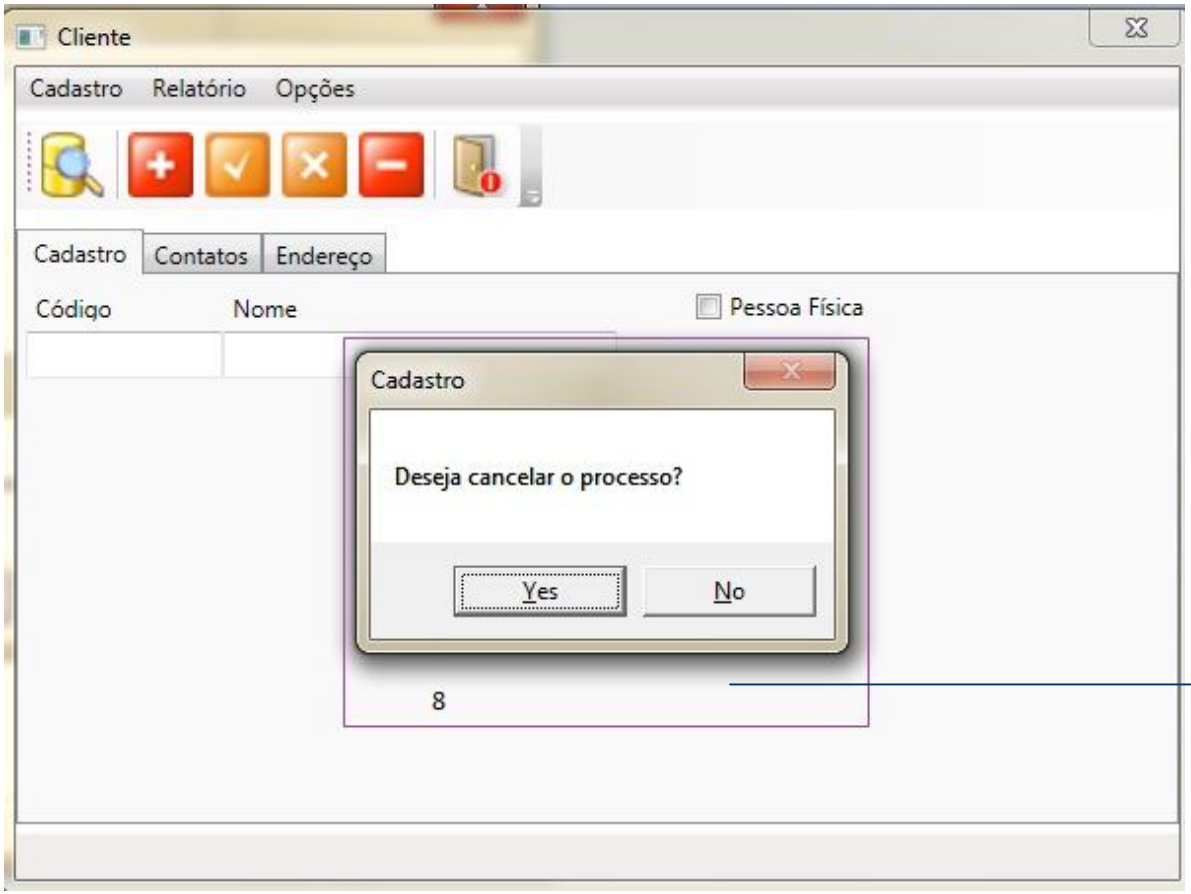
Resultados e discussão



Resultados e discussão



Resultados e discussão



Exibe mensagem de orientação ao usuário

Conclusão

- Os resultados foram alcançados em relação aos objetivos previamento formulados;
- Foi implementado um aplicativo que auxilia na semi-automatização nas construções de interfaces para nova tecnologia WPF;
- WPF tem seu futuro garantido;
- Automatização tira o trabalho “braçal” do desenvolvedor;
- Verificou-se a importância dos critérios ergonômicos de usabilidades;
- Critérios ergonômicos auxiliam o usuário a realização de tarefas.

Limitações

- As principais limitações são:
  - Não são tratados mais de um diagrama a serem exportado no mesmo projeto definido pela ferramenta EA e fora da pasta do projeto raiz;
  - Não são reconhecidos relacionamentos do tipo generalização, composição e especialização;
  - Não são reconhecidos campos do tipo *long* e *short*.

Extensões

- Sugere-se que as interfaces não sejam somente de cadastros;
- Gerar camada de persistência para gravação e edição dos dados;
- Gerar interfaces para aplicativos web.