**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO**

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

**ANDRÉ LUIS BECKER**

**BLUMENAU**
**2009**

**2009/2-01**

**ANDRÉ LUIS BECKER**

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

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Everaldo Arthur Grahl, Mestre – Orientador

**BLUMENAU**
**2009**

**2009/2-01**

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

Por

**ANDRÉ LUIS BECKER**

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

| | |
|---|---|
| Presidente: | Prof. Everaldo Artur Grahl, Mestre – Orientador, FURB |
| Membro: | Prof. Adilson Vahldick, Mestre – FURB |
| Membro: | Prof. Jacques Robert Heckmann, Mestre – FURB |

Blumenau, 16 de dezembro de 2009

Dedico este trabalho aos meus pais, irmãos, amigos e namorada que foram a fonte de minha inspiração e motivação para o seu desenvolvimento.

## AGRADECIMENTOS

Aos meus pais, que sempre me deram apoio.

À minha família, que me ajudou nas horas precisas.

À minha namorada e família, que me acolheram com carinho e pelos incentivos.

Aos professores da FURB, que ajudaram a me tornar um profissional.

Ao meu coordenador, pela orientação e incentivo do trabalho final.

Ao meu chefe, pela ajuda no desenvolvimento do aplicativo.

# RESUMO

Este trabalho apresenta um aplicativo para geração de interfaces *standalone*, conforme critérios ergonômicos de usabilidade, a partir de diagrama de classes de domínio, modelados e exportados pela ferramenta Enterprise Architect (EA). O modelo de diagrama de classes é exportado para um arquivo eX*tensible Markup Language Metadata Interchange* (XMI) que auxilia na obtenção e na troca de informações. Através do aplicativo é possível importar as informações contidas no arquivo XMI para sua visualização e edição, configurar alguns critérios de usabilidade e gerar as interfaces. As interfaces geradas destinam-se a nova tecnologia da Microsoft, o *Windows Presentation Foundation* (WPF). O aplicativo desenvolvido utiliza-se dá linguagem de programação *C-Sharp* (C#) e tem como objetivo auxiliar nas construções de interfaces.

Palavras-chave: Interfaces. Usabilidade. Diagrama de classes.

# ABSTRACT

This work presents an application to generate standalone interfaces, according to usability ergonomic criteria, from a domain class diagram, modeled and exported from Enterprise Architect (EA). The class diagram model is exported to a eXtensible Markup Language Metadata Interchange (XMI) file which helps in obtaining and exchanging information. Through the application it`s possible to import the information contained in the XMI file for viewing and editing them, set some criteria for usability and generate the interfaces. These interfaces are designed to work with the new technology from Microsoft, Windows Presentation Foundation (WPF). The application was developed using the C-Sharp (C #) programming language and aims to assist interfaces construction.



Key-words: Interfaces. Usability. Class diagram.

# LISTA DE ILUSTRAÇÕES

# LISTA DE SIGLAS

CASE – *Computer-Aided Software Engineering*

C# - C-Sharp

DLL - *Dynamic-Link Library*

DTD - *Document Type Definition*

EA – Enterprise Architect

HTML - *HyperText Markup Language*

MOF - *Managed Object Format*

UML – *Unified Modeling Language*

OMG - *Object Management Group*

XAML - *eXtensible Application Markup Language*

XMI - *eXtensible markup language Metadata Interchange*

WPF – *Windows Presentation Foundation*

# SUMÁRIO

# 1 INTRODUÇÃO

Atualmente vive-se uma revolução nas áreas relacionadas ao desenvolvimento de software. Esta revolução está acontecendo em virtude do amadurecimento das pesquisas e das técnicas relacionadas à Engenharia de Software. Há muito tempo, a única preocupação das empresas era concluir o software dentro dos prazos estabelecidos (dentro dos limites aceitáveis). Contudo, muitas empresas estão utilizando diversas técnicas oferecidas pela Engenharia de Software para auxiliar na concepção e, sobretudo, na construção de softwares com índices maiores de qualidade e produtividade.

Juntamente com estas evoluções e técnicas criou-se a *Unified Modeling Language* (UML) que é uma notação padronizada para a especificação e modelagem de softwares orientados a objeto amplamente aceita pelo mercado. Fundamentalmente, a UML oferece um conjunto de diagramas cujo objetivo é representar graficamente os diversos elementos de um software. Um deles em particular é o diagrama de classes, um dos mais importantes diagramas estruturais da UML (CAETANO, 2003).

Tem-se a disposição várias ferramentas *Computer–Aided Software Engineering* (CASE) para modelagem de software e uma delas é o Enterprise Architect (EA). Esta ferramenta permite a construção de modelos usando diagramas e notação UML. Dentre estes diagramas tem-se o diagrama de classes. Levando em consideração que a modelagem de software geralmente é um processo trabalhoso, a ferramenta CASE EA dá a possibilidade de automatizar alguns processos tais como permitir geração e engenharia reversa de classes escritas em diversas linguagens, geração de código *Dynamic-Link Library* (DLL). No que diz a respeito à automação da geração de interfaces, a ferramenta EA não disponibiliza esta opção. A construção de interface no decorrer de um projeto torna-se um processo trabalhoso e cauteloso, envolvendo aspectos de usabilidade como critérios ergonômicos, pois é através dela que usuário interage com as demais funcionalidades que um sistema qualquer pode nos propor.

Tentando automatizar parte deste processo, surgiu a idéia de desenvolver uma ferramenta para construção de interfaces para aplicações *standalone* a partir de um diagrama de classes, conforme critérios ergonômicos de usabilidade. As interfaces são geradas para a nova tecnologia da Microsoft, o *Windows Presentation Foundation* (WPF), que é um conjunto de classes (bibliotecas), mais flexíveis e independentes do código fonte, as quais acrescentam novos recursos como 3D, animações, entre outras. Um dos pilares do WPF é o

*eXtensible Application Markup Language* (XAML), com o qual pode-se criar toda a interface gráfica e grande parte do seu comportamento. A idéia central da ferramenta é permitir configurar valores para atender critérios ergonômicos de usabilidade e automatizar o processo de geração de interfaces.

## 1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta para construção de interfaces conforme critérios ergonômicos de usabilidade a partir de diagrama de classes.

Os objetivos específicos do trabalho são:

a) obter informações do diagrama de classes gerado pela ferramenta CASE EA;
b) traduzir as informações obtidas para linguagem XAML (código fonte da interface);
c) filtrar o código gerado e adaptá-lo para gerar interfaces gráficas para aplicações *standalone* conforme critérios ergonômicos de usabilidade;
d) permitir configurar valores para atender os critérios ergonômicos de usabilidade.

## 1.2 ESTRUTURA DO TRABALHO

Na seção 2.1 aborda-se os problemas de usabilidade. Em seguida, na seção 2.2 são evidenciados os principais critérios ergonômicos de usabilidade. Na seção 2.3 apresenta-se a tecnologia WPF e XAML. Na seção 2.4 relata-se sobre o diagrama de classes. Na seção 2.5 descreve-se sobre o Enterprise Architect. Na seção 2.6 fala-se sobre XMI, na seção 2.7 mostra-se o Model Driven Architecture (MDA) e por fim, na seção 2.8, são apresentados alguns trabalhos correlatos.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo apresenta-se os principais problemas de usabilidade e alguns critérios ergonômicos. Também relata-se a tecnologia WPF, os conceitos de diagrama de classes e uma breve apresentação da ferramenta EA. Fala-se sobre o formato padrão XMI, a tecnologia MDA e por fim, os trabalhos correlatos.

## 2.1 PROBLEMAS DE USABILIDADE

Os problemas de usabilidade relacionam-se com as coisas que impedem o usuário de interagir com o sistema. Quando a interface não é capaz de fornecer ao usuário métodos que podem facilitar o aprendizado para manusear o sistema, então ocorrem estes problemas.

Segundo Nielsen e Molich (1990), os problemas de usabilidade encontrados na interface do sistema impedem que o usuário consiga chegar ao seu objetivo. Os principais problemas são:

a) a aparência da interface do sistema pode afetar a execução das ações;
b) muitas funcionalidades podem confundir o usuário, fazendo com que haja redução de uso do sistema. O usuário também pode executar, com dificuldades, apenas algumas ações do sistema;
c) o problema no item b pode causar frustração do usuário porque o mesmo não completa a execução das ações e desiste da tarefa;
d) quando os nomes das funcionalidades do sistema não são apresentados no mesmo idioma do usuário, podem dificultar o aprendizado assim como influenciar na desistência do uso do sistema;
e) quando o usuário tenta executar as ações, erra, mas não consegue voltar e corrigir os erros porque não possui mensagens informativas. Estas mensagens devem ajudar o usuário a resolver os problemas;
f) quando as mensagens informativas não fornecerem clareza podem causar conseqüências como influenciar na desistência do uso do sistema.

## 2.2 CRITÉRIOS ERGONÔMICOS

A Ergonomia de Software tem o papel de adaptar a interface do sistema de acordo com as necessidades dos usuários, para que a interação seja eficiente e apresente conforto e satisfação na realização das tarefas. Para que a interface seja adequada aos usuários finais, a Ergonomia de Software apresenta critérios ergonômicos que visam avaliar a usabilidade da interface dos sistemas com a finalidade de identificar problemas de interação e apresentar soluções ergonômicas para melhorar e ajustar seu projeto. Dois pesquisadores de língua francesa, Dominique Scapin e Christian Bastien, ligados ao Instituto Nacional de Pesquisa em Automação e Informática da França (INRIA), adotaram um sistema de qualidade conhecido como Critérios Ergonômicos em 1993, um conjunto de oito critérios ergonômicos elementares. Conforme Cybis, Betiol e Faust (2007, p. 25), o objetivo de tal sistema é o de minimizar a ambigüidade na identificação e classificação das qualidades e problemas ergonômicos do software interativo. A lista completa dos principais critérios elementares, conforme Bastien e Scapin (1993) é a seguinte:

a) condução: este critério visa favorecer o aprendizado e a utilização do sistema por usuário novato. Refere-se aos meios para aconselhar, orientar, informar e conduzir o usuário na interação com o sistema;
b) carga de trabalho: este critério se aplica a um contexto de trabalho intenso e repetitivo, no qual os profissionais que operam o sistema precisarão de interfaces econômicas sob o ponto de vista cognitivo e motor. Carga de trabalho diz respeito a todos os elementos da interface que têm um papel importante na redução da carga cognitiva e no aumento da eficiência do diálogo;
c) controle explícito: este critério aplica-se em particular às tarefas longas seqüenciais e nas quais os processamentos sejam demorados. São situações delicadas, nas quais a falta de controle do usuário sobre as ações do sistema pode implicar perda de tempo e de dados;
d) adaptabilidade: este critério é uma qualidade particularmente esperada em sistemas em que o público-alvo é vasto e variado. Nestes casos, fica evidente que a única interface não pode atender plenamente a todos os tipos de usuários. Para que todos tenham direito ao mesmo nível de usabilidade, a interface deve propor maneiras variadas de realizar uma tarefa, deixando ao usuário a liberdade de escolher e dominar uma delas no curso de seu aprendizado;

e) gestão de erros: este critério aplica-se em todas as situações, em particular quando as ações dos usuários forem sujeitas a erros de grande responsabilidade, envolvendo a perda de dados, dinheiro ou colocando em risco a saúde de pessoas. A gestão de erros diz respeito a todos os mecanismos que permitem evitar ou reduzir a ocorrência de erros e que favoreçam a sua correção;
f) homogeneidade/coerência: este critério aplica-se de forma geral, mas em particular quando os usuários são novatos ou intermitentes. Refere-se à forma na qual as escolhas no projeto da interface são conservadas idênticas em contextos idênticos e diferentes para contextos diferentes;
g) significado dos códigos e denominações: este critério refere-se à adequação entre o objeto ou a informação apresentada ou pedida e sua referência na interface. Códigos e denominações não-significativos para os usuários podem levá-los a cometer erros como escolher a opção errada ou deixar de informar um dado importante;
h) compatibilidade: este critério diz a respeito ao grau de similaridade entre diferentes sistemas que são executados em um mesmo ambiente operacional (Windows, Mac, OpenLook). Trata-se de um tipo de consistência externa entre aplicativos de um mesmo ambiente.

## 2.3 TECNOLOGIA WINDOWS PRESENTATION FOUNDATION (WPF)

O WPF faz parte do *.NET Framework* 3.0 e é constituído por um conjunto de classes. A função do WPF é substituir o *Windows Forms* na criação de aplicativos *standalone*, de forma que não se devem esperar grandes *upgrades* na tecnologia *Windows Forms* no futuro, já que o foco da Microsoft vai estar no WPF (BASSI, 2007).

O WPF surgiu em 2001, com a nova tecnologia de apresentação do *Windows Vista*. Suas principais características são flexibilidade da interface, reconhecimento de voz, *layouts* avançados, 3D, animações, gráficos vetoriais, entre outras. Permite a separação entre o *design* e o código, ou seja, a interface pode ser criada por um *designer* e o código por um programador especializado, de maneira independente.

Um programa que usa WPF é composto por duas partes: um arquivo XAML e um código fonte para *.NET* (pode ser escrito em qualquer linguagem compatível como VB.NET,

C#, etc). O arquivo XAML contém as diretrizes da interface gráfica que leva para as aplicações *standalone* o conceito já existente na *Web* da separação entre o *design* e o código fonte.

Conforme Farias (2005), XAML é a nova linguagem de marcação usada para criar interfaces de forma simples e rápida. É equivalente, porém muito mais poderosa que sua antecessora, o *HyperText Markup Language* (HTML). Na figura 1 tem-se uma janela de *login* gerada pelo código XAML da figura 2.

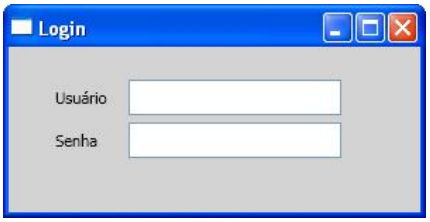



Figura 1 – Janela de *login*

```
<Window x:Class="WindowsApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Login" Height="150" Width="300">
    <Grid Name="myGrid">
        <Canvas Background="LightGray">
            <TextBlock Canvas.Left="33" Canvas.Top="29" Width="68" Height="14"> Usuário</TextBlock >
            <TextBox Canvas.Left="85" Canvas.Top="23" Width="151" Height="25"/>
            <TextBlock Canvas.Left="33" Canvas.Top="59" Width="95" Height="14">Senha</TextBlock >
            <PasswordBox Canvas.Left="85" Canvas.Top="53" Width="151" Height="25"/>
        </Canvas >

    </Grid>

</Window>
```

Figura 2 – Código XAML da janela de *login*

## 2.4 DIAGRAMA DE CLASSES

O diagrama de classes é utilizado na construção do modelo de classes desde o nível de análise até o nível de especificação. De todos os diagramas da UML, esse é o mais rico em termos de notação (BEZERRA, 2004, p. 97).

As classes são representadas ilustrativamente por uma caixa dividida em três partes, sendo a primeira o nome da classe, a segunda os atributos e por último as operações. Os atributos correspondem às informações que um objeto armazena e as operações são as ações que o mesmo realiza.

Há três níveis sucessivos de abstração pelos quais o modelo de classes passa. Conforme Bezerra (2004, p. 96), os níveis são os de modelo de classes de domínio ou negócio, modelo de classes de especificação e o modelo de classes de implementação. Modelos de classes de domínio, por definição, não levam em consideração a tecnologia a ser utilizada na solução do problema.

## 2.5 ENTERPRISE ARCHITECT

Conforme Lima (2005, p. 41), Enterprise Architect é uma ferramenta de análise, projeto e desenvolvimento de aplicações usando UML. Fornece o tipo de visualização robusta e eficiente, além de permitir geração e engenharia reversa de classes escritas em C++, Java, C#, VB, Delphi e PHP.

O EA cobre todos os aspectos do ciclo de desenvolvimento, fornecendo suporte para teste, manutenção e controle de mudanças de requisitos. De todas as ferramentas UML, é a que permite produzir documentação do modo mais fácil e com os melhores resultados (LIMA, 2005).

Dentre as diversas funcionalidades da ferramenta, tem-se a opção de importar e exportar arquivos no formato XMI. A figura 3 mostra a definição de duas classes e o relacionamento entre elas, cada qual com seus respectivos atributos. Na figura 4 tem-se as informações parciais destas classes que se encontra dentro do arquivo XMI.

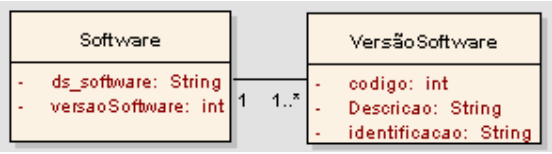



Figura 3 – Diagrama de classe


Figura 4 – Arquivo XMI

Pode-se observar na figura 4, que no arquivo gerado, encontra-se o nome da classe, os atributos e o relacionamento entre elas, delimitado por uma *tag*, representado pelo símbolo “< >”.

Algumas ferramentas apresentam papel semelhante ao EA como a Fast Case, desenvolvida pela URFJ, o Voodoo que suporta a modelagem de diagrama de classes, a ArgoCASEGEO que suporta geração de código e engenharia reversa e por fim, Poseidon que suporta a maioria dos diagramas da UML.

A ferramenta ArgoCASEGEO permite a modelagem de banco de dados geográficos com base no modelo conceitual UML-GeoFrame, que é específico para aplicações de Sistemas de Informação Geográfica (SIG). O dicionário de dados associado ao esquema modelado é armazenado como um documento XMI, visando sua utilização por outros programas.

## 2.6 XML METADATA INTERCHAGE XMI

Conforme Carlson (2001 apud MOSCARDINI, 2003, p. 5), XMI é um formato padrão recomendado pela *Object Management Group* (OMG) desde 1999, que tem como objetivo o intercâmbio de dados possibilitando o compartilhamento de modelos entre ferramentas de diferentes modelagem.

O XMI define um conjunto de regras para geração de *Document Type Definition* (DTD) e mais recentemente esquemas (XML *Schema*), a partir de modelos de classes. XMI é um sistema aberto a qualquer um que queira usufruir de sua capacidade de troca de informações.

O padrão XMI foi projetado para permitir a troca de qualquer modelo de metadados especificado segundo o modelo *Managed Object Format* (MOF) e é composto de dois componentes principais: regras de produção de DTDs XML, que expressam como produzir

DTDs para metadados codificados em XMI; e regras de produção de documento XML, que expressam como codificar metadados em documentos XML válidos e bem formados (MOSCARDINI, 2003).

## 2.7 MODEL DRIVEN ARCHITECTURE (MDA)

MDA é uma metodologia de desenvolvimento de software criada pela OMG. MDA propõe-se separar a especificação das funcionalidades de um sistema da especificação de sua implementação e é divido em três etapas. A primeira etapa é a construção de um modelo com um alto nível de abstração, independente de qualquer tecnologia. Esse modelo é chamado de Modelo Independente de Plataforma (PIM). A segunda etapa, considerada a mais complexa, é a transformação do PIM em um ou mais Modelos Específico de Plataforma (PSM). A terceira etapa é transformar um PSM em código.

Segundo a OMG (2003), devido as suas características, a MDA oferece os seguintes benefícios:

a) produtividade: a transformação do PIM para o PSM precisa ser definida um única vez e pode ser aplicada no desenvolvimento de diversos sistemas. Devido a este fato, tem-se uma redução no tempo de desenvolvimento;
b) portabilidade: dentro da MDA a portabilidade é alcançada é alcançada através do foco dado no desenvolvimento do PIM, que é independente de plataforma. Um mesmo PIM pode ser automaticamente transformado em vários PSMs de diferentes plataformas;
c) interoperabilidade: diferentes PSMs gerados a partir de um mesmo PIM podem conter ligações entre eles, denominada em MDA de pontes. Quando PSMs são gerados em diferentes plataformas, eles não podem se relacionar entre si. É necessário então transformas os conceitos de uma plataforma para outra plataforma.

A figura 5 representa os passos da arquitetura MDA, com seus artefatos e relacionamentos.

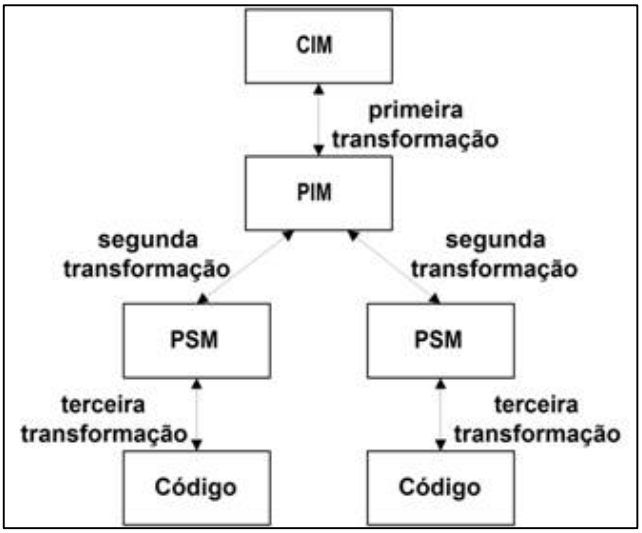


Figura 5 – Passos da especificação MDA

Apesar de MDA oferecer tais benefícios, ainda não encontra-se totalmente evoluída. Para obter os benefícios, todas as transformações entre os modelos devem ser automatizadas, mas o estado atual da tecnologia ainda não permite plenamente.

## 2.8 TRABALHOS CORRELATOS

Algumas ferramentas desempenham papel semelhante ao proposto no presente trabalho, cada qual com as suas peculiaridades e para determinadas linguagens. Dentre elas destaca-se a ferramenta para padronização de interfaces para sistemas de informação (PIRES e SIQUEIRA, 2004).

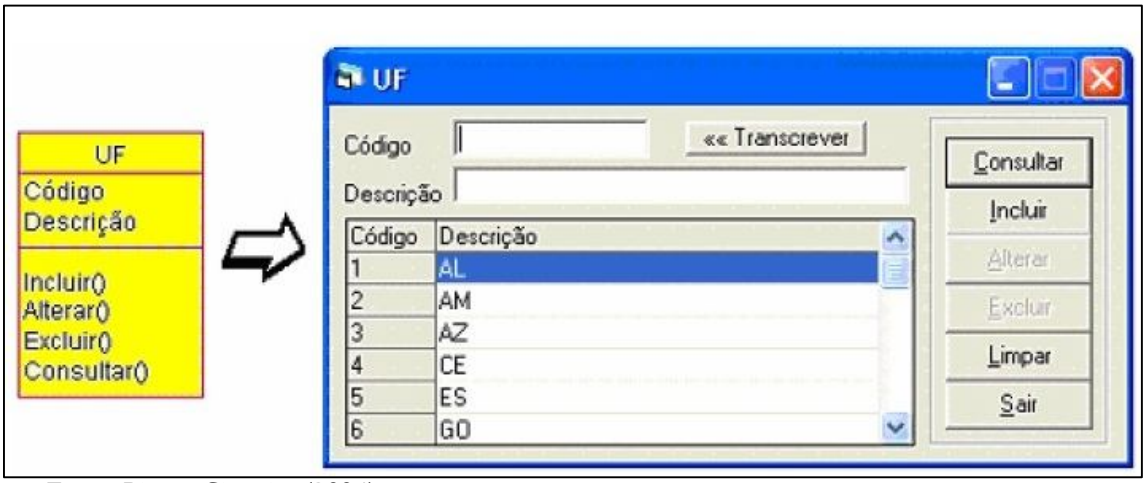
No trabalho desenvolvido por Pires e Siqueira (2004), a idéia é gerar interfaces a partir de diagrama de classes modelados na ferramenta UMLStudio (ferramenta para modelagem de diagrama de classes). Propõe-se a padronização de interface a fim de facilitar o processo de levantamento de requisitos. A ferramenta adota *templates* e gera códigos automaticamente. Para a definição deste padrão utilizou-se o seguinte:

a) o nome da classe é usado para dar o nome à interface a ser criada;
b) os serviços básicos das classes são representados pelos botões de comando na interface visual, além de mais dois botões: “limpar” e “sair” para melhorar as funcionalidades do sistema;

c) os atributos serão transformados em caixas de texto;

d) os atributos que são originados da associação com uma tabela básica de padronização são apresentados neste caso como um *combobox*, devidamente preenchido com os dados da tabela que contém os atributos.

Na figura 6 é apresentado um exemplo da interface gerada a partir da classe de unidade de federação.




Fonte: Pires e Siqueira (2004).

Figura 6 – Interface gerada a partir da classe de unidade de federação

# 3 DESENVOLVIMENTO DO APLICATIVO

De acordo com os objetivos propostos no trabalho, desenvolveu-se um aplicativo para construção de interfaces de software *standalone* a partir de diagrama de classes conforme critérios ergonômicos de usabilidade para auxiliar na qualidade e agilidade de um projeto. Desta forma, apresenta-se a seguir, os requisitos principais, diagramas, especificações, a implementação e por fim os resultados obtidos com a realização do presente trabalho.

## 3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Nos quadros 1 e 2 são apresentados respectivamente os requisitos funcionais e não funcionais do aplicativo.

| **REQUISITOS FUNCIONAIS** | |
|---|---|
| RF01 | O aplicativo deve permitir que o usuário configure as máscaras de data, hora, valor e telefone. |
| RF02 | O aplicativo deve permitir que o usuário configure as mensagens ao excluir, cancelar e fechar. |
| RF03 | O aplicativo deve permitir que o usuário defina as teclas de atalhos. |
| RF04 | O aplicativo deve permitir que o usuário defina os ícones dos botões de inserir, cancelar, gravar, excluir, fechar e procurar. |
| RF05 | O aplicativo deve permitir que usuário salve as configurações dos parâmetros gerais. |
| RF06 | O aplicativo deve permitir que o usuário selecione o arquivo XMI a ser importado. |
| RF07 | O aplicativo deve permitir que o usuário recarregue o arquivo XMI para atualizar as informações. |
| RF08 | O aplicativo deve permitir a visualização das interfaces, campos e relacionamentos importados. |
| RF09 | O aplicativo deve permitir editar a altura, largura e visibilidade de cada interface. |
| RF10 | O aplicativo deve permitir definir para cada campo os *hints*, visibilidade, alinhamento e máscara. |
| RF11 | O aplicativo deve permitir escolher a pasta do projeto onde deseja salvar os *templates*, bibliotecas e as interfaces geradas. |
| RF12 | O aplicativo deve permitir que o usuário visualize e edite os arquivos XAML gerados para ajustar algumas funcionalidades de cada interface. |

Quadro 1 – Requisitos funcionais

| **REQUISITOS NÃO FUNCIONAIS** | |
|---|---|
| RNF01 | A ferramenta deve utilizar .NET Framework 3.0 para ser compatível com a nova tecnologia WPF para geração de interfaces no formato XAML. |
| RNF02 | Ser implementado usando o ambiente Microsoft Visual C# 2008 Express Edition. |
| RNF03 | Utilizar o arquivo XML para armazenar as configurações dos parâmetros gerais de usabilidade. |
| RNF04 | Ser compatível com o sistema operacional Windows. |

Quadro 2 – Requisitos não funcionais

## 3.2 ESPECIFICAÇÃO

Foi utilizada a UML como linguagem para especificação dos diagramas de casos de uso, atividades e de classes, com a utilização da ferramenta *Enterprise Architect*. Os diagramas são detalhados na seção 3.2.1, 3.2.2 e 3.2.3.

### 3.2.1 Diagrama de casos de uso

O diagrama de casos de uso é formado por um único ator, que é responsável por configurar os parâmetros gerais para atender os critérios ergonômicos de usabilidade. Também administrar as configurações das interfaces, campos e relacionamentos a serem gerados. Na figura 7 é representado o usuário do aplicativo.

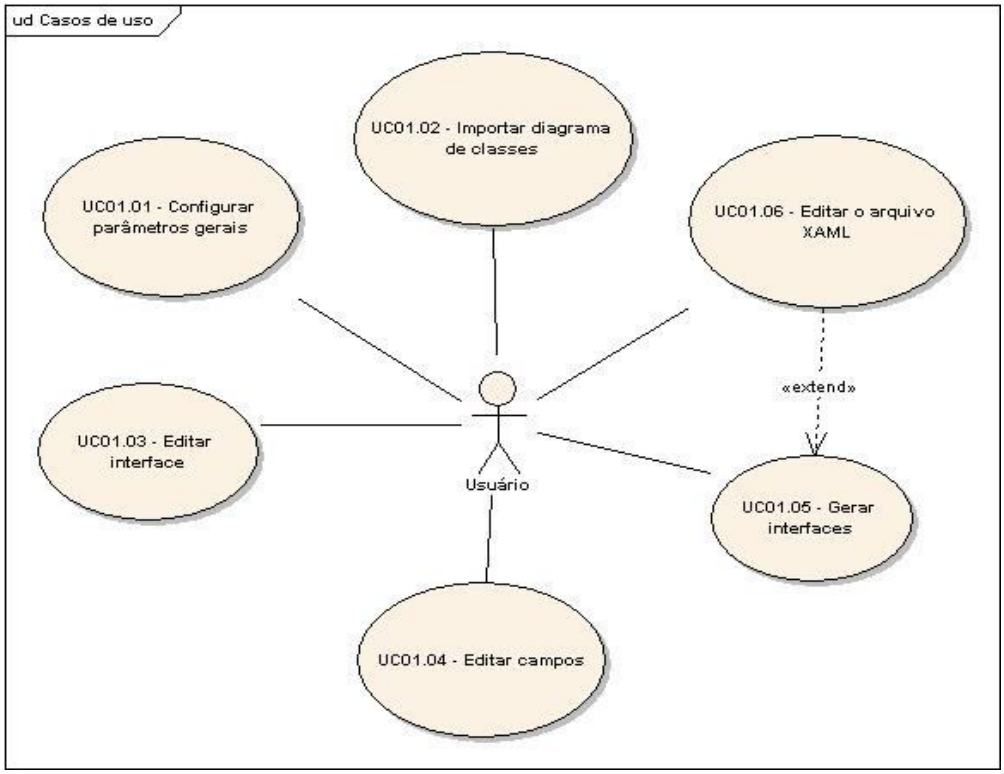



Figura 7 – Diagrama de casos de uso do aplicativo

Nos quadros 3, 4, 5, 6, 7 e 8 são apresentadas as descrições dos cenários principais, alternativos e de exceções de cada casos de uso.

| **UC01.01 - Configurar parâmetros gerais** |
|---|
| **Cenário principal:**<br>1. Sistema apresenta tela para o usuário definir as configurações para atender os critérios de usabilidade.<br>2. Usuário define as máscaras, teclas de atalho, mensagens, ícones, altura e largura de uma interface.<br>3. Usuário pressiona o botão **Continuar.**<br>4. Sistema valida as configurações.<br>5. Sistema salva as configurações no arquivo XML.<br>6. O usuário encerra o caso de uso. |
| **Cenários alternativos:**<br>A1. Cancelar: no passo 2, caso o usuário opte por cancelar:<br>e) Sistema fecha a tela de configuração e não prossegue para o próximo passo. |
| **Cenário de exceção:**<br>E1. No passo 2, caso o usuário defina uma tecla de atalho já existente:<br>2.1 A ferramenta apresenta a seguinte mensagem: “Tecla de atalho já definida”.<br>E2. No passo 4, caso o usuário não defina algum parâmetro:<br>4.1 A ferramenta apresenta uma mensagem conforme o parâmetro não definido. |
| **Pré-condição:** Ter acessado a opção para configuração.<br>**Pós-condição:** Um arquivo XML foi gerado. |

Quadro 3 – Caso de uso `configurar parâmetros gerais`

| **UC01.02 – Importar diagrama de classes** |
|---|
| **Cenário principal:**<br>1. Sistema apresenta tela para procurar o arquivo XMI, exportado através do EA.<br>2. Usuário pressiona **Procurar.**<br>3. Sistema valida o arquivo.<br>4. Usuário pressiona **Continuar**.<br>5. Sistema importa os diagramas para o aplicativo.<br>6. Sistema apresenta a tela principal com os dados carregados como interface, campos e relacionamentos. |
| **Cenários alternativos:**<br>A1. Cancelar: no passo 1, caso o usuário opte por cancelar:<br>1.1 Sistema fecha a tela de procura e sai do aplicativo. |
| **Cenário de exceção:**<br>E1. No passo 3, caso o usuário escolha um arquivo com formato inválido:<br>3.1 A ferramenta apresenta a seguinte mensagem: “Arquivo inválido”. |
| **Pré-condição:** Ter configurado os critérios ergonômicos de usabilidade.<br>**Pós-condição:** Um arquivo XMI foi carregado contendo as informações das interfaces. |

Quadro 4 – Caso de uso `importar diagrama de classes`

| **UC01.03 – Editar interface** |
|---|
| **Cenário principal:**<br>1. Sistema apresenta a tela para editar a interface.<br>2. Usuário define a largura, altura e visibilidade.<br>3. Usuário pressiona em **Salvar.**<br>4. Sistema valida o mínimo permitido da altura e largura.<br>5. Sistema salva as configurações.<br>6. Usuário opta por outra opção ou encerra o caso de uso. |
| **Cenários alternativos:**<br>A1. Cancelar: no passo 1, caso o usuário opte por cancelar:<br>1.1 Sistema cancela a alteração e volta às informações anteriores.<br>A2. Fechar: no passo 1, caso o usuário opte por fechar:<br>1.2 Sistema fecha a janela atual e volta para a janela principal. |
| **Cenário de exceção:**<br>E1. No passo 4, caso o usuário defina uma altura ou largura inválida:<br>4.1 A ferramenta apresenta a seguinte mensagem: “Tamanho mínimo permitido inválido”. |
| **Pré-condição:** Os parâmetros gerais contidos no arquivo XML precisam existir.<br>**Pós-condição:** Uma interface foi configurada. |

Quadro 5 – Caso de uso `editar interface`

| **UC01.04 – Editar campo** |
|---|
| **Cenário principal:**<br>1. Sistema apresenta a tela para editar o campo.<br>2. Usuário opta por editar as configurações como tamanho, alinhamento, *hint* e a máscara.<br>3. Usuário pressiona em **Salvar.**<br>4. Sistema salva as configurações.<br>5. Usuário opta por outra opção ou encerra o caso de uso. |
| **Cenários alternativos:**<br>A1. Cancelar: no passo 1, caso o usuário opte por cancelar:<br>1.1 Sistema cancela a alteração e volta as informações anteriores.<br>A2. Fechar: no passo 1, caso o usuário opte por fechar:<br>1.2 Sistema fecha a janela atual e volta para a janela principal. |
| **Pré-condição:** Os parâmetros gerais contidos no arquivo XML precisam existir.<br>**Pós-condição:** Um campo foi configurado. |

Quadro 6 – Caso de uso `editar campo`

| **UC01.05 – Gerar interfaces** |
|---|
| **Cenário principal:**<br>1. Sistema apresenta a tela principal.<br>2. Usuário pressiona em **Gerar.**<br>3. Sistema apresenta tela para procurar a pasta do projeto onde deseja salvar.<br>4. Sistema valida.<br>5. Sistema gera as interfaces.<br>6. O usuário opta por outra opção ou encerra o caso de uso. |
| **Cenários alternativos:**<br>A1. Fechar: no passo 1, caso o usuário opte por fechar:<br>1.1 Sistema fecha o aplicativo.<br>A2. Editar XAML: no passo 5, caso o usuário opte por editar XAML:<br>1.2 Vide caso de uso UC01.06. |
| **Cenário de exceção:**<br>E1. No passo 4, caso ocorra alguma exceção:<br>4.1 A ferramenta apresenta a seguinte mensagem: " Problemas ao gerar as interfaces".<br>4.2 Sistema não gera as interfaces.<br>4.3 Sistema volta para o passo 1. |
| **Pré-condição:** Os parâmetros gerais contidos no arquivo XML e a pasta `Templates` com os arquivos `MyMaskedTextBox.cs`, `TemplateCadastro.xaml` e `TemplateCadastro.xaml.cs` precisam existir.<br>**Pós-condição:** As interfaces foram geradas. |

Quadro 7 – Caso de uso `gerar interfaces`

| **UC01.06 – Editar o arquivo XAML** |
|---|
| **Cenário principal:**<br>1. Sistema apresenta a tela para editar o arquivo XAML.<br>2. Usuário seleciona um arquivo para editar.<br>3. Sistema carrega o conteúdo do arquivo XAML.<br>4. Usuário altera o arquivo.<br>5. Usuário pressiona em **Salvar**.<br>6. Sistema valida o arquivo.<br>7. Usuário opta por outra opção ou encerra o caso de uso. |
| **Cenários alternativos:**<br>A1.Visualizar: no passo 3, caso o usuário opte por somente visualizar:<br>3.1 Usuário visualiza o arquivo XAML.<br>A2. Salvar como: no passo 5, caso o usuário opte por salvar como:<br>5.1 Sistema apresenta a tela para escolher a pasta onde deseja salvar o arquivo.<br>5.2 Sistema valida.<br>5.3 Sistema salva o arquivo.<br>5.4 Sistema retorna para o passo 1. |
| **Cenário de exceção:**<br>E1. No passo 6, caso ocorra alguma exceção:<br>6.1 A ferramenta apresenta a seguinte mensagem: " Arquivo XAML inválido". |
| **Pré-condição:** Os parâmetros gerais contidos no arquivo XML e os arquivos XAML precisam existir.<br>**Pós-condição:** As interfaces foram alteradas ou visualizadas. |

Quadro 8 – Caso de uso `editar o arquivo XAML`

Entre todos os casos de uso citados nos quadros anteriormente, destacam-se os casos de uso UC01.02 e UC01.05 que são os de maiores importância do projeto, devido a complexidade e finalidade do mesmo.

### 3.2.2 Diagrama de atividades

Esta seção apresenta o diagrama de atividades. Na figura 8 é ilustrado o processo para a geração das interfaces pela ferramenta, contemplando todos os casos de uso.

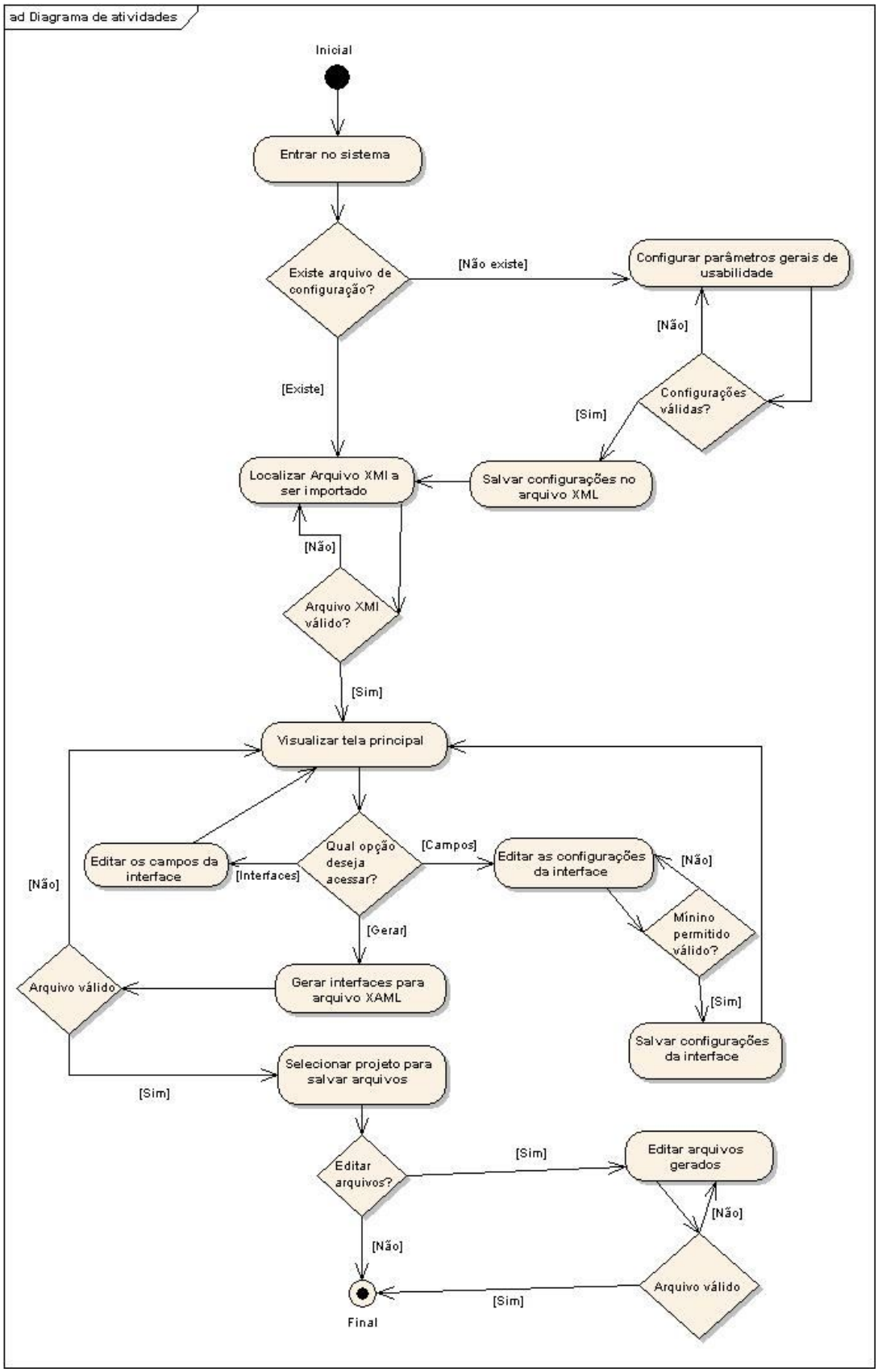



Figura 8 – Diagrama de atividades

### 3.2.3 Diagrama de classes

Esta seção apresenta o diagrama de classes e os objetivos das mesmas. A figura 9 ilustra o diagrama de classes.

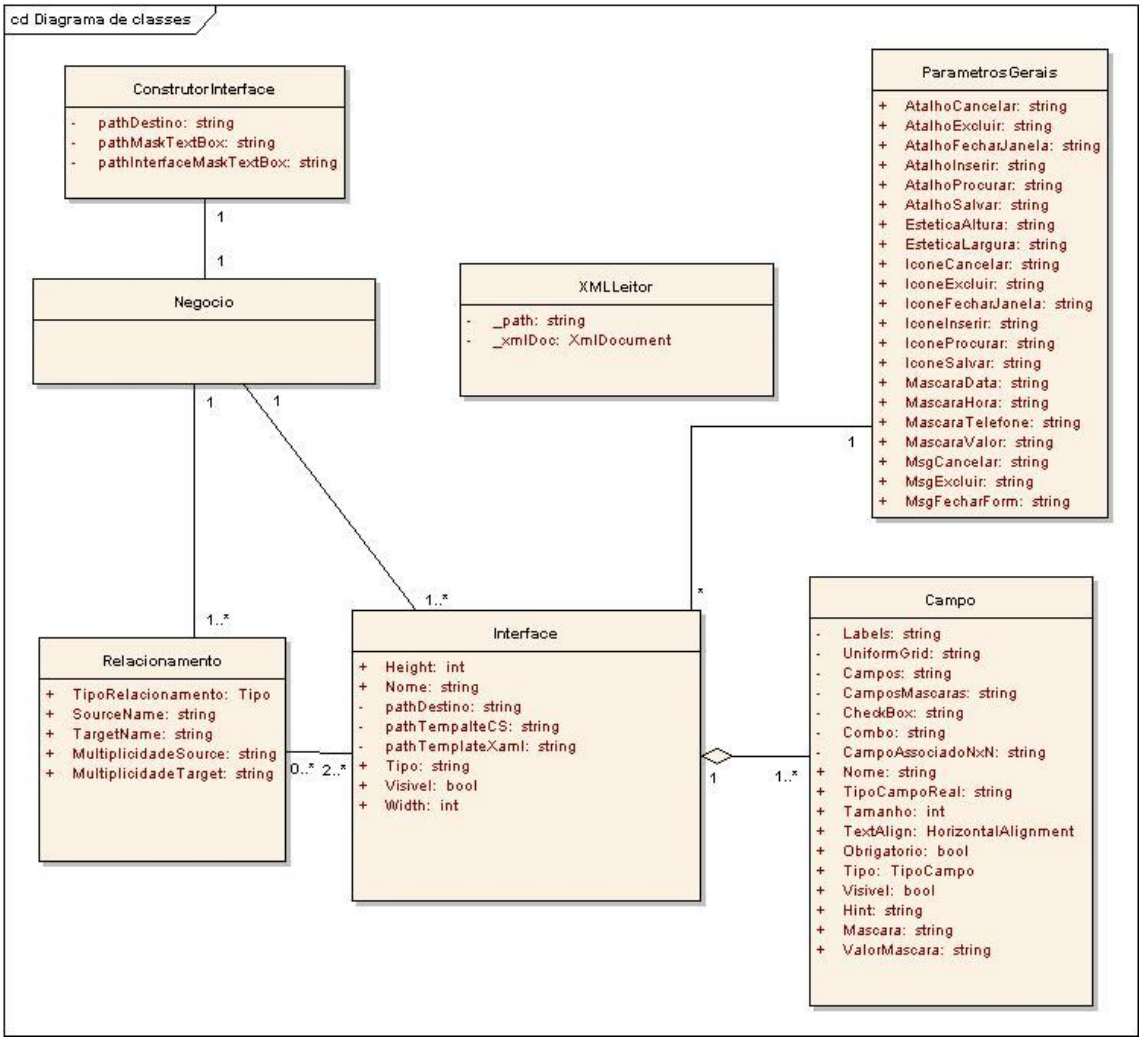



Figura 9 – Diagrama de classes

Os objetivos de cada classe são:

a) `Interface`: classe responsável por conter suas definições e gerar cada interface com seus campos, relacionamentos e aspecto visual em disco, no formato XAML, obtidos pela importação do arquivo XMI;

b) `Campo`: classe que contém as informações dos campos e métodos para salvar as alterações feitas pelo usuário na edição dos campos. A classe também é responsável em configurar as propriedades dos campos como máscara, *hints*, tamanho, alinhamento, etc.;

c) `ConstrutorInterface`: classe auxiliar para excluir os arquivos existente na pasta do projeto atual, carregar o *template* de cadastro que será um padrão de interface na qual todas as interfaces a serem geradas baseiam-se. A classe também é responsável por copiar o arquivo de configuração geral e as bibliotecas onde encontram-se os critérios de usabilidade para a pasta onde deseja salvar as interfaces geradas. Esta classe é associada a uma instância da classe de negócio para poder acessar cada interface e cada qual invocar o método gerar;
d) `Negocio`: classe auxiliar que contém as interfaces e relacionamentos obtidos através da importação do arquivo XMI. A classe também é responsável por definir os relacionamentos de uma interface;
e) `ParametrosGerais`: classe responsável por buscar e salvar os critérios de usabilidade no formato XML definidos pelo usuário;
f) `Relacionamento`: classe que contém as multiplicidades e as classes de origem e destino de cada relacionamento. Esta classe é importante para definir se uma interface tem relacionamento simples ou composto. Relacionamento simples são denominadas as classes com relacionamentos de multiplicidade 1 x 1 e relacionamentos compostos são classes com relacionamentos de multiplicidade 1 x N ou N x N.
g) `XMLLeitor`: classe mais importante do projeto. É responsável por buscar as informações contidas no arquivo XMI importado pelo aplicativo, percorrer o arquivo e obter as interfaces, campos, relacionamentos e as definições dos mesmos.

## 3.3 IMPLEMENTAÇÃO

Esta seção apresenta as ferramentas e técnicas utilizadas no desenvolvimento do trabalho proposto. São apresentados o ambiente de desenvolvimento, a linguagem de programação, a ferramenta para exportação das interfaces, as técnicas para importação do diagrama de classes, as técnicas para definição das usabilidades e por fim, as técnicas para geração das interfaces.

### 3.3.1 Técnicas e ferramentas utilizadas na implementação

Na implementação do aplicativo utilizou-se a linguagem de programação C#, em conjunto com o *.NET Framework* 3.0. Optou-se por utilizar esta linguagem por questão de aprendizado, flexibilidade e tendência do mercado. O ambiente escolhido para o desenvolvimento do aplicativo foi o Microsoft Visual C# 2008 Express Edition.

Microsoft Visual C# 2008 Express Edition é uma ferramenta gratuita e juntamente com o *.NET Framework* 3.0 oferece uma gama de componentes e bibliotecas para atender os mais variados tipos de funcionalidade. Por ser *Express Edition* (edição otimizada), a ferramenta tem algumas limitações, mas por outro lado, sua performance é excelente e o tamanho do executável é menor do que as ferramentas profissionais completas, podendo assim não preocupar-se muito com a questão de espaço físico e memória. Na figura 10 é apresentado a tela principal do ambiente Microsoft Visual C# 2008 Express Edition.

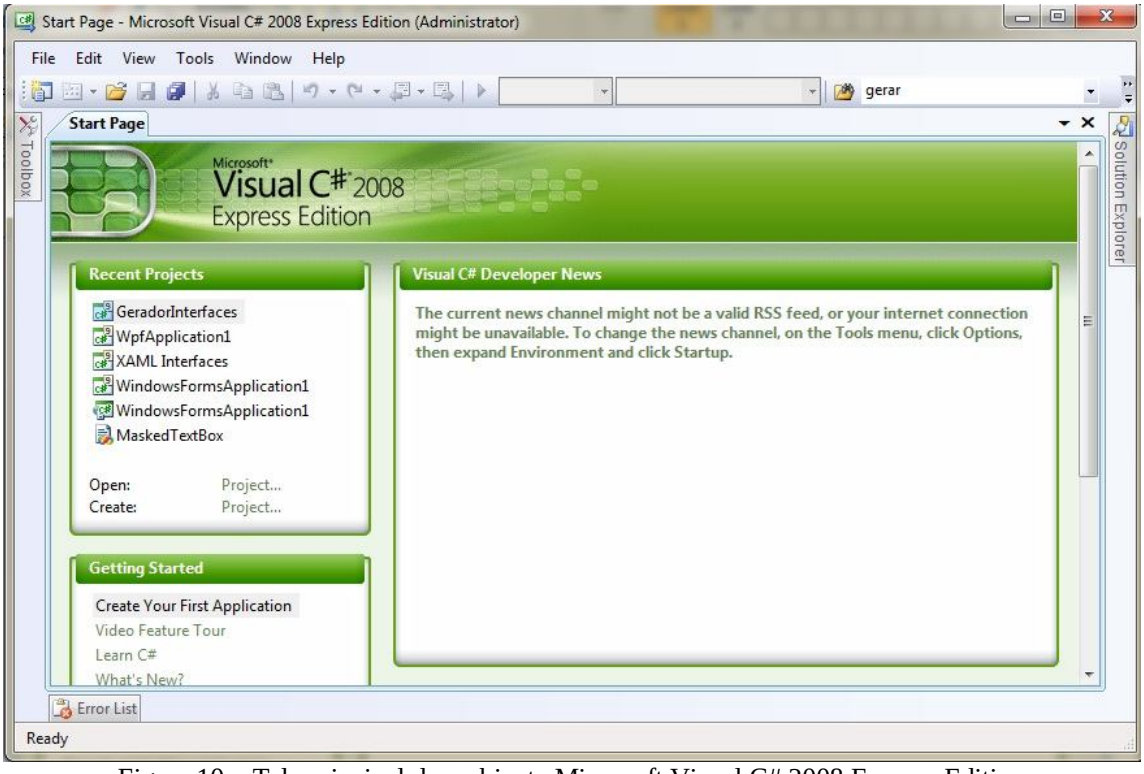



Figura 10 – Tela principal do ambiente Microsoft Visual C# 2008 Express Edition

#### 3.3.1.1 Enterprise Architect

Para a modelagem e exportação do diagrama de classes utiliza-se o EA. O EA permite que os diagramas sejam exportados no formato XMI. Leva-se em consideração algumas regras e limitações para a modelagem do diagrama que são as seguintes:

a) são reconhecidos relacionamentos de associação `1x1`, `1xN`, `NxN` e de agregação `1xN`;
b) serão reconhecidos campos do tipo `string`, `int`, `char`, `float`, `double`, `boolean`, `mascaraTelefone`, `mascaraValor`, `mascaraData` e `mascaraHora`;
c) para cada relacionamento cria-se um campo adicional automaticamente pelo aplicativo podendo ser, `Associado1x1`, `Associado1xN`, `AssociadoNxN`, `Agregado1X1`, `Agregado1xN` e `AgregadoNxN`;
d) o *hint* é reconhecido se caso o usuário preencha o campo *Notes* na opção *Attibutes*.

Na figura 11 é mostrado o diagrama de classes modelado através do EA e na figura 12 é apresentado o arquivo XMI exportado desse diagrama com foco na classe cliente.

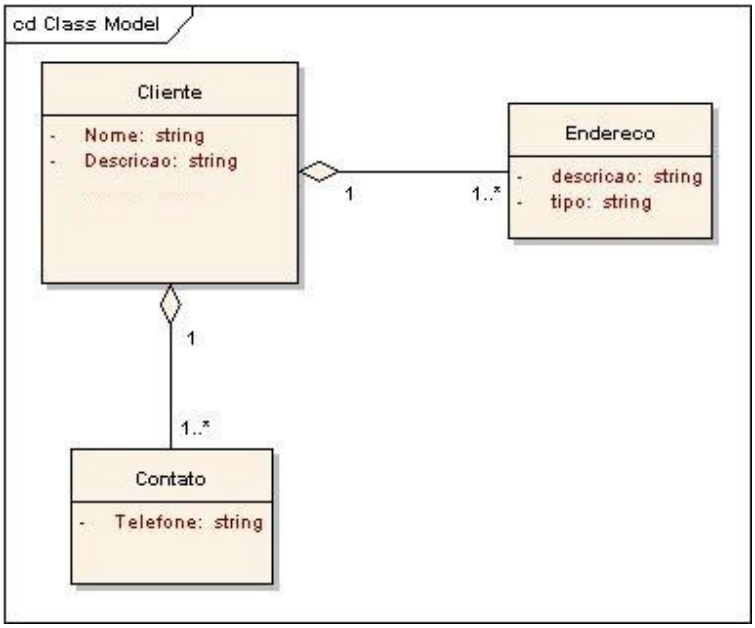



Figura 11 – Diagrama de classes exportado pelo EA

```
- <UML:Class name="Cliente" xmi.id="EAID_1888DE91_3C25_4887_AA5E_E44AA5E1AA1E" visibility="public"
    namespace="EAPK_78F5E6CA_02B7_4640_B819_2922AA5AC2B2" isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
  + <UML:ModelElement.taggedValue>
  - <UML:Classifier.feature>
    + <UML:Attribute name="Nome" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">
    + <UML:Attribute name="Descricao" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">
    </UML:Classifier.feature>
  </UML:Class>
- <UML:Association xmi.id="EAID_84F2FD69_CBDA_43fa_89DE_9765C5DB0E63" visibility="public" isRoot="false" isLeaf="false" isAbstract="false"
  - <UML:ModelElement.taggedValue>
      <UML:TaggedValue tag="style" value="3" />
      <UML:TaggedValue tag="ea_type" value="Aggregation" />
      <UML:TaggedValue tag="direction" value="Source -> Destination" />
      <UML:TaggedValue tag="linemode" value="3" />
      <UML:TaggedValue tag="linecolor" value="-1" />
      <UML:TaggedValue tag="linewidth" value="0" />
      <UML:TaggedValue tag="seqno" value="0" />
      <UML:TaggedValue tag="subtype" value="Weak" />
      <UML:TaggedValue tag="headStyle" value="0" />
      <UML:TaggedValue tag="lineStyle" value="0" />
      <UML:TaggedValue tag="ea_localid" value="8" />
      <UML:TaggedValue tag="ea_sourceName" value="Contato" />
      <UML:TaggedValue tag="ea_targetName" value="Cliente" />
      <UML:TaggedValue tag="ea_sourceType" value="Class" />
```

Figura 12 - Parte do arquivo XMI exportado pelo EA

Observa-se na figura 12 que é possível identificar o nome da classe, os atributos e os relacionamentos.

### 3.3.1.2 Técnica para importação dos dados XMI

Para a importação dos dados utiliza-se a biblioteca *System.XML* da própria ferramenta de desenvolvimento. Com essa biblioteca é possível obter as informações necessárias para identificar as interfaces, campos e relacionamentos. Na figura 13 mostra o método responsável pela leitura do arquivo XMI.

```
// rotina que le um arquivo xml conforme o caminho indicado
public void LerArquivoXml()
{
    //Limpar interfaces e negocios da memoria
    _negocio.LimparInterfaces();
    _negocio.LimparRelacionamentos();

    // Carre o arquivo XML
    _xmlDoc.Load(_path);

    // Retornaremos todas as interfaces
    XmlNodeList listaInterfaces = RetornarIntefacesPorNomeTag("UML:Class");

    // Retornar Todos os relaciomentos das interfaces.
    XmlNodeList listaRelacionamentos = RetornarRelacionamentosPorNomeTags("UML:Association");

    DefinirInterfaces(listaInterfaces);

    DefinirRelacionamentos(listaRelacionamentos);

    _negocio.DefinirRelacionamentosInterface();

    _negocio.DefinirCamposRelacionados();
}
```

Figura 13 – Método responsável pela leitura do arquivo XMI

Os métodos `RetornarInterfacesPorNomeTag` e `RetornarRelacionamentosPorNomeTags`, possuem as mesmas características. O objetivo é retornar uma lista de nodos que contenha um elemento por cada *tag* `UML:Class` e `UML:Association`, respectivamente. Assim pode-se obter todos os relacionamentos e interfaces encontradas no arquivo XMI. Nesse método destaca-se a classe `XmlNodeList` muito importante da biblioteca *System.XML*, pois esta classe representa uma coleção de nodos ordenada, permitindo retornar os nodos filhos e os nodos selecionados. A figura 14 apresenta o método `RetornarInterfacesPorNomeTag`.

```
// rotina que retorna uma coleção de nodos(interface) conforme o nome do elemento
public XmlNodeList RetornarIntefacesPorNomeTag(String iNomeElemento)
{
    XmlNodeList listaInterfaces = _xmlDoc.GetElementsByTagName(iNomeElemento);

    return listaInterfaces;
}
```

Figura 14 – Método `RetornarInterfacesPorNomeTag`

As características das interfaces, são obtidas através do método `DefinirAtributosInterface`, para o qual é passado uma coleção de atributos através dos quais obtém o necessário para definição da interface. Nesse método destacam-se duas classes para obtenção dos atributos que são `XmlNode` e `XmlAttributeCollection`. `XmlNode` contém as definições de um único nodo como nome, nodos filhos e a coleção de atributos. `XmlAttributeCollection` é responsável por conter os valores e nomes dos atributos de um nodo. A figura 15 ilustra o método `DefinirAtributosInterface`.

```
// rotina que varrer o conjunto de atributos que contem nesta interface conforme o nome
private Interface DefinirAtributosInterface(XmlAttributeCollection iAtributos)
{

    XmlNode nodeAux = iAtributos.GetNamedItem("name");

    Interface iInterface = new Interface(nodeAux.Value);
    iInterface.Visivel = true;
    iInterface.Tipo = "Simples";
    iInterface.Height = Convert.ToInt32(_parametrosGerais.EsteticaAltura);
    iInterface.Width = Convert.ToInt32(_parametrosGerais.EsteticaLargura);

    _negocio.AdicionarInterface(iInterface);

    return iInterface;
}
```

Figura 15 – Método `DefinirAtributosInterface`

Nas figuras 16 e 17 são ilustradas respectivamente a definição dos campos da interface através do método `DefinirCampos(XmlNode iInterface, Interface iObjetoInterface)` e `DefinirAtributosCampo(XmlNode iNodoPaiAtributos, XmlAttributeCollection iAtributos, Interface iObjetoInterface)`. O método `DefinirCampos()` é recursivo, já que a profundidade dos nodos são desconhecidos.

```
// 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);
            }
        }
    }

}
```

Figura 16 – Método `DefinirCampos`

```
// 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
```

Figura 17 – Método `DefinirAtributosCampos`

Na figura 18 é representado o método `DefinirRelacionamentosInterface` que vincula uma interface aos seus devidos relacionamentos.

```
public void DefinirRelacionamentosInterface()
{
    for (int i = 0; i < _interfaces.Count; i++)
    {
        for (int b = 0; b < _relacionamentos.Count; b++)
        {
            if (_relacionamentos[b].SourceName == _interfaces[i].Nome)
            {
                // retornar a interface pertencente ao relacionamento
                Interface interfaceRelacionamento;
                if (_relacionamentos[b].TipoRelacionamento == Relacionamento.Tipo.Aggregation)
                    interfaceRelacionamento = RetornarInterface(_relacionamentos[b].TargetName);
                else
                    interfaceRelacionamento = RetornarInterface(_relacionamentos[b].SourceName);

                _interfaces[i].AdicionarRelacionamento(_relacionamentos[b], interfaceRelacionamento);
            }

        }
    }
}
```

Figura 18 – Método `DefinirRelacionamentosInterface`

Para cada relacionamento existente, cria-se um campo com o nome da classe de relacionamento que auxilia na geração das interfaces. Na figura 19 é apresentado parte do método `DefinirCamposRelacionados`.

```
else if (_relacionamentos[i].TipoRelacionamento == Relacionamento.Tipo.Aggregation)
{
    if (_relacionamentos[i].InterfaceRelacionamento._campos[j].TipoCampoReal == _relacionamentos[i].SourceName)
    {
        if (_relacionamentos[i].InterfaceRelacionamento.Tipo == "Composto Associado")
            _relacionamentos[i].InterfaceRelacionamento.Tipo = "Composto Associado x Agregado";
        else if (_relacionamentos[i].InterfaceRelacionamento.Tipo == "Simples")
            _relacionamentos[i].InterfaceRelacionamento.Tipo = "Composto Agregado";

        _relacionamentos[i].InterfaceRelacionamento._campos[j].CampoInterfaceDestino = _relacionamentos[i].InterfaceR
        _relacionamentos[i].InterfaceRelacionamento._campos[j].CampoInterfaceOrigem = RetornarInterface(_relacionamen

        if ((_relacionamentos[i].MultiplicidadeSource == "1") && (_relacionamentos[i].MultiplicidadeTarget == "1"))
            _relacionamentos[i].InterfaceRelacionamento._campos[j].Tipo = Campo.TipoCampo.Agregado1X1;
        else if ((_relacionamentos[i].MultiplicidadeSource == "1..*") && (_relacionamentos[i].MultiplicidadeTarget ==
            _relacionamentos[i].InterfaceRelacionamento._campos[j].Tipo = Campo.TipoCampo.Agregado1xN;
        else if ((_relacionamentos[i].MultiplicidadeSource == "1..*") && (_relacionamentos[i].MultiplicidadeTarget ==
        {
            _relacionamentos[i].InterfaceRelacionamento._campos[j].Tipo = Campo.TipoCampo.AgregadoNxN;
        }

    }
}
```

Figura 19 – Parte do método `DefinirCamposRelacionados`

### 3.3.1.3 Técnicas para definição de usabilidades

Os parâmetros de usabilidades definidos pelo usuário são armazenados no arquivo

XML, através do método `GerarArquivoConfiguracaoGeral`, conforme apresentado na figura 20 e carregado pelo método `CarregarArquivoConfiguracaoGeral` apresentado na figura 21.

```
private void GerarArquivoConfiguracaoGeral()
{
    //gerar o arquivo para guardar as configurações inciais gerais;
    XmlTextWriter myXmlTextWriter = new XmlTextWriter("configuracoes.xml", null);
    myXmlTextWriter.Formatting = Formatting.Indented;
    myXmlTextWriter.WriteStartDocument(false);

    myXmlTextWriter.WriteStartElement("configuracoes");
    myXmlTextWriter.WriteStartElement("mascaras", null);
    myXmlTextWriter.WriteElementString("telefone", null, this._mascaraTelefone);
    myXmlTextWriter.WriteElementString("data", null, this._mascaraData);
    myXmlTextWriter.WriteElementString("hora", null, this._mascaraHora);
    myXmlTextWriter.WriteElementString("valor", null, this._mascaraValor);
    myXmlTextWriter.WriteEndElement();
    myXmlTextWriter.Flush();

    myXmlTextWriter.WriteStartElement("teclasAtalho", null);
    myXmlTextWriter.WriteElementString("inserir", null, this._atalhoInserir);
    myXmlTextWriter.WriteElementString("salvar", null, this._atalhoSalvar);
    myXmlTextWriter.WriteElementString("cancelar", null, this._atalhoCancelar);
    myXmlTextWriter.WriteElementString("excluir", null, this._atalhoExcluir);
    myXmlTextWriter.WriteElementString("procurar", null, this._atalhoProcurar);
    myXmlTextWriter.WriteElementString("fecharjanela", null, this._atalhoFecharJanela);
    myXmlTextWriter.WriteEndElement();
    myXmlTextWriter.Flush();

    myXmlTextWriter.WriteStartElement("mensagens", null);
    myXmlTextWriter.WriteElementString("excluir", null, this._msgExcluir);
    myXmlTextWriter.WriteElementString("cancelar", null, this._msgCancelar);
    myXmlTextWriter.WriteElementString("fechar", null, this._msgFecharForm);
    myXmlTextWriter.WriteEndElement();
```

Figura 20 – Método `GerarArquivoConfiguracaoGeral`

```
public void CarregarArquivoConfiguracaoGeral()
{
    //ler o arquivo de configuracao e extrai informacoes para os atributos
    if (!ExisteArquivoConfiguracaoIncial())
    {
        DefinirValoresDefault();
        GerarArquivoConfiguracaoGeral();
    }
    else
    {
        // Mascaras
        this._mascaraTelefone = RetornarValorNomeTag("configuracoes/mascaras/telefone");
        this._mascaraData = RetornarValorNomeTag("configuracoes/mascaras/data");
        this._mascaraHora = RetornarValorNomeTag("configuracoes/mascaras/hora");
        this._mascaraValor = RetornarValorNomeTag("configuracoes/mascaras/valor");

        //Teclas de atalhos
        this._atalhoInserir = RetornarValorNomeTag("configuracoes/teclasAtalho/inserir");
        this._atalhoSalvar = RetornarValorNomeTag("configuracoes/teclasAtalho/salvar");
        this._atalhoCancelar = RetornarValorNomeTag("configuracoes/teclasAtalho/cancelar");
        this._atalhoExcluir = RetornarValorNomeTag("configuracoes/teclasAtalho/excluir");
        this._atalhoProcurar = RetornarValorNomeTag("configuracoes/teclasAtalho/procurar");
        this._atalhoFecharJanela = RetornarValorNomeTag("configuracoes/teclasAtalho/fecharjanela");

        //Mensagens
        this._msgExcluir = RetornarValorNomeTag("configuracoes/mensagens/excluir");
        this._msgCancelar = RetornarValorNomeTag("configuracoes/mensagens/cancelar");
        this._msgFecharForm = RetornarValorNomeTag("configuracoes/mensagens/fechar");
```

Figura 21 – Método `CarregarArquivoConfiguracaoGeral`

#### 3.3.1.4 Técnicas para geração das interfaces

O método `GerarInterfaces(string pathDestino)` é o ponto de partida para geração de interfaces, conforme mostrado na figura 20 e também responsabiliza-se por copiar todos os arquivos necessários para o projeto onde deseja gerar as interfaces, eliminar os arquivos existentes e para cada interface, invocar o método `_interface.Gerar()`.

```
public void GerarInterfaces(string pathDestino)
{
    /* Primeiro Carregar o template Cadastro e configurar o template conforme parametros gerais
     * Salvar ele novamente no disco e carregar ele novamente e definir nome da interface e campos
     * Salvar na pasta definida pelo usuário
     */
    if (File.Exists(pathDestino + "\\MyMaskedTextBox.cs"))
    {
        File.Delete(pathDestino + "\\MyMaskedTextBox.cs");
        File.Copy(pathMaskTextBox, pathDestino + "\\MyMaskedTextBox.cs");
    }
    else
        File.Copy(pathMaskTextBox, pathDestino + "\\MyMaskedTextBox.cs");

    if (File.Exists(pathDestino + "\\configuracoes.xml"))
    {
        File.Delete(pathDestino + "\\configuracoes.xml");
        File.Copy("configuracoes.xml", pathDestino + "\\configuracoes.xml");
    }
    else
        File.Copy("configuracoes.xml", pathDestino + "\\configuracoes.xml");

    for (int i = 0; i < _negocio._interfaces.Count; i++)
    {
        string interfaceGerada;
        interfaceGerada = _negocio._interfaces[i].Gerar(pathDestino);

        if (interfaceGerada != "")
            _interfacesGeradas.Add(interfaceGerada);
    }
}
```

Figura 22 – Método `GerarInterfaces(string pathDestino)`

O método `interface.Gerar()`, contém sub-métodos como, `DefinirNomeInterface()`, `DefinirTamanhoInterface()`, `DefinirTeclasAtalhos()`, `DefinirIconesInterface()`, `DefinirCampos()`. O objetivo desses métodos é buscar dentro do arquivo “XAML” a *tag* responsável por conter a definição de determinada estrutura da interface. Após a localização das mesmas, é necessário percorrer todos os atributos dessa *tag* e definir os valores, ou até mesmo concatenar outras *tags* filhas para definição de uma nova estrutura.

Na figura 23 é mostrado parte do método `DefinirTeclasAtalhos()` que exemplifica a localização da *tag* e definição de um determinado valor para seu atributo.

```
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

Figura 23 – Métodos `DefinirTeclasAtalhos()`

Outros métodos de definição da interface seguem a mesma idéia ilustrado na figura 23, exceto o `DefinirCampos()`. A semelhança é a mesma, mas foram pré-definidas constantes com códigos XAML facilitando na junção com outros nodos e na atribuição de valores. O método responsável da biblioteca por unir dois trechos de códigos XAML distintos é o `InnerXML`.

Na figura 24 é mostrado parte das constantes pré-definidas, a figura 25 apresenta parte do código XAML que contém a estrutura do `tabItem` da interface e na figura 26 é detalhado a junção da constante `UniformGrid` com o código da figura 24 ocasionado pelo método `DefinirCampos` e sub-método `_campos[i].Gerar()`.

```
private const string UniformGrid = "<UniformGrid Columns='1' Height='48.907' Rows='2' Width='212' HorizontalAli
private const string Labels = "<Label Height='Auto' HorizontalAlignment='Left' VerticalAlignment='Top' Width='A
private const string Campos = "<TextBox Height='Auto' HorizontalAlignment='Stretch' VerticalAlignment='Stretch'
private const string CamposMascaras = "StayInFocusUntilValid='True' IgnoreSpace='True' Width='Auto' HorizontalA
private const string CheckBox = "<CheckBox Height='16' Width='120' Click='checkBox_Click'>CheckBox</CheckBox>";
private const string Combo = "<ComboBox Height='Auto' Width='Auto' DropDownClosed='comboBox_DropDownClosed'/>";
```

Figura 24 – Constantes pré-definidas

```
<TabControl Grid.Row="2" Name="tbcCadastro" Margin="0,10,0,4.411" Grid.RowSpan="2">
  <TabItem Header="Cadastro" Name="tbiCadastroPrincipal">
    <Grid Name="grdTabCadastro">
      <WrapPanel HorizontalAlignment="Stretch" Name="wrpTabPrincipal" Width="Auto" Grid.ColumnSpan="1"
      </WrapPanel>
    </Grid>
  </TabItem>
</TabControl>
```

Figura 25 – Parte do código XAML

```
<TabControl Grid.Row="2" Name="tbcCadastro" Margin="0,10,0,4.411" Grid.RowSpan="2">
  <TabItem Header="Cadastro" Name="tbiCadastroPrincipal">
    <Grid Name="grdTabCadastro">
      <WrapPanel HorizontalAlignment="Stretch" Name="wrpTabPrincipal" Width="Auto" Grid.ColumnSpan="1" G
        <UniformGrid Columns="1" Height="48.907" Rows="2" Width="100" HorizontalAlignment="Stretch">
          <Label Height="Auto" HorizontalAlignment="Left" VerticalAlignment="Top" Width="Auto">Código</L
          <TextBox Height="Auto" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" HorizontalCon
        </UniformGrid>
        <UniformGrid Columns="1" Height="48.907" Rows="2" Width="200" HorizontalAlignment="Stretch">
          <Label Height="Auto" HorizontalAlignment="Left" VerticalAlignment="Top" Width="Auto">Descrição
          <TextBox Height="Auto" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" HorizontalCon
        </UniformGrid>
      </WrapPanel>
    </Grid>
  </TabItem>
</TabControl>
```

Figura 26 – Junção XAML

O nodo que possui a tag `WrapPanel`, uniu-se com o nodo `UniformGrid`, através do método `InnerXML`.

### 3.3.2 Operacionalidade da implementação

Nesta seção são apresentadas algumas funcionalidades do aplicativo desenvolvido, através de um estudo de caso. A figura 27 mostra o diagrama de classes modelado pela ferramenta EA e importado pelo aplicativo para gerar as interfaces no padrão XAML.

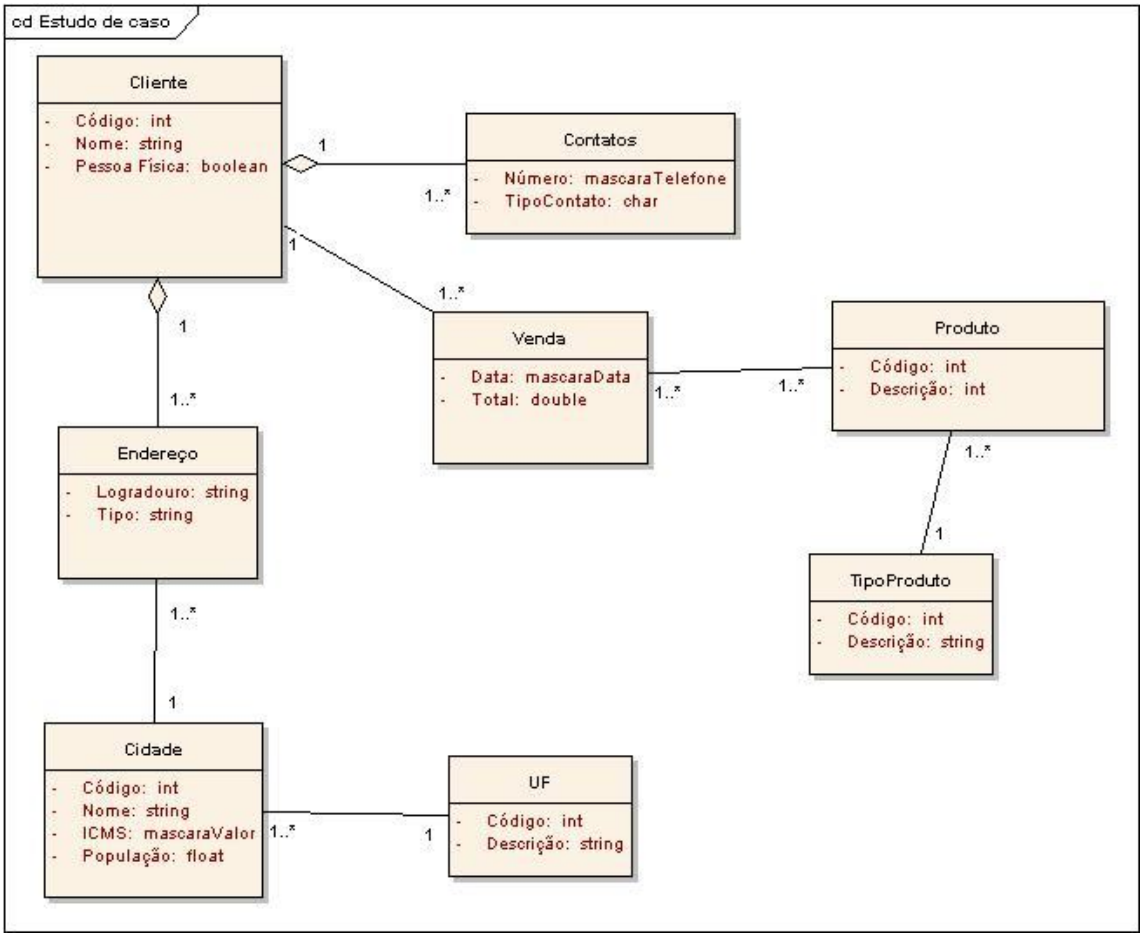



Figura 27 – Estudo de caso

Ao acessar o aplicativo e verificar que o arquivo `configuracoes.xml` não existe é apresentada a tela para configurar os parâmetros de critérios ergonômicos de usabilidade do sistema como é mostrado na figura 28. Esses parâmetros serão salvos no arquivo `configuracoes.xml` que está localizado na pasta raiz do executável. Faz-se necessário a existência desse arquivo, pois ele contém as mensagens, ícones, teclas de atalho, máscaras, altura e largura das interfaces geradas.

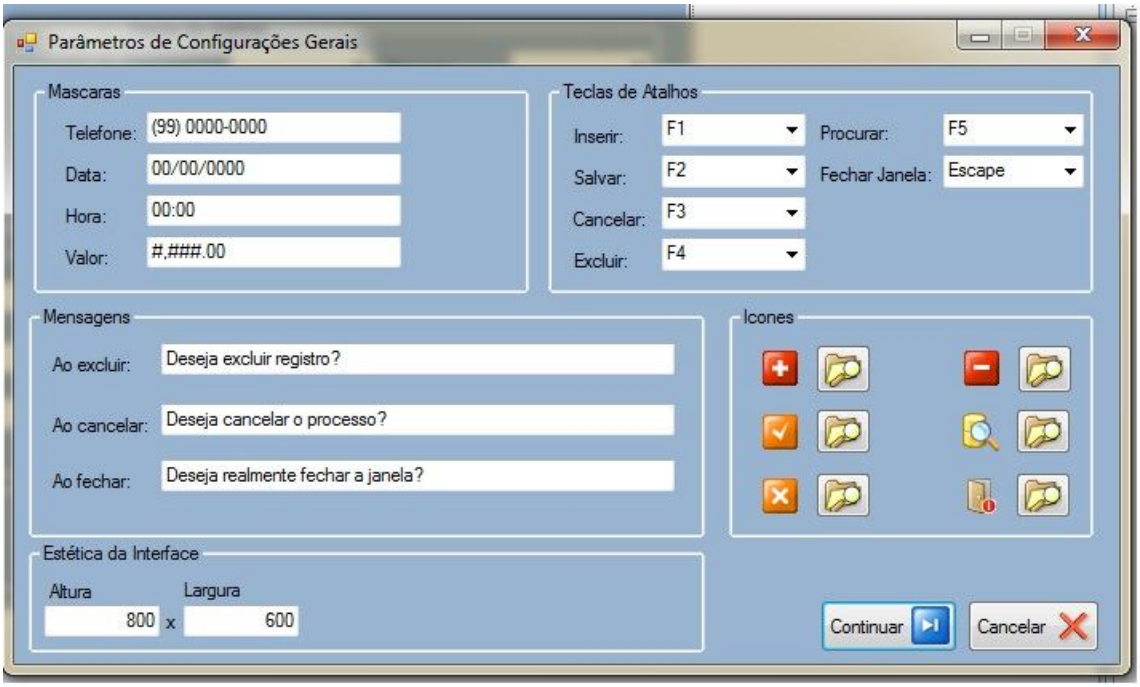



Figura 28 – Parâmetros de configurações gerais

O arquivo `configuracoes.xml` é exportado juntamente com as interfaces geradas para dentro da pasta do projeto, pois cada interface utiliza suas definições.

Após as configurações o aplicativo apresenta a tela para selecionar o arquivo XMI para importação, conforme é mostrado na figura 29.

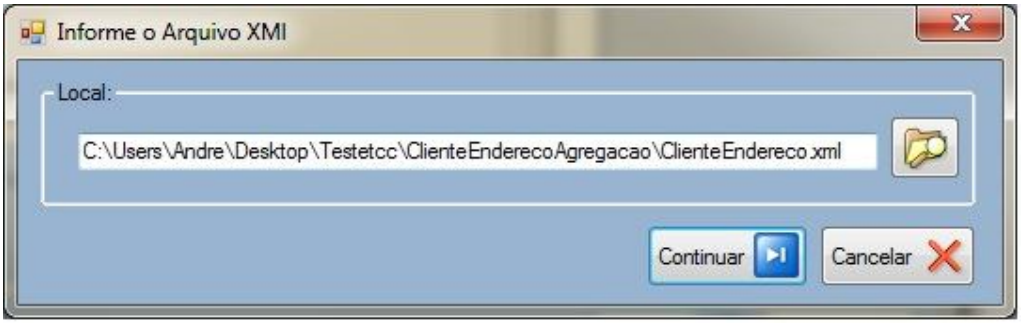



Figura 29 – Selecionar arquivo XMI

Na figura 30 é apresentado a tela principal onde pode-se visualizar as interfaces, relacionamentos, campos importados e configurar quaisquer opções a critério do usuário.

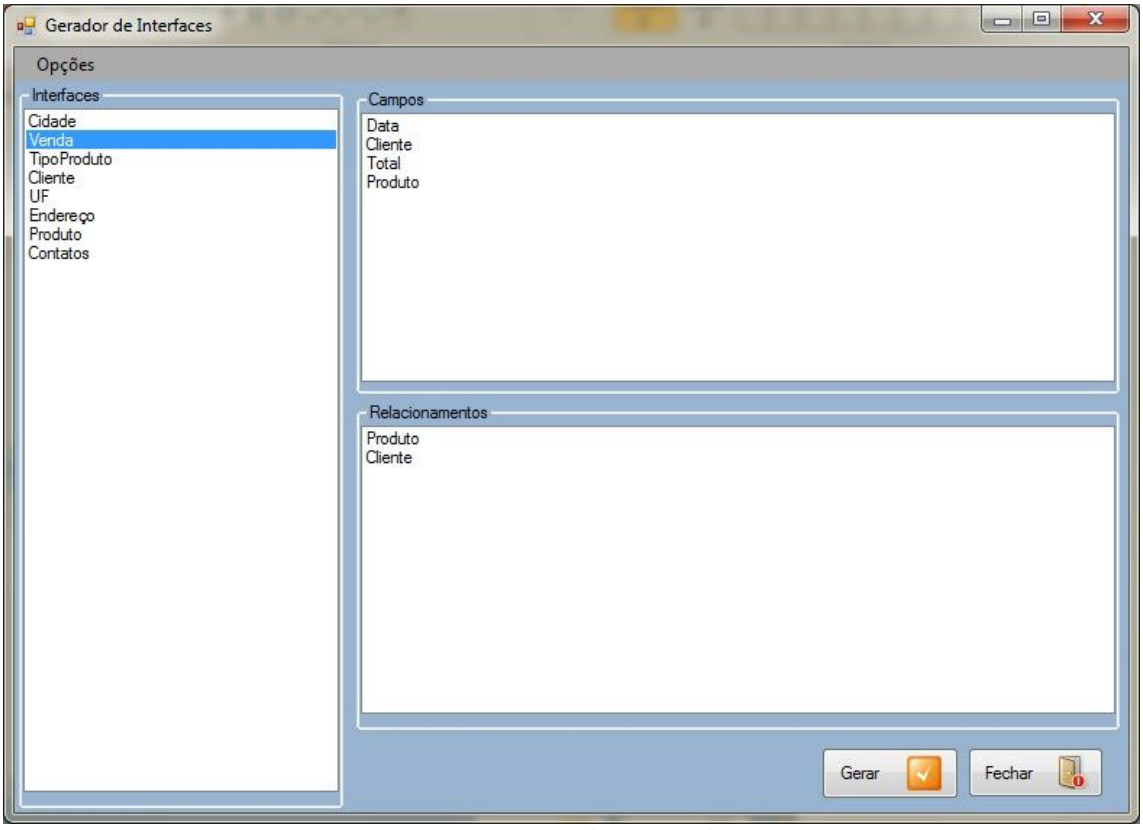



Figura 30 – Tela principal

Ao alterar a seleção das interfaces, os grupos `Campos` e `Relacionamentos` passam a apresentar o conteúdo de acordo com a interface selecionada.

O usuário pode optar em visualizar a interface dando um duplo clique. A tela apresentada é mostrada na figura 31. O usuário pode editar ou não, dependendo de sua necessidade. A seguir os campos são detalhados:

a) nome: campo somente leitura para questão de visualização;
b) altura: define a altura da interface a ser gerada;
c) lagura: define a largura da interface a ser gerada;
d) visível: define se a interface deve ser gerada ou não;
e) tipo: campo somente leitura. É definido automaticamente pelo aplicativo, podendo ser simples em interfaces sem relacionamentos, composto associado em interfaces somente com relacionamentos do tipo associação, composto agregado em interfaces somente com relacionamentos do tipo agregação e por fim composto agregado x associado em interfaces com os dois tipos de relacionamentos;
f) relacionamento: campo somente leitura, podendo ser do tipo *Association* ou *Aggregation*;

g) nome da origem: nome da interface de relacionamento de origem, campo somente leitura;
h) nome do destino: nome da interface de relacionamento de destino, campo somente leitura;
i) multiplicidade: campo somente leitura, refere-se multiplicidade dos relacionamentos.

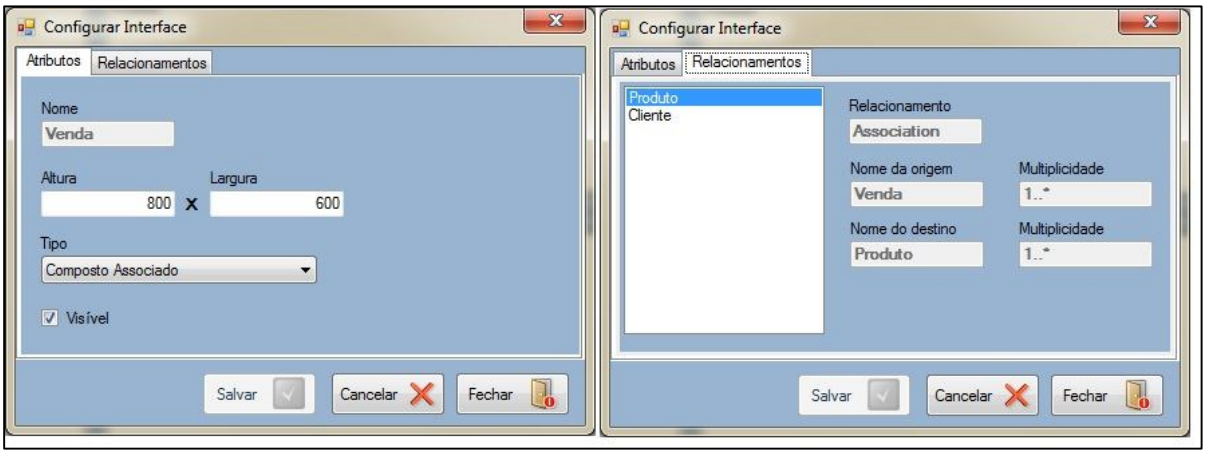


Figura 31 – Configurar interface

Através da tela principal também é possível visualizar os campos de cada interface com um duplo clique. A figura 32 mostra a tela para configurar o campo. O usuário pode editar ou não, dependendo de sua necessidade. A seguir os campos são detalhados:

a) nome: campo somente leitura;
b) tamanho: define a largura do campo na tela a ser gerado;
c) tipo: campo somente leitura e definido automaticamente pelo sistema, podendo ser `string`, `int`, `char`, `float`, `double`, `boolean`, `indefinido`, `mascaraTelefone`, `mascaraValor`, `mascaraData`, `mascaraHora`, `associado1x1`, `associado1xN`, `associadoNxN`, `agregado1x1`, `agregado1xN` e `agregadoNxN`;
d) alinhamento: campo somente leitura definido automaticamente pelo sistema, podendo ser alinhado a esquerda, para tipos alfanuméricos e alinhados a direita, para tipos numéricos;
e) *hint*: campo para explicar através de balões qual finalidade tem o mesmo;
f) máscara: define o tipo de máscara do campo, podendo ser valor, data, hora ou telefone.

Em relação aos tipos de campo é importante mencionar que os tipos indefinidos são os não suportados pelo aplicativo. Os campos `associado1x1`, `associado1xN` e `associadoNxN`,

são campos do tipo classe criado automaticamente pelo sistema através dos relacionamentos de associação. Os campos `agregado1x1`, `agregado1xN` e `agregadoNxN` são campos do tipo classe criado automaticamente pelo sistema através dos relacionamentos de agregação.


Figura 32 – Configurar campo

Após a definição das configurações o usuário pode optar pela geração das interfaces. O sistema apresenta a tela para selecionar o projeto onde deseja-se salvar os arquivos, conforme ilustrado na figura 33. Juntamente com as interfaces copia-se os arquivos `MyMaskedTextBox.cs`, localizado na pasta `Templates` dentro da pasta raiz do aplicativo, e o arquivo `configuracoes.xml`. É necessário que usuário já tenha os arquivos e as pastas mencionadas anteriormente, antes da geração das interfaces.

O arquivo `MyMaskedTextBox.cs` é uma classe que tem a implementação dos campos do tipo máscaras. O arquivo `configuracoes.xml` possui os parâmetros de usabilidade definido pelo usuário. As interfaces geradas para um determinado projeto necessitam-se dos arquivos citados anteriormente para definição dos critérios ergonômicos de usabilidade.

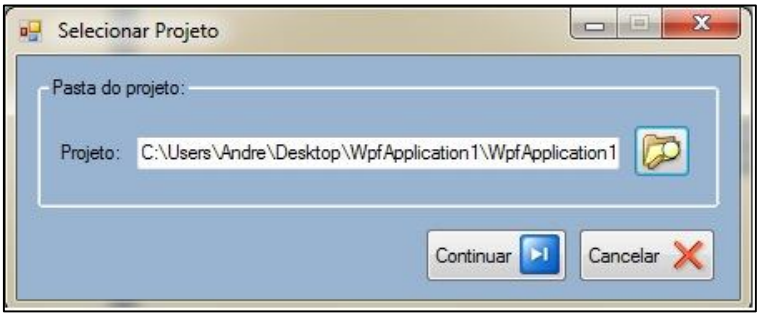



Figura 33 – Selecionar projeto

As interfaces geradas são baseadas no *template* `TemplateCadastro.xaml`, com sua classe de negócio `TemplateCadastro.cs`. Para cada interface (arquivo XAML) gerada deve-se existir um arquivo `.cs` , com o mesmo nome, e é neste que está contido as regras de negócio da interface como tratamentos de erros, confirmações, etc.

Foi criado um *template* de cadastro com objetivo de facilitar a geração e a padronização das interfaces. A figura 34 mostra a interface e na figura 35 é ilustrado o seu respectivo código XAML. Necessita-se da existência dos arquivos citados acima, pois é através dos mesmos que todas as interfaces são geradas.

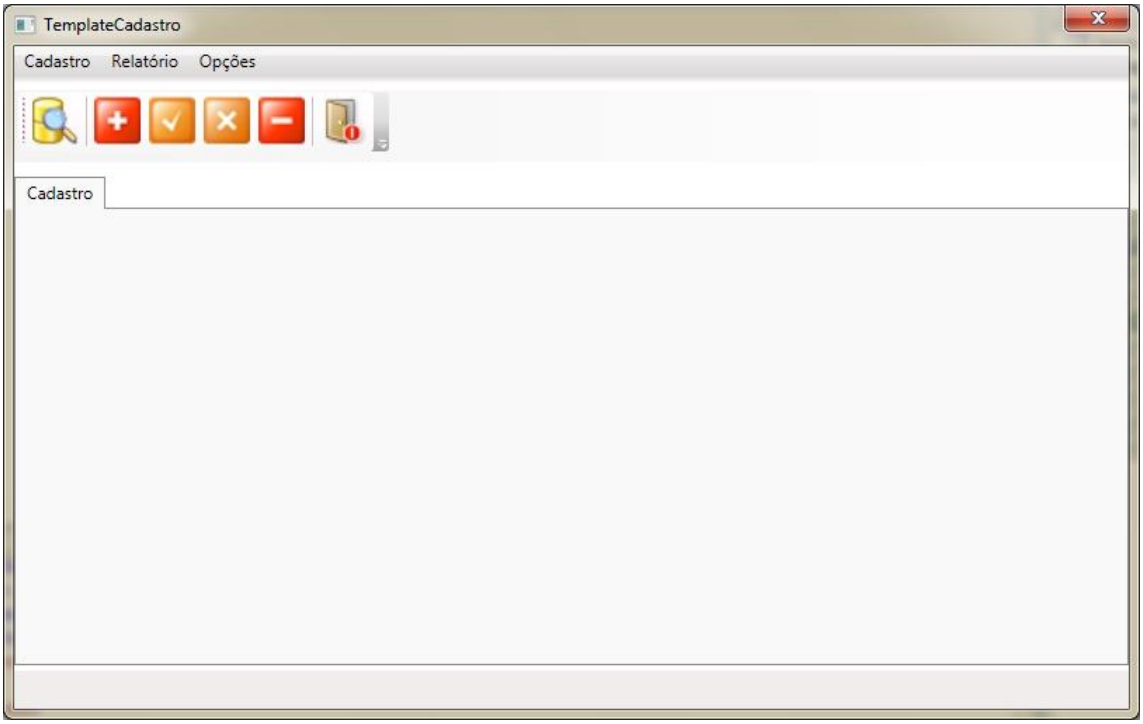



Figura 34 – *Template* de cadastro

```
<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" AutomationPro
        <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.Ro
      <ToolBar Band="1" BandIndex="1" Height="44" HorizontalAlignment="Stretch" VerticalContentAlignment="Stretch" Na
```

Figura 35 – Código XAML

Pode-se observar na figura 35 que as definições da interface estão contidas no código XAML acima. Um exemplo é a *tag* `<menuItem>`, onde é definido o menu da interface.

No *template* de cadastro estão definidas as seguintes funcionalidades:

a) tratamentos dos botões para desabilitar ou habilitar conforme o tipo de operação;

b) mensagens de confirmação ao excluir, cancelar e sair;

c) tratamentos das teclas de atalho;

d) definição dos botões e teclas de atalho;

e) definição dos menus;

f) tratamento para bloqueio de caracteres alfanuméricos em campos que são do tipo numérico;

g) carregar as usabilidades contidas no arquivo `configuracao.xml`.

Na figura 36 tem-se o diagrama de classes com relacionamento do tipo agregação, com destaque à classe cliente. A figura 37 mostra a interface que é o resultado da importação deste diagrama. Na figura 38 e 39 são apresentadas as abas de endereço e contato respectivamente.

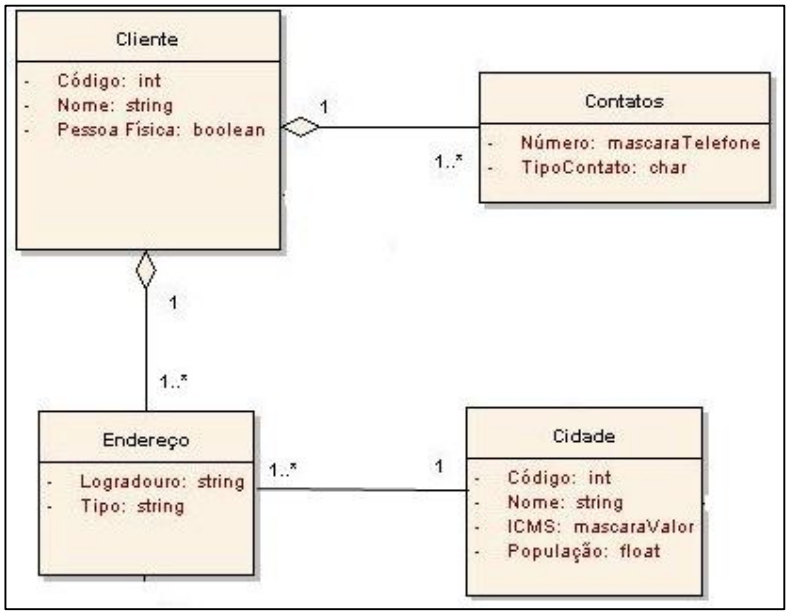



Figura 36 – Classe cliente

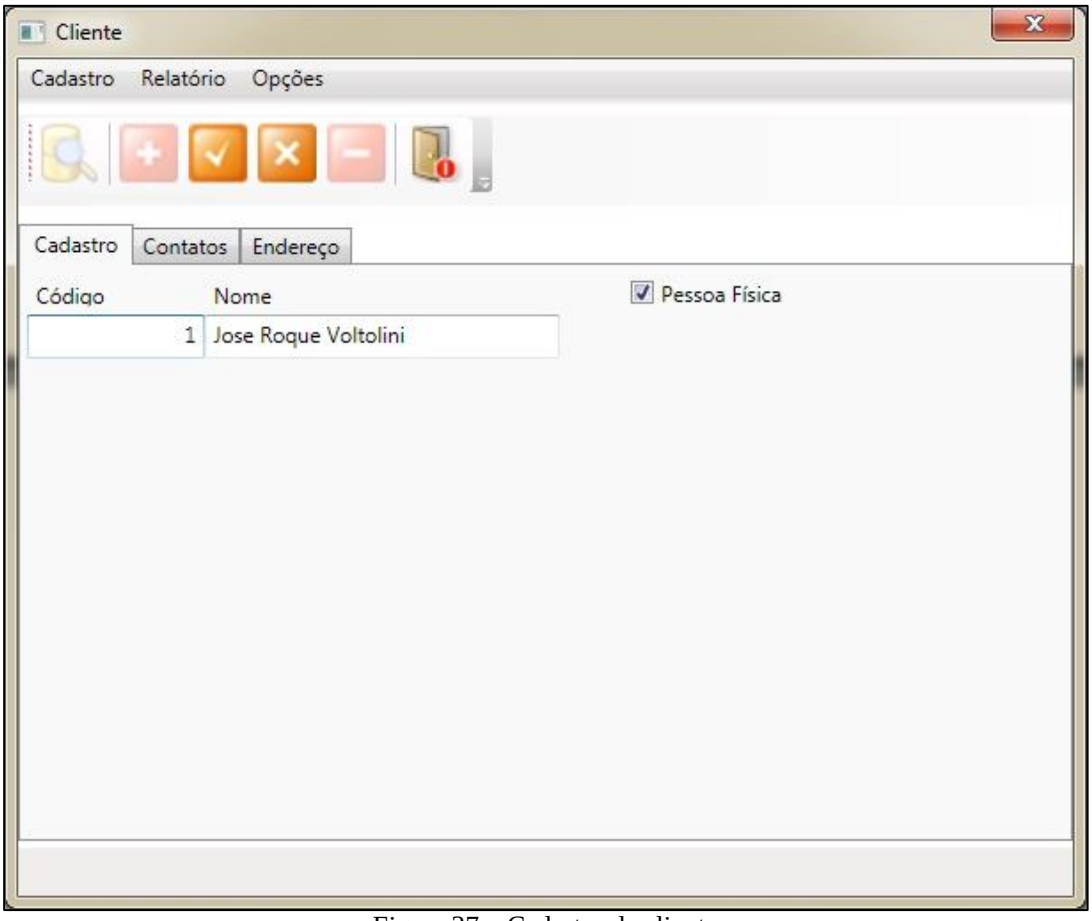



Figura 37 – Cadastro de cliente

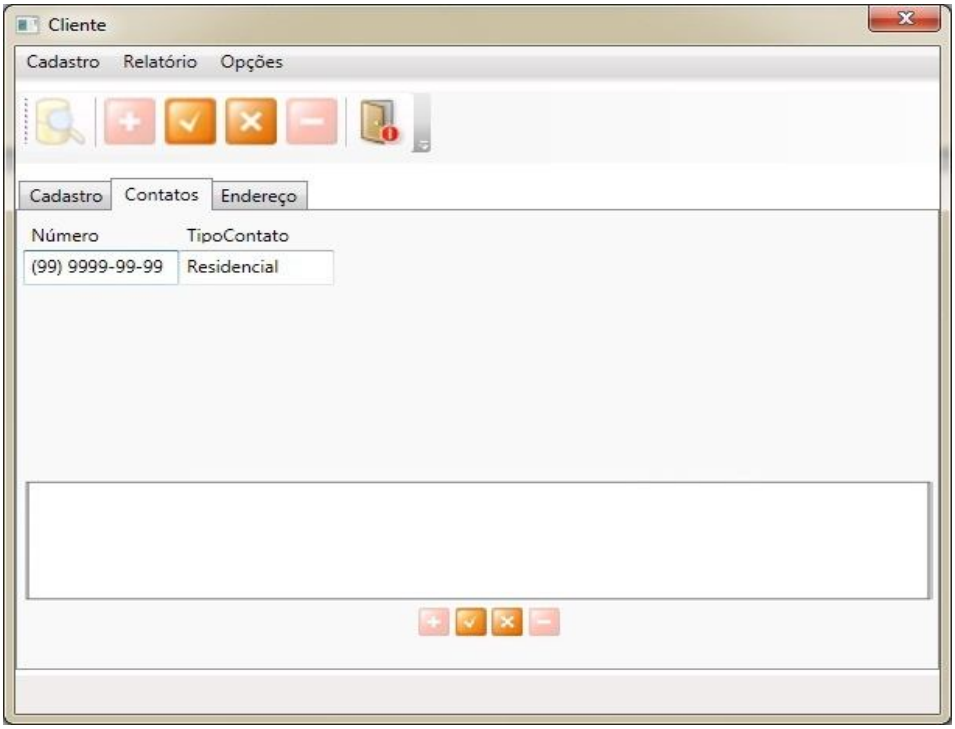



Figura 38 – Aba contatos

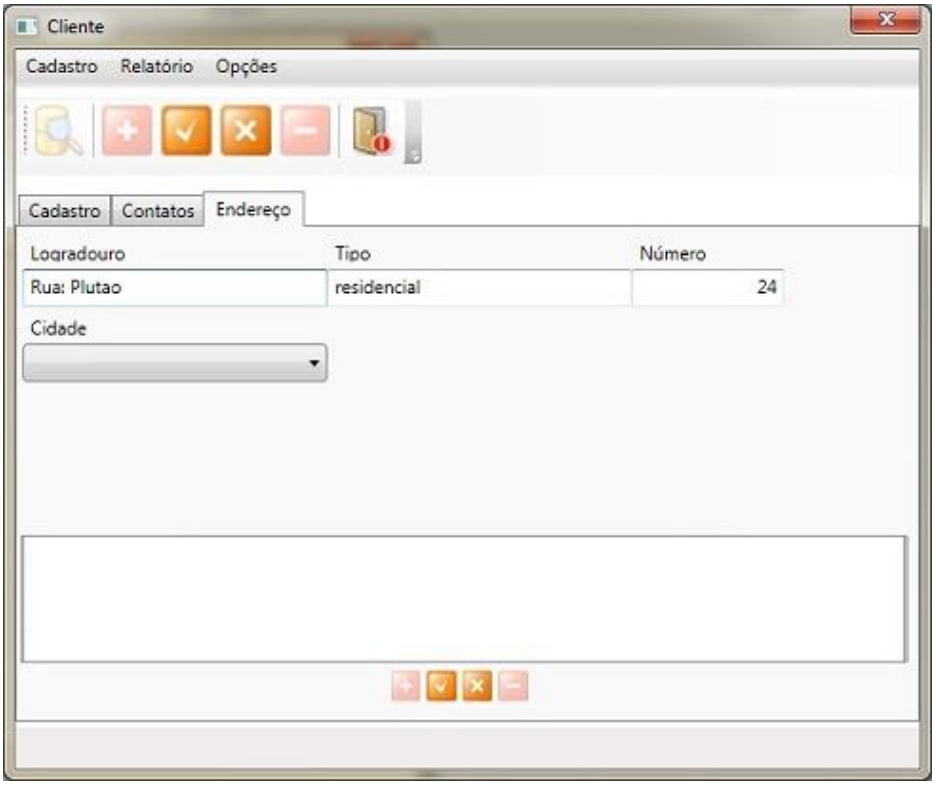



Figura 39 – Aba endereço

Na figura 40 tem-se o diagrama de classes do tipo associação n x n, com destaque a

classe venda e na figura 41 tem-se a interface que é o resultado da importação deste diagrama.

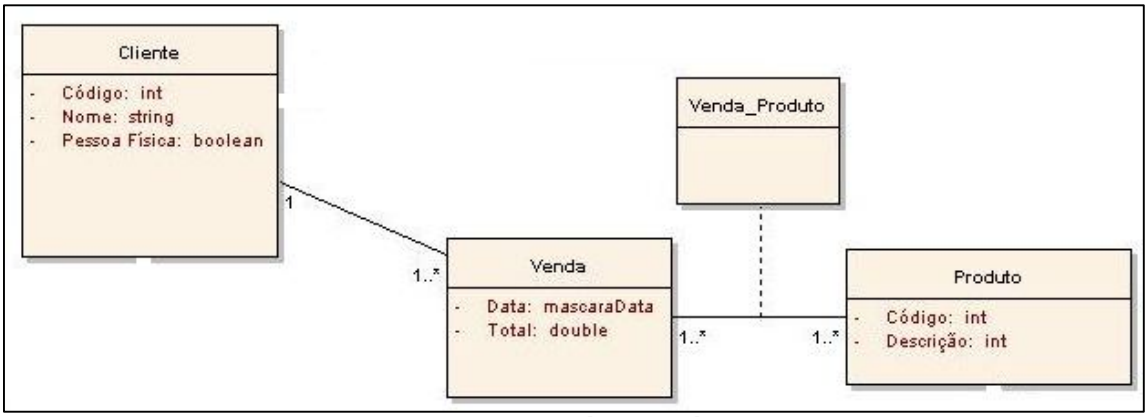



Figura 40 – Classe venda

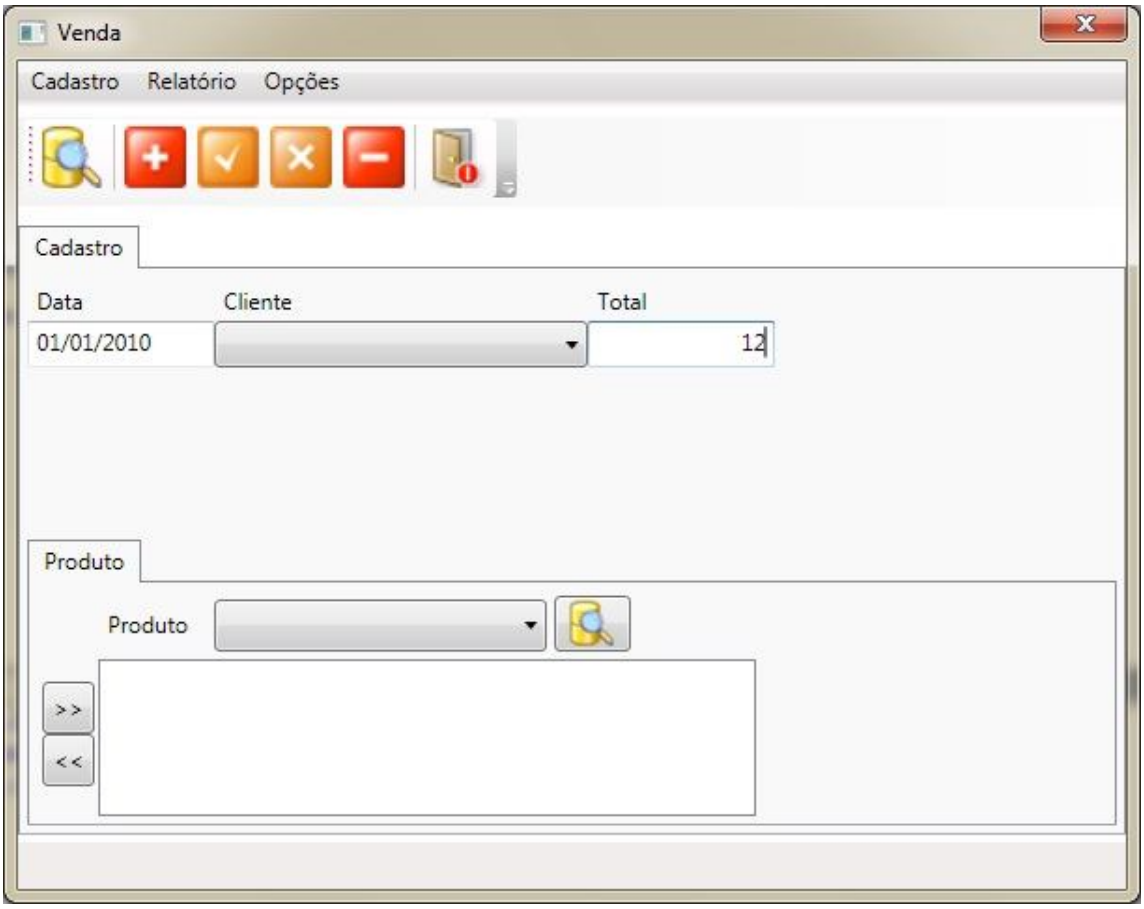



Figura 41 – Cadastro de venda

Na figura 42 tem-se o diagrama de classes do tipo associação 1 x n, com destaque a classe cidade e na figura 43 tem-se a interface que é o resultado da importação deste diagrama.

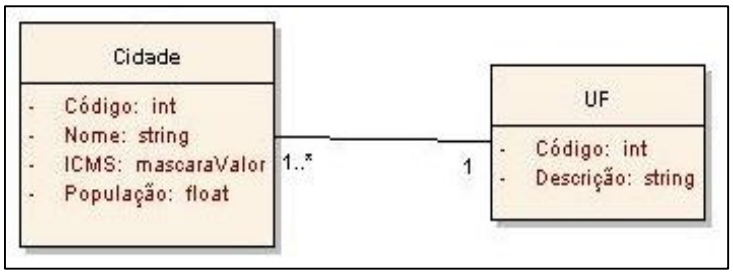



Figura 42 – Classe cidade

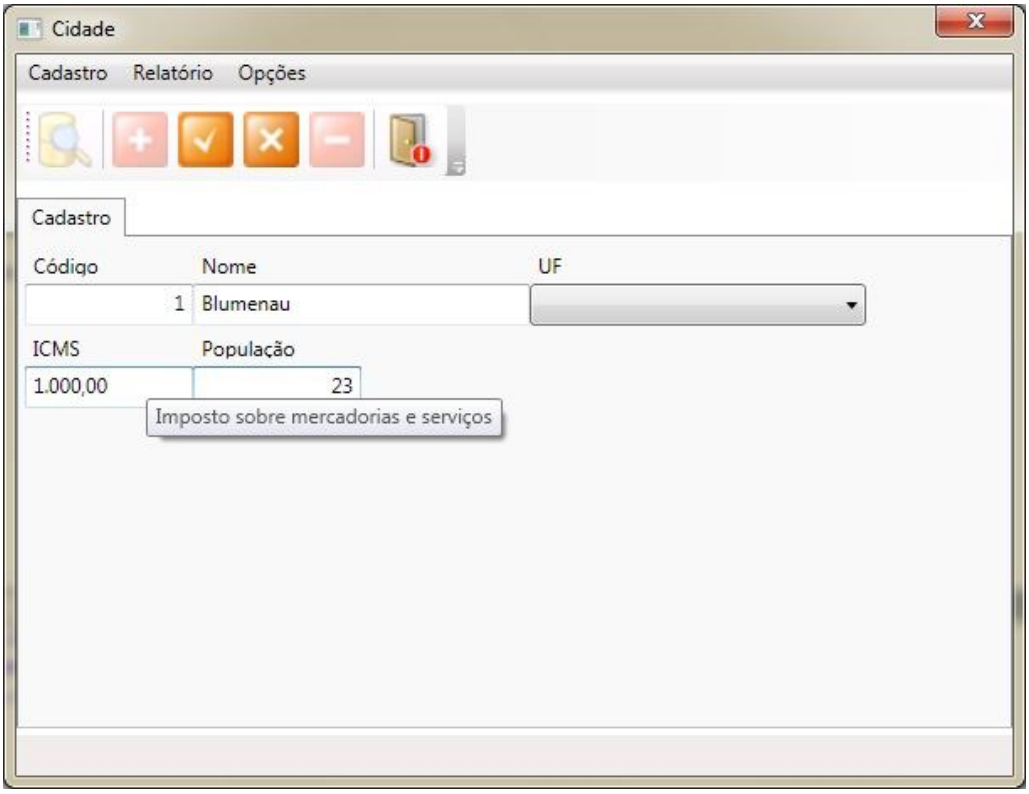



Figura 43 – Cadastro de cidade

Por fim na figura 44 tem-se o diagrama de classes sem relacionamento, com destaque a classe UF e a figura 45 mostra a interface que é o resultado da importação deste diagrama.


Figura 44 – Classe uf

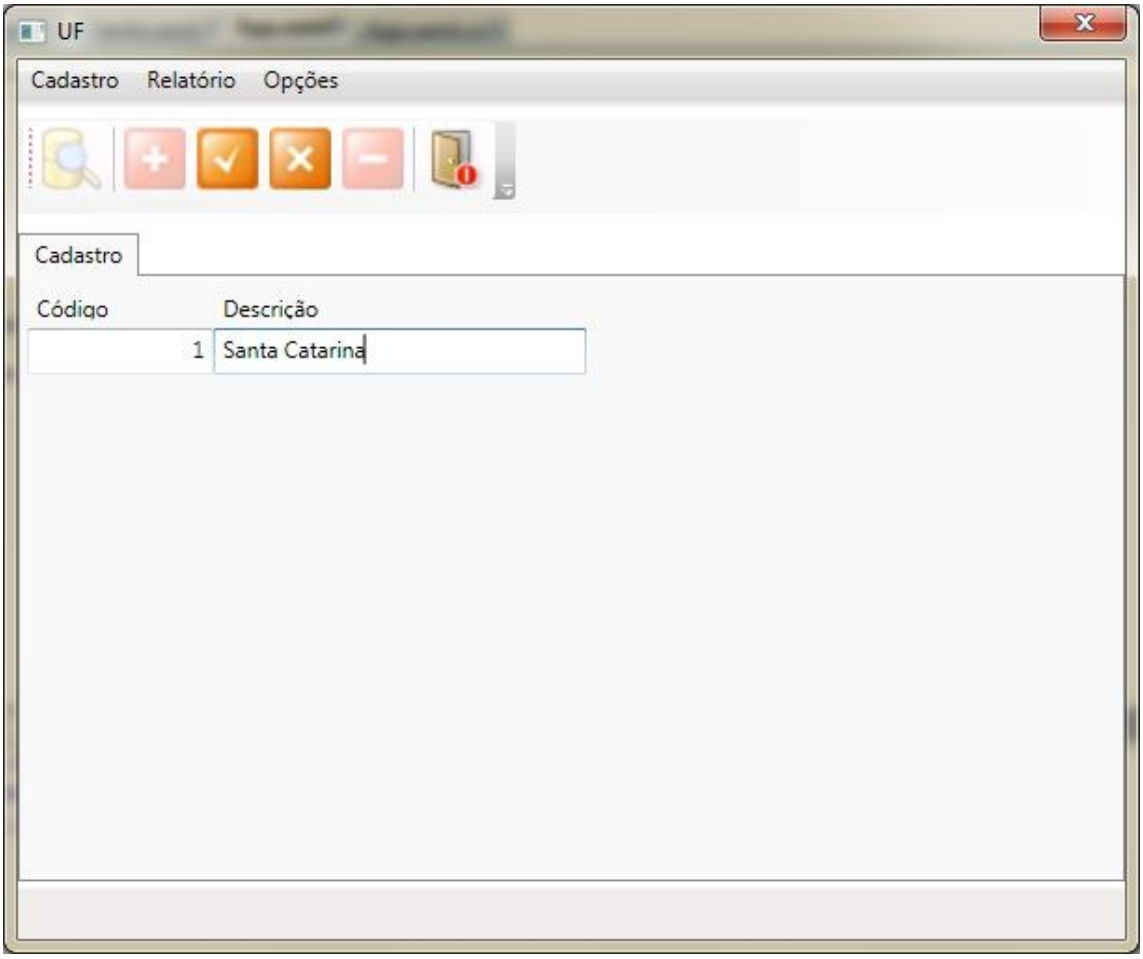



Figura 45 – Cadastro de uf

Conforme as imagens anteriores os seguintes critérios para geração das interfaces, campos foram definidos:

a) classes sem relacionamentos geram somente interfaces com uma aba principal, denominada cadastro;
b) 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;
c) 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;
d) atributos do tipo `int`, `float` e `double` geram campos *textbox* alinhados à direita;
e) atributos do tipo `string` e `char` geram campos *texbox* alinhados à esquerda;
f) atributos do tipo `mascaraValor`, `mascaraData`, `mascaraHora` e `mascaraTelefone` geram campos do tipo `MyMaskedTextBox` com o valor da

mascara definida pelo usuário no aplicativo;

g) atributos do tipo *class*, exemplo `UF:UF`, são automaticamente trocado pelo sistema para, `Associado1x1`, `Associado1xN`, `AssociadoNxN` ou `Agregado1xN`;

h) os atributos do tipo `Associado1x1` e `Associado1xN` geram campos do tipo *combobox* dentro da aba pertencente a sua origem;

i) 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 ;

j) 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.

As interfaces geradas poderão ser editadas conforme mostra a figura 46. No lado esquerdo estão os caminhos das interfaces geradas, ao clicar sobre cada item é carregado ao lado direito o conteúdo XAML para ser visualizado ou editado.

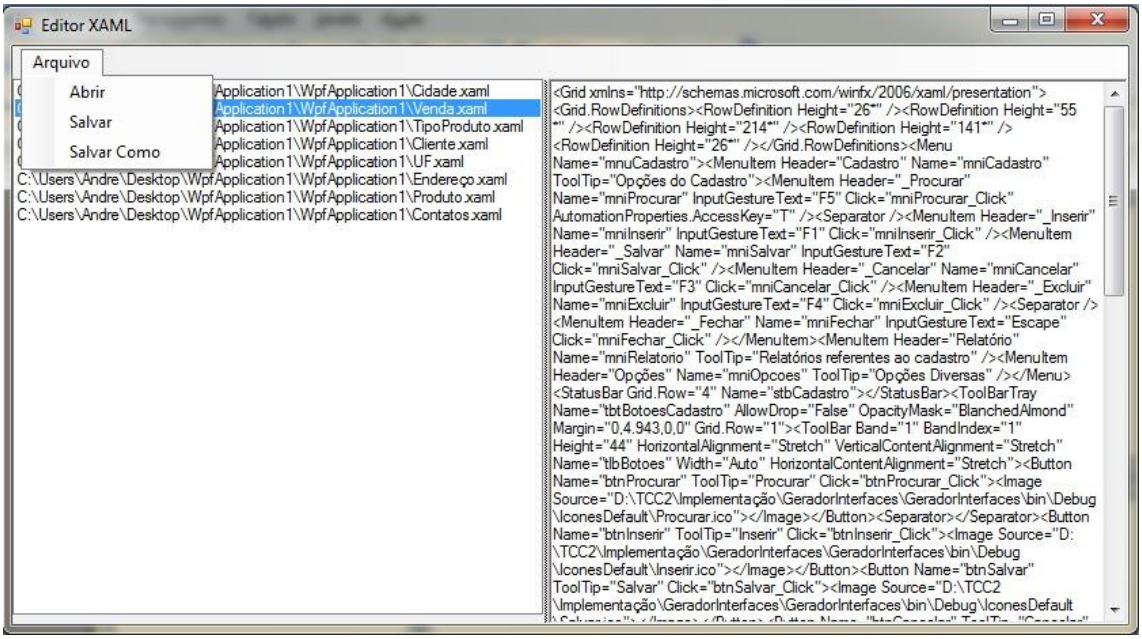



Figura 46 – Editor XAML

## 3.4 RESULTADOS E DISCUSSÃO

O aplicativo atingiu a meta esperada. O estudo de caso comprova a grande quantidade de relacionamentos atendidos e os diversos tipos de campos possíveis gerados. Houve certa dificuldade para atingir os critérios ergonômicos de usabilidade e distribuir os campos pela interface de uma forma legível. As fontes de pesquisas referentes aos assuntos WPF e XAML foram limitados para interfaces de aplicativos *standalone*. Houve a necessidade de criar-se uma camada de negócio para cada camada visual, a fim de atender os critérios ergonômicos de usabilidade.

Apesar de o MDA ser de grande importância nos aplicativos de troca de informações a sua arquitetura não foi utilizada.

A maneira que as interfaces são geradas pelo aplicativo é similar a utilizada em Pires e Siqueira (2004), porém neste aplicativo são levados em consideração alguns critérios de usabilidade. No trabalho correlato há funcionalidades para persistências dos dados, como consulta dos cadastros que difere-se do trabalho proposto, sendo extensão para trabalhos futuros. Os relacionamentos do tipo dependências são tratados pelo trabalho correlato, no aplicativo desenvolvido isto não foi tratado. A grande vantagem do aplicativo desenvolvido é que as interfaces são para uma nova tecnologia WPF, com grande futuro no mercado tecnológico.

Com base nos estudos de critérios ergonômicos de usabilidades, foi possível utilizar os seguintes critérios na construção das interfaces:

a) usabilidade – inteligibilidade:
   - a interface gerada está organizada em grupos, faz uso de identificadores que representam claramente seu significado. Ex.: títulos, ícones, etc. e informa ao usuário sobre um botão, menu, ícone ou caixa de diálogo faz ao posicionar o cursor do mouse sobre ele em balões explicativos;

b) aspectos visuais:
   - distribuição dos objetos, facilitando o entendimentos dos mesmos, campos de entrada de dados compatíveis com a necessidade, áreas de seleção dos itens de menu, identificação do formato dos campos de entrada de dados, diferencia ícones habilitados dos não habilitados com relação ao contexto, alinha campos alfa numéricos à esquerda e alinha campos numéricos à direita;

c) mensagens apresentadas:

- exibe mensagens de orientação ao usuário;

d) usabilidade – operacionalidade
   - teclas de atalho, agilizando a ação de usuários experientes;

e) prevenção de erros;
   - impossibilita a entrada de dados numéricos em campos alfanuméricos e desativação dos botões pra não exercer ações não permitidas.

No quadro 9 é apresentada a legenda da numeração utilizada nas figuras mostradas anteriormente.

| **Legenda** | |
|---|---|
| 1 | A interface gerada está organizada em grupos.<br>Distribuição dos objetos, facilitando os entendimentos dos mesmos. |
| 2 | Faz uso de identificadores que representam claramente seu significado. Ex.: títulos, ícones, etc. Informa ao usuário sobre um botão, menu, ícone ou caixa de diálogo faz ao posicionar o cursor do mouse sobre ele em balões explicativos. |
| 3 | Campos de entrada de dados compatíveis com a necessidade.<br>Identificação do formato dos campos de entrada de dados. |
| 4 | Alinha campos numéricos à direita. |
| 5 | Desativação dos botões pra não exercer ações não permitidas. |
| 6 | Alinha campos alfa numéricos à esquerda. |
| 7 | Teclas de atalho, agilizando a ação de usuários experientes. |
| 8 | Exibe mensagens de orientação ao usuário. |

Quadro 9 – Legenda

As figuras 47, 48, 49 e 50 mostram os critérios citados anteriormente.

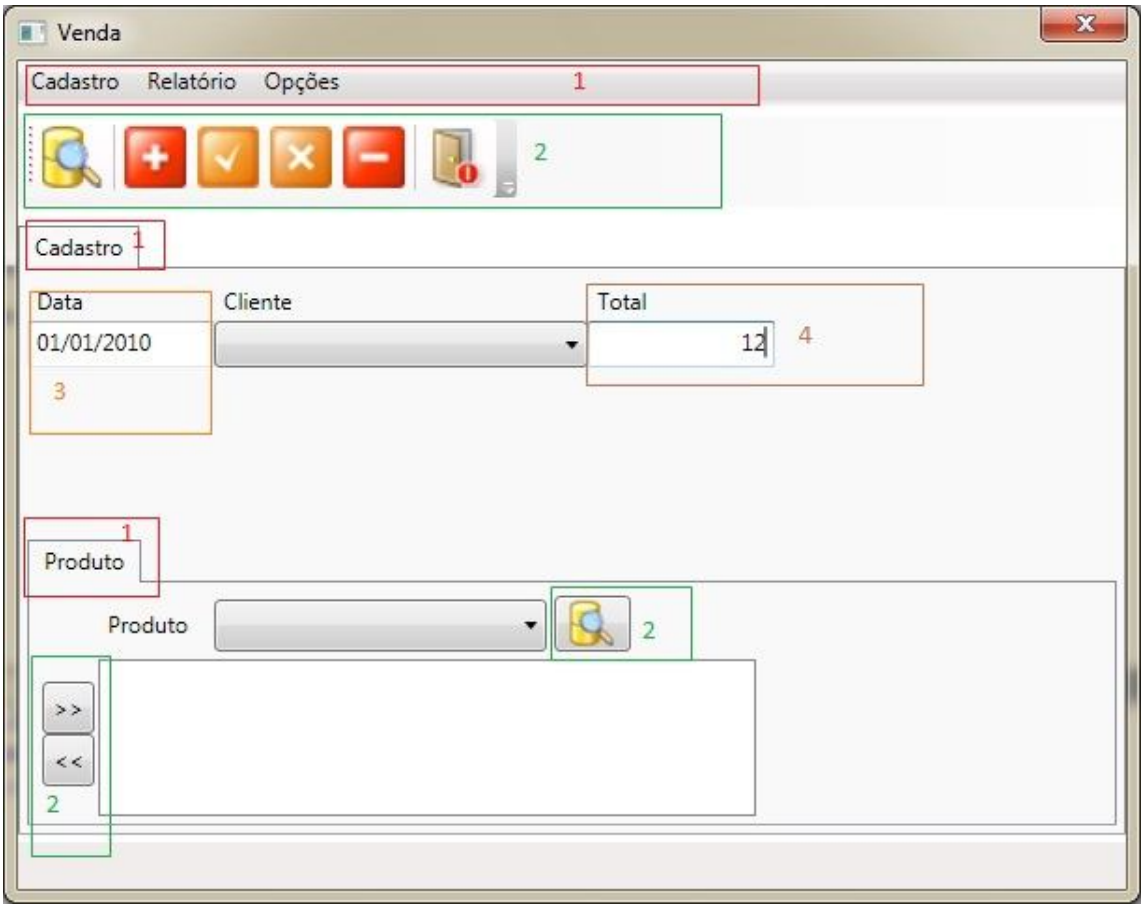



Figura 47 – Critérios ergonômicos do cadastro de venda

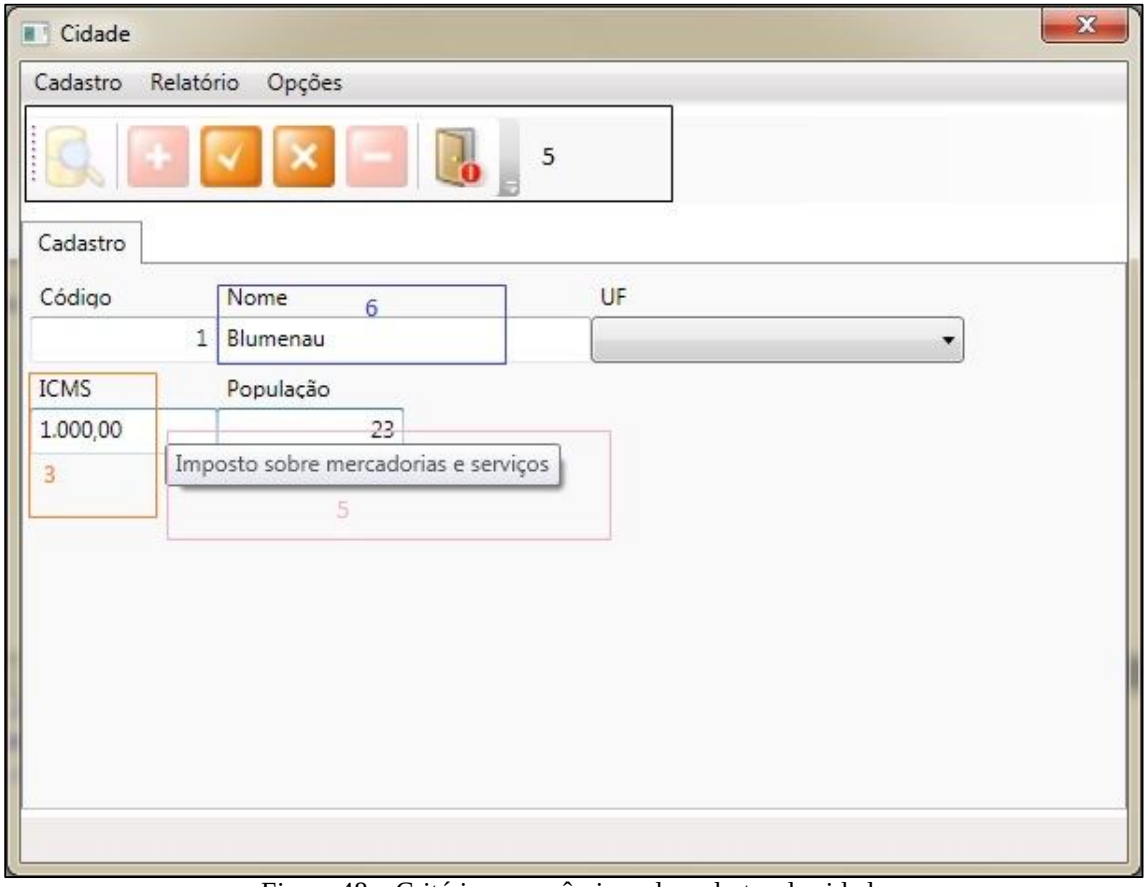



Figura 48 – Critérios ergonômicos do cadastro de cidade

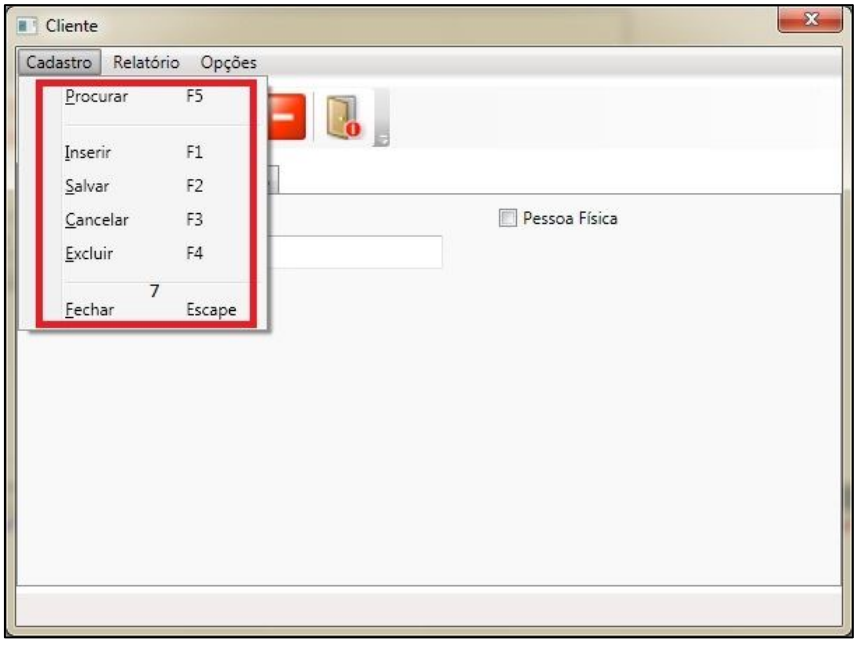



Figura 49 – Critérios ergonômicos do cadastro de cliente

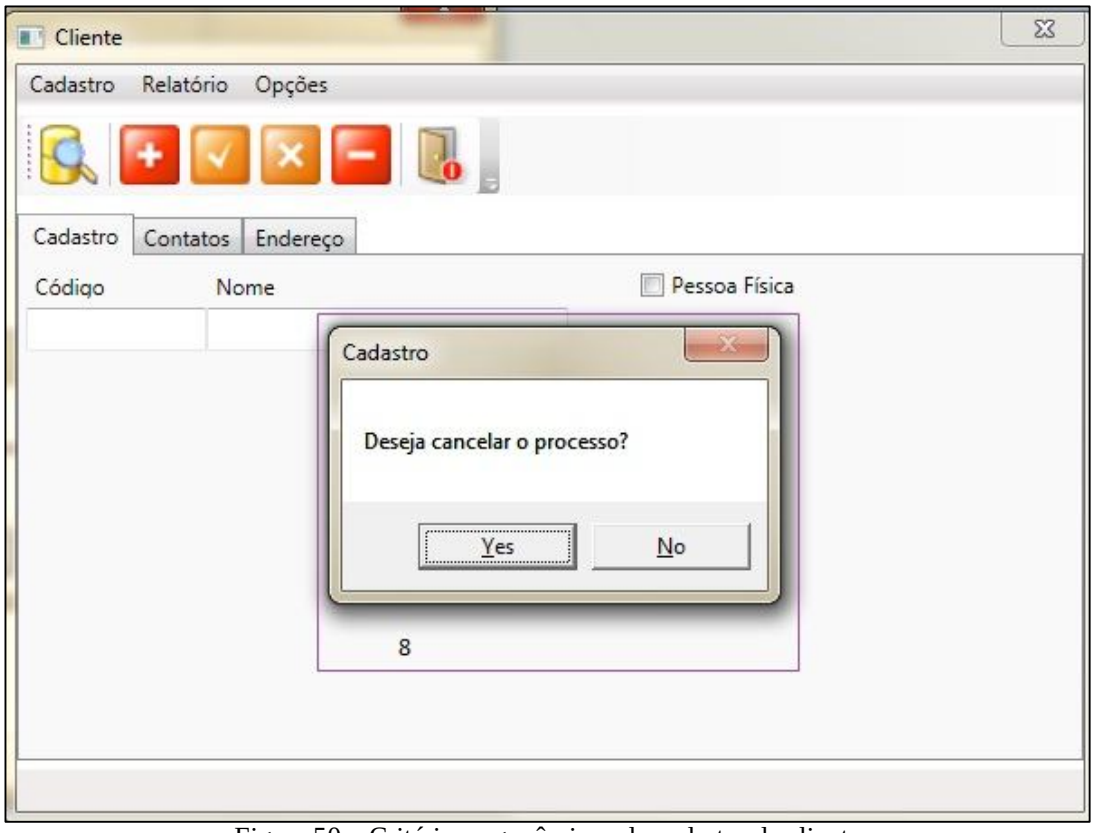



Figura 50 – Critérios ergonômicos do cadastro de cliente

O quadro 10 mostra a comparação entre o aplicativo desenvolvido e 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 |

Quadro 10 – Comparativo entre os aplicativos

# 4 CONCLUSÕES

Ao término do desenvolvimento do trabalho pode-se concluir que os resultados foram alcançados em relação aos objetivos previamente formulados. Foi implementado um aplicativo que auxilia na semi-automatização das construções de interfaces para a nova tecnologia WPF com código programático em XAML. A semi-automatização foi atingida pela possibilidade de importação do diagrama de classes modelado pela ferramenta EA.

O aplicativo construído sob a nova tecnologia WPF, poderá ser fonte de trabalhos futuros e de aprendizado na disciplina de qualidade de software, pois o mesmo usa critérios ergonômicos de usabilidade.

Através dos resultados obtidos pode-se dizer que a tecnologia WPF tem seu futuro garantido para aplicações web, devido a facilidade de programar uma interface visual e a robustez que as interfaces são geradas. O processo de geração automática de códigos torna-se cada vez mais importante para área da computação. O papel de automatização tira do desenvolvedor uma grande parte do trabalho “braçal”, tendo também certo padrão na geração das interfaces.

Verificou-se ainda a importância dos critérios ergonômicos de usabilidades adotadas nas interfaces geradas. Os critérios ergonômicos de usabilidade auxiliam o usuário, em um determinado contexto de operação, a realização de tarefas, de maneira eficaz, eficiente e agradável.

## 4.1 LIMITAÇÕES

As principais limitações da ferramenta construída são:

a) não são tratados mais de um diagrama exportados a partir do mesmo projeto criado pela ferramenta EA e fora da pasta do projeto raiz;
b) não são reconhecidos relacionamentos do tipo generalização e composição;
c) não são reconhecidos campos do tipo `long` e `short`.

## 4.2 EXTENSÕES

Como extensão para este trabalho sugere-se gerar interfaces auxiliares e para pesquisas. Definir como critério ergonômico de usabilidade, o agrupamento dos campos por área de funcionalidade. Juntamente com as interfaces, gerar uma camada de persistência para gravação, edição dos dados. Utilizar o trabalho como extensão de uma ferramenta MDA já existente.

Também sugere-se gerar o mapeamento objeto-relacional, interfaces para aplicativos web e criar métodos para automatizar ainda mais o processo da geração das interfaces. Ter a possibilidade de determinar um *template* para as interfaces e gerar uma tela principal com os menus.

# REFERÊNCIAS BIBLIOGRÁFICAS


BASSI, G. Wpf windows presentation foundation. **.NET Magazine,** Rio de Janeiro, v.1, n. 43, p. 8–14, out. 2007.

BASTIEN, C.; SCAPIN, D. 1993. **Ergonomic criteria for the evaluation of human-computer interfaces**. Technical Report 156. INRIA - Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France. França, 1993. Disponível em: <http://www.webmaestro.gouv.qc.ca/ress/Webeduc/2000nov/criteres.pdf> . Acesso em: 22 set. 2007.

BEZERRA, E. **Princípios de análise e projeto de sistemas com uml.** Rio de Janeiro: Campus, 2004.

CAETANO, Cristiano. **Ess-model**: obtenha diagramas de classes por meio de engenharia reversa. Paraná, 2003. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/2003/bb135/ess.shtml>. Acesso em: 13 nov. 2007.

CYBIS, W.; BETIOL, A.; FAUST, R. **Ergonomia e usabilidade:** conhecimento, métodos e aplicações. São Paulo: Novatec, 2007.

FARIAS, José A.; **Windows presentation foundation:** introdução ao XAML. [S.l.], 2005. Disponível em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=843>. Acesso em: 13 nov. 2007.

LIMA, Adilson S. **UML 2.0**: do requisito à solução. São Paulo: Érica, 2005.

MOSCARDINI, C. **Mini-curso**: mapeamento UML para XML. Minas Gerais, 2003. Disponível em: <http://www.inf.pucpcaldas.br/eventos/seminarios/2003_1/xml/MINICurso.doc>. Acesso em: 01 ago. 2007.

NIELSEN, J.; MOLICH, R. Heuristic evaluation of user interfaces**.** In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 8., 1990, Seattle. **Anais**... New York: ACMPress, 1990. p. 249-256. Disponível em: <http://delivery.acm.org/10.1145/100000/97281/p249-nielsen.pdf?key1=97281&key2=3513851511&coll=GUIDE&dl=GUIDE&CFID=331790&CFTOKEN=91434906>. Acesso em: 28 ago. 2007.

OBJECT MANAGEMENT GROUP. MDA guide version 1.0.1. OMG: 2003. Disponível em <http://www.omg.org/docs/omg/03-06-01.pdf>

PIRES, S. R.; SIQUERIA, I. L. **Padronização de interfaces para sistema de informação**. [S.l.], 2004. Artigo não publicado. Disponível em: <http://www.alfa.br/revista/pdf/6.pdf>. Acesso em: 01 ago. 2007.x