UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE UMA APLICAÇÃO RICA DE INTERNET PARA MONITORAMENTO DE VÍDEO ATRAVÉS DE STREAMING E SILVERLIGHT

THIAGO DA SILVA NEGHERBON

BLUMENAU
2011

2011/1-34

THIAGO DA SILVA NEGHERBON

PROTÓTIPO DE UMA APLICAÇÃO RICA DE INTERNET PARA MONITORAMENTO DE VÍDEO ATRAVÉS DE STREAMING E SILVERLIGHT

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. Roosevelt dos Santos Júnior - Orientador

BLUMENAU
2011

2011/1-34

PROTÓTIPO DE UMA APLICAÇÃO RICA DE INTERNET PARA MONITORAMENTO DE VÍDEO ATRAVÉS DE STREAMING E SILVERLIGHT

Por

THIAGO DA SILVA NEGHERBON

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

Presidente:	Prof. Roosevelt dos Santos Júnior – Orientador, FURB
Membro:	Prof. Paulo Fernando da Silva, Mestre – FURB
Membro:	Prof. Miguel Alexandre Wisintainer, Mestre – FURB

Blumenau, 04 de julho de 2011

Dedico este trabalho a toda minha família, que não mediu esforços para investir na minha educação.

AGRADECIMENTOS

À minha família, que sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Roosevelt dos Santos Júnior, por ter acreditado na conclusão deste trabalho.

RESUMO

Este trabalho demonstra o desenvolvimento de um protótipo que permite a visualização – através de uma aplicação rica de internet utilizando a tecnologia Silverlight – de câmeras conectadas a um computador a fim de monitorar ambientes em tempo real para proteção de patrimônio ou de pessoas. Foi dividido em módulos sendo uma biblioteca de classes, um serviço web e a interface em Silverlight. A interface permite selecionar uma das câmeras disponíveis e visualizar numa janela flutuante as suas imagens. É possível visualizar quantas câmeras estiverem disponíveis, além de movimentar e redimensionar as janelas e, ainda, interromper e continuar a reprodução do vídeo. A aplicação detecta automaticamente novas câmeras conectadas e atualiza a lista, disponibilizando para o usuário. Foram utilizadas as bibliotecas DirectShow.NET e WindowsMedia.NET para capturar, codificar e transmitir o vídeo. A compressão feita entrega o vídeo com uma resolução de 160 *pixels* de largura por 120 *pixels* de altura e com taxa de 15 quadros por segundo.

Palavras-chave: Silverlight. *Rich internet application*. *Streaming*. Monitoramento de imagens.

ABSTRACT

This paper demonstrates a prototype's development that allows viewing – through a rich internet application using Silverlight technology – cameras connected to a computer to real-time environments' monitoring to protect property or people. It was divided into modules which are a class library, a web service and the interface in Silverlight. The interface allows selecting one of the available cameras and visualizing in a floating window your images. It is possible to view how many cameras are available, as well as move and resize the windows and also pause and resume video playback. The application automatically detects the connected new cameras and refreshes the list, making available to the user. Used the libraries DirectShow.NET and WindowsMedia.NET to capture, encode and stream the video. Compression done delivers video with a resolution of 160 pixels wide by 120 pixels tall and with a rate of 15 frames per second.

Key-words: Silverlight. Rich internet application. Streaming. Image monitoring.

LISTA DE ILUSTRAÇÕES

Quadro 1 – *Frameworks* RIA que necessitam *plugin* ou software 16
Quadro 2 – Exemplo de arquivo XAML 19
Figura 1 – Exemplo de arquivo XAML processado 20
Figura 2 – Passos de um processo de compressão/descompressão de vídeo 22
Figura 3 – Relacionamento entre aplicação, DirectShow e componentes suportados 23
Figura 4 – Relacionamento entre grafo de captura e grafo de filtros 24
Quadro 3 – Requisitos funcionais 27
Quadro 4 – Requisitos não funcionais 28
Figura 5 – Diagrama de casos de uso 29
Quadro 5 – Caso de uso UC01 30
Quadro 6 – Caso de uso UC02 30
Quadro 7 – Caso de uso UC03 31
Quadro 8 – Caso de uso UC04 31
Quadro 9 – Caso de uso UC05 31
Quadro 10 – Caso de uso UC06 32
Quadro 11 – Caso de uso UC07 32
Figura 6 – Diagrama de classes da biblioteca `Captura` 33
Figura 7 – Diagrama de classes da aplicação `Cliente.Web` 34
Figura 8 – Diagrama de classes da aplicação `Cliente` 35
Figura 9 – Diagrama de sequência do caso de uso UC06 37
Figura 10 – *Layout* da interface do protótipo 38
Quadro 12 – Atributos e propriedades da classe `Camera` 40
Quadro 13 – Construtor da classe `Camera` 41
Quadro 14 – Construção do grafo de captura 41
Quadro 15 – Definição da compressão de vídeo 42
Quadro 16 – Configuração do *streaming* de vídeo 43
Quadro 17 – Liberação dos recursos utilizados 44
Quadro 18 – Início do processo de captura, compressão e transmissão 45
Quadro 19 – Classe de acesso às configurações da biblioteca 45
Quadro 20 – Configurações da biblioteca codificadas em XML 46
Quadro 21 – Marcação XAML do painel principal 47

Quadro 22 – Classe parcial *code-behind* da tela `MainPage` 47
Quadro 23 – Construtor do painel principal 48
Quadro 24 – Manipuladores de eventos do painel principal 49
Quadro 25 – Manipulador do evento `Click` no painel principal 50
Quadro 26 – Marcação XAML da janela flutuante 51
Quadro 27 – Classe `CameraWindow` e método `DesligarCamera` 51
Quadro 28 – Manipulador do evento `Click` do botão de ação 52
Quadro 29 – Manipulador do evento `MouseLeftButtonDown` e método auxiliar 52
Quadro 30 – Marcação onde é carregada a aplicação `Cliente` 53
Quadro 31 – Classe para armazenar informações da câmera 54
Quadro 32 – Atributos do serviço web 54
Quadro 33 – Rotina de obtenção das câmeras conectadas 55
Quadro 34 – Obtenção da próxima porta disponível 56
Quadro 35 – Obtenção e criação da lista de informações de câmeras 56
Figura 11 – Operacionalidade do caso de uso UC01 57
Figura 12 – Operacionalidade do caso de uso UC02 58
Figura 13 – Operacionalidade do caso de uso UC03 59
Figura 14 – Operacionalidade do caso de uso UC04 60
Figura 15 – Operacionalidade do caso de uso UC05 61
Quadro 36 – Atendimento dos requisitos 62
Quadro 37 – Comparativo entre ferramentas correlatas 63
Quadro 38 – Possibilidades de extensão 65

LISTA DE TABELAS

Tabela 1 – Aplicações de compressão e respectivas taxas de dados 22

LISTA DE SIGLAS

AAC – *Advanced Audio Coding*

ACM – *Audio Compression Manager*

ASF – *Advanced Systems Format*

ASP.NET – *Active Server Pages* .NET

BAML – *Binary Application Markup Language*

CFTV – Circuito Fechado de TeleVisão

COM – *Component Object Model*

DDR – *Double Data Rate*

DLL – *Dynamic-Link Library*

FPS – *Frames Per Second*

GBPS – GigaBit Por Segundo

GUI – *Graphic User Interface*

GUID – *Globally Unique Identifier*

HTML – *HyperText Markup Language*

IDE – *Integrated Development Environment*

JRE – *Java Runtime Environment*

KBPS – Kilobit Por Segundo

LCD – *Liquid Crystal Display*

MBPS – MegaBit Por Segundo

RAM – *Random Access Memory*

RIA – *Rich Internet Application*

RTP – *Real-Time Protocol*

RTSP – *Real-Time Streaming Protocol*

SDK – *Software Development Kit*

TCP – *Transmission Control Protocol*

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

USB – *Universal Serial Bus*

VCM – *Video Compression Manager*

VFW – *Video For Windows*

WCF – *Windows Communication Foundation*

WDM – *Windows Driver Model*

WMA – *Windows Media Audio*

WMV – *Windows Media Video*

WPF – *Windows Presentation Foundation*

XAML – *eXtensible Application Markup Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS DO TRABALHO 13
1.2 ESTRUTURA DO TRABALHO 14
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 RICH INTERNET APPLICATION 15
2.2 SILVERLIGHT 16
2.2.1 MediaElement 17
2.3 XAML 18
2.4 STREAMING 20
2.5 COMPRESSÃO 21
2.6 DIRECTSHOW 23
2.6.1 Filtros 24
2.6.2 Captura 24
2.7 WINDOWS MEDIA FORMAT SDK 25
2.8 TRABALHOS CORRELATOS 25
2.8.1 Protótipo (hardware e software) para segurança predial através de monitoração via câmera digital e *display* gráfico 26
2.8.2 Visualização de imagens capturadas em um Circuito Fechado de TeleVisão (CFTV) no iPhone 26
3 DESENVOLVIMENTO DO PROTÓTIPO 27
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 27
3.2 ESPECIFICAÇÃO 28
3.2.1 Diagrama de casos de uso 28
3.2.1.1 Visualizar câmera 29
3.2.1.2 Parar/continuar câmera 30
3.2.1.3 Movimentar janela da câmera 30
3.2.1.4 Fechar câmera 31
3.2.1.5 Redimensionar câmera 31
3.2.1.6 Obter câmeras 32
3.2.1.7 Atualizar câmeras 32
3.2.2 Diagrama de classes 32

3.2.2.1 Diagrama de classes da biblioteca Captura 33
3.2.2.2 Diagrama de classes da aplicação servidora 34
3.2.2.3 Diagrama de classes da aplicação Cliente 35
3.2.3 Diagrama de sequência 36
3.2.4 Layout da interface 37
3.3 IMPLEMENTAÇÃO 38
3.3.1 Técnicas e ferramentas utilizadas 39
3.3.1.1 Biblioteca Captura 39
3.3.1.2 Aplicação Cliente 46
3.3.1.3 Aplicação servidora 53
3.3.2 Operacionalidade da implementação 56
3.3.2.1 Visualizar câmera 57
3.3.2.2 Parar/continuar câmera 58
3.3.2.3 Movimentar janela da câmera 58
3.3.2.4 Fechar câmera 60
3.3.2.5 Redimensionar câmera 61
3.4 RESULTADOS E DISCUSSÃO 62
4 CONCLUSÕES 64
4.1 EXTENSÕES 65
REFERÊNCIAS BIBLIOGRÁFICAS 66

1 INTRODUÇÃO

Cubas (2005, p. 9) aponta o crescimento da violência e do crime na sociedade brasileira como causas do medo e da insegurança coletiva quanto à sua integridade física ou a de seu patrimônio. Os cidadãos evitam-se, o quanto possível, trafegar desatentos, entrar em ruas onde consideram o policiamento ineficaz ou entrar em contato com estranhos. Soma-se a isso, a percepção de que as agências responsáveis por proteger os cidadãos e seus bens se mostram cada vez mais impotentes.

Os itens criados para garantir a segurança de um estabelecimento ou de um perímetro com circulação de pessoas são desde alarmes sonoros, sensores de movimento, sensores magnéticos, até câmeras e cercas elétricas. Segundo Cubas (2005, p. 86), os itens mais populares responsáveis pela segurança são os equipamentos de observação e alarmes. Entre os equipamentos de observação estão as câmeras de alta definição, câmeras camufladas (escondidas em relógios, detectores de movimento e com microfone) e câmeras encobertas (para uma vigilância discreta).

Este trabalho terá como foco, a exploração das tecnologias utilizadas para efetuar o monitoramento utilizando câmeras. Utilizando os recursos gráficos e funcionais do Silverlight através da linguagem *eXtensible Application Markup Language* (XAML) e as imagens das câmeras, será desenvolvido um aplicativo *Rich Internet Application* (RIA) que facilitará o acesso à visualização das imagens geradas e enviadas pela internet através de *streamings* de vídeo produzidos por um servidor.

Uma das utilizações possíveis para um produto com as características deste protótipo é o monitoramento, por empresa de vigilância responsável, de áreas públicas ou privadas para garantir a segurança humana ou patrimonial. Com a interface rica e dinâmica, o protótipo oferece uma qualidade maior na interação com o usuário.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma interface RIA para facilitar o monitoramento da imagem de várias câmeras conectadas a um servidor, que será responsável por gerar o *streaming* delas e enviar através da internet para o aplicativo cliente.

Os objetivos específicos do trabalho são:

a) desenvolver uma aplicação servidora que obtenha a imagem das câmeras conectadas;
b) processar as imagens e transformá-las em *streaming* de vídeo;
c) disponibilizar o *streaming* de vídeo através da internet;
d) desenvolver uma interface RIA utilizando Silverlight com a linguagem de marcação XAML;
e) conectar a aplicação RIA cliente na aplicação servidora e obter o *streaming* de vídeo das câmeras;
f) exibir o vídeo de cada câmera em quadros individuais, permitindo ações básicas de configuração como movimentar, redimensionar, pausar, continuar e fechar cada uma.

1.2 ESTRUTURA DO TRABALHO

Para fundamentar os conceitos teóricos utilizados para o desenvolvimento deste protótipo foi escrito o capítulo 2, que está subdividido nas tecnologias empregadas. Ao final do capítulo há um resumo dos trabalhos correlatos.

No terceiro capítulo é exibido todo o detalhamento acerca do desenvolvimento do trabalho. São expostos os requisitos básicos do sistema e a especificação, bem como descritos os passos e as ferramentas utilizadas. Por fim, são colocados os resultados alcançados.

No capítulo 4, as conclusões a que se chegou com o desenvolvimento do protótipo e outras considerações são colocadas. Na sequência, no mesmo capítulo, são propostas extensões baseando-se no trabalho aqui descrito.

2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica desta monografia inicia-se, na seção 2.1, tratando das RIAs e das características que a tornam tão utilizada na internet. Na seção 2.2 é descrita a tecnologia Silverlight e seus aspectos tecnológicos. A seção 2.3 explica como arquivos XAML podem representar a interface da aplicação. Em seguida, na seção 2.4 serão colocadas as características de funcionamento da tecnologia *streaming*. Na seção 2.5 são definidos os conceitos básicos da compressão de dados. A seção 2.6 descreve a arquitetura DirectShow e como ela utiliza filtros para capturar e processar mídia. Na sequência, na seção 2.7, é apresentado o Windows Media Format *Software Development Kit* (SDK), que é utilizado neste trabalho para efetuar o *streaming* de vídeo. Por fim, na seção 2.8 são descritos os trabalhos correlatos.

2.1 RICH INTERNET APPLICATION

Segundo Busch e Koch (2009, p. 7), as aplicações web tradicionais possuem grandes limitações quanto à usabilidade e interatividade de suas interfaces. Entre as limitações estão a necessidade de recarregamento da página para obtenção de dados do servidor e comunicação síncrona. As RIAs oferecem interfaces com tempo de resposta menor, comunicação assíncrona, animações multimídia e aplicações mais robustas e ao mesmo tempo leves.

Busch e Koch (2009, p. 7) procuram definir RIA como sendo aplicações web que podem processar os dados tanto do lado cliente quanto do lado servidor. Possuem interfaces similares à experiência de aplicações *desktop*, podem funcionar tanto *online* quanto *offline* e possuem uma variedade de controles interativos.

Aplicações ricas, por poderem processar dados do lado cliente, têm a vantagem de apenas se comunicar com o servidor no momento que algum dado exclusivo daquele for necessário, ou quando a consistência destes dados se tornar imprescindível. Além do mais, a aplicação cliente pode efetuar todas as validações necessárias dos dados antes de qualquer início de comunicação (BUSCH; KOCH, 2009, p. 8).

Segundo Busch e Koch (2009, p. 12), há tecnologias que implementam RIA utilizando arquiteturas nativas, como JavaScript, *HyperText Markup Language* (HTML) e *eXtensible*

Markup Language (XML), e outras que dependem de *plugins* próprios instalados no computador cliente para que as aplicações sejam executadas, como o Flash e o próprio Silverlight.

Alguns problemas devem ser considerados ao decidir implementar uma aplicação rica de internet. Busch e Koch (2009, p. 12) citam a possibilidade de estes *plugins* indispensáveis para a aplicação nem sequer estarem instalados no cliente ou, no caso de dependência do JavaScript, estarem desabilitados no navegador. Muitas das vezes isto pode ocorrer por questões de segurança em algumas empresas.

No Quadro 1 são mostrados alguns *frameworks* para desenvolvimento de aplicações RIA que necessitam instalação de *plugin* ou software para execução, bem como alguns detalhes que os diferenciam entre si.

Nome	**Desenvolvedor**	**Linguagem**	**Ambiente de execução**	**Outras características**
Java Applets	Sun Microsystems	Java	*Java Runtime Environment* (JRE)	Executa normalmente com um navegador.
JavaFX	Sun Microsystems	JavaFX Script	JavaFX Runtime (possui JRE)	Executa em navegadores, no *desktop* ou em telefones celulares.
Adobe Flash	Adobe Systems	ActionScript	Flash Player Plugin	Foco em áudio, vídeo e animações.
Adobe Flex	Adobe Systems	MXML, ActionScript	Flash Player Plugin	Ambiente de execução "*Flex builder*", suporta arrastar e soltar para criar *Graphic User Interfaces* (GUIs).
Microsoft Silverlight	Microsoft	Linguagens .NET; linguagens em script; XAML	Silverlight Plugin	Acesso *offline*. Inclui um tocador de mídia. Similar ao *Windows Presentation Foundation* (WPF).

Fonte: adaptado de Busch e Koch (2009, p. 15).

Quadro 1 – *Frameworks* RIA que necessitam *plugin* ou software

2.2 SILVERLIGHT

Segundo Microsoft (2009), o Silverlight é uma plataforma para desenvolvimento de aplicações ricas e com o objetivo de criar experiências de usuário interativas e atraentes, tanto para a internet quanto para estações de trabalho, dispositivos móveis e estando *online* ou *offline*. Ainda possui a capacidade de exibir efeitos 3D e, para dar maior ênfase à tecnologia

RIA, possui dezenas de controles para exibir gráficos, mídia e layout com temas profissionais.

Ghoda e Scanlon (2009, p. 1) completam que no núcleo do *framework* de apresentação do Silverlight está o XAML, que permite aos desenhistas e desenvolvedores definirem GUIs desacopladas e exteriorizadas com suas respectivas folhas de estilos. Sendo assim, o Silverlight é uma extensão natural de tecnologias já existentes, especificamente o .NET *Framework* e o WPF.

Para suportar as imagens provenientes de *streaming*, Ghoda e Scanlon (2009, p. 198) citam a capacidade reforçada de gerenciamento de mídia que o Silverlight possui. Tal qualidade faz com que a tecnologia suporte *streamings* seguros e de alta qualidade. Há também o suporte para alguns dos novos formatos de compressão de mídia tais como H.264, *Advanced Audio Coding* (AAC) e MP4.

É possível desenvolver para diversas plataformas. Segundo Microsoft (2011a) e Microsoft (2011b), além dos navegadores, *smartphones* com Windows Phone 7 e Symbian também executam aplicativos Silverlight. Para quem já desenvolveu utilizando a tecnologia *Active Server Pages* .NET (ASP.NET) vai considerar familiar os elementos que formam o *layout*.

2.2.1 MediaElement

Um dos controles mais utilizados durante o desenvolvimento deste protótipo, e o que mais evidencia a tecnologia RIA é o `MediaElement`. Trata-se de uma região retangular capaz de tocar áudio e vídeo. Sua propriedade `Source` pode receber tanto arquivos locais como *Uniform Resourse Locators* (URLs) de *streamings* (MICROSOFT DEVELOPER NETWORK, 2011b).

Segundo Ghoda e Scanlon (2009, p. 202), podem haver alterações de estado neste componente. Quando o *streaming* é iniciado e não há dados suficientes para a exibição da imagem, o estado é alterado para *buffering* até que o vídeo seja carregado suficientemente para o estado alterar para *playing*. Isso é gerenciado de forma automática pelo componente, mas o usuário também pode intervir na exibição, e para isto existem os métodos `Play` e `Stop`, que podem ser chamados por eventos de botões.

2.3 XAML

MacDonald (2010, p. 23) define XAML como sendo uma linguagem de marcação utilizada para instanciar objetos do .NET *Framework*. Embora o XAML seja uma tecnologia que pode ser aplicada para diversos tipos de problemas, a sua tarefa principal é construir interfaces WPF. Os arquivos XAML podem definir a posição dos botões, painéis e controles que formam as janelas em aplicações WPF.

Apesar de simples de construir, as interfaces em XAML possuem certa complexidade na medida em que é possível programá-las personalizando cada detalhe. MacDonald (2010, p. 26) coloca que se podem programar manipuladores de eventos, expressões vinculadoras de dados, definir os recursos utilizados na tela, definir animações e modelos para cada controle.

Arquivos XAML, quando vistos em modo texto, são arquivos XML que, geralmente, têm a extensão `.xaml`. Pelo fato de a estrutura XML ter sido desenhada para ser legível e fácil de ser entendida, ela não é compacta. O WPF trata este problema com o *Binary Application Markup Language* (BAML), que não é nada mais que uma representação binária do XML (MACDONALD, 2010, p. 26).

MacDonald (2010, p. 26) completa que, ao compilar a aplicação, os arquivos XML são convertidos para BAML e então embarcados como recurso num *Dynamic-Link Library* (DLL) ou num arquivo executável. Durante o processo ocorre uma troca de *tokens* compridos por *tokens* mais curtos contendo o mesmo significado, e por isso são significantemente menores e também otimizados para a leitura em tempo de execução.

No arquivo XAML, cada elemento presente é mapeado para a instância de uma classe do .NET. Os elementos podem ser aninhados dentro de outros e também é possível definir propriedades das classes através de atributos. Para fazer a ligação entre os manipuladores de eventos e também para separar marcação declarativa da lógica de execução, são utilizados os arquivos *code-behind*[1] (MACDONALD, 2010, p. 27, 30).

No Quadro 2 encontra-se um exemplo da codificação de um arquivo XAML e na Figura 1 está o mesmo arquivo já processado.

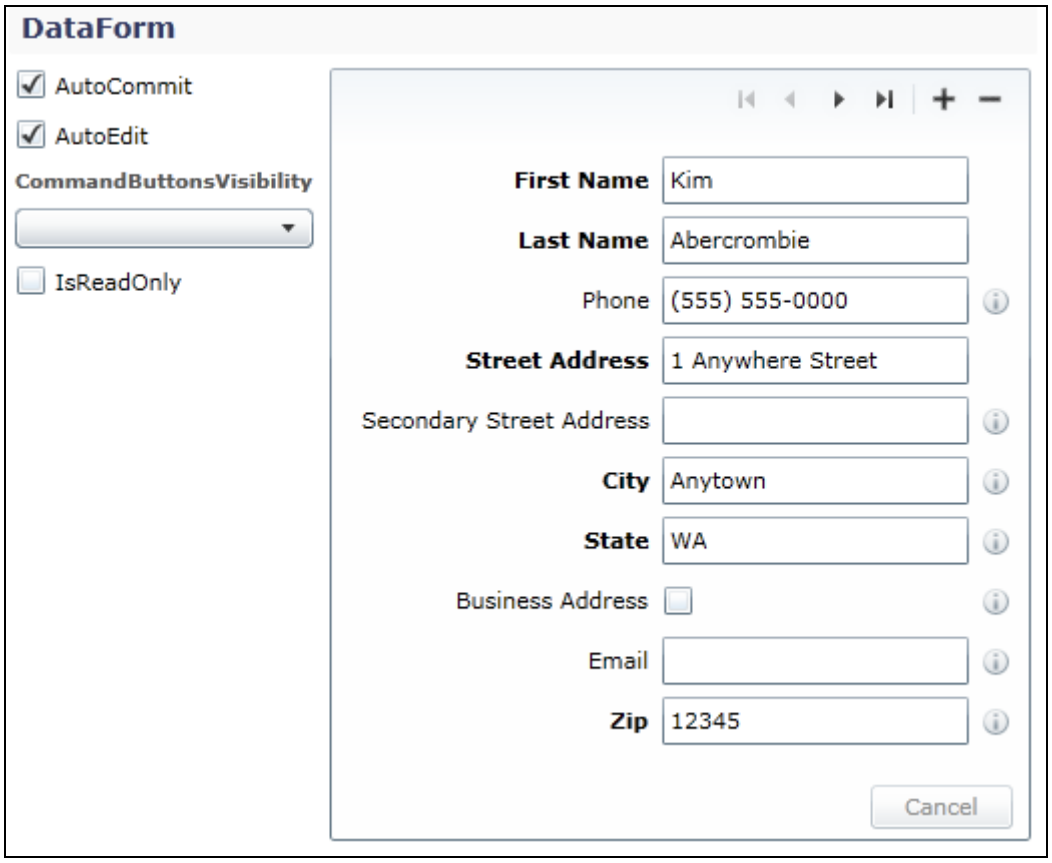
---

[1] Segundo Microsoft Developer Network (2011a), *code-behind* é um termo utilizado para definir o código que é unido ao código gerado pelo processador XAML quando a página é compilada numa aplicação.

```
<UserControl x:Class="System.Windows.Controls.Samples.DataFormSample"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:dataform="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.DataForm.
Toolkit"
    xmlns:input="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Input.Toolkit"
    xmlns:sys="clr-namespace:System;assembly=mscorlib">
    <StackPanel>
        <ContentControl Content="DataForm" Style="{StaticResource Header}" />
        <StackPanel>
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition />
                </Grid.ColumnDefinitions>
                <StackPanel Grid.Column="0">
                    <CheckBox Content="AutoCommit" IsChecked="{Binding
AutoCommit, ElementName=dataForm, Mode=TwoWay}" Margin="4" />
                    <CheckBox Content="AutoEdit" IsChecked="{Binding AutoEdit,
ElementName=dataForm, Mode=TwoWay}" Margin="4" />
                    <TextBlock Text="CommandButtonsVisibility"
Style="{StaticResource ApiName}" Margin="4" />
                    <ComboBox SelectedItem="{Binding CommandButtonsVisibility,
ElementName=dataForm, Mode=TwoWay}" SelectedIndex="0" Margin="4">
                        <sys:String>All</sys:String>
                        <sys:String>Add</sys:String>
                        <sys:String>Delete</sys:String>
                        <sys:String>Edit</sys:String>
                        <sys:String>Navigation</sys:String>
                        <sys:String>None</sys:String>
                    </ComboBox>
                    <CheckBox Content="IsReadOnly" IsChecked="{Binding
IsReadOnly, ElementName=dataForm, Mode=TwoWay}" Margin="4" />
                </StackPanel>
                <dataform:DataForm x:Name="dataForm" Width="350"
ItemsSource="{Binding}" HorizontalAlignment="Left" MaxWidth="500" Margin="4"
Grid.Column="1" />
            </Grid>
        </StackPanel>
    </StackPanel>
</UserControl>
```

Fonte: Microsoft (2010).

Quadro 2 – Exemplo de arquivo XAML



Fonte: Microsoft (2010).

Figura 1 – Exemplo de arquivo XAML processado

2.4 STREAMING

Segundo Follansbee (2006, p. 16), o processo de funcionamento do *streaming* possui basicamente quatro passos: capturar as imagens que se deseja enviar, codificar as imagens, distribuir as imagens e, por fim, exibir as imagens no cliente. Segundo Austerberry (2005, p. 7), codificação de imagens é a conversão do sinal de vídeo em arquivo de *streaming*.

Austerberry (2005, p. 140) afirma que não há uma forma padrão para codificar mídia. A forma com que as imagens serão trabalhadas vão depender da plataforma de hardware escolhida e da qualidade final de exibição desejada. Uma das variáveis na forma de realizar a codificação é o *codec*[2], que são algoritmos responsáveis por compactar os dados multimídia.

O *Streaming* é uma tecnologia que utiliza *Real-Time Protocol* (RTP) para transferir dados de conteúdo multimídia. Este protocolo possui campos extras nos seus datagramas que

[2] Segundo Austerberry (2005), *codec* é uma sigla para compressão/descompressão.

representam uma facilidade maior que o *Transmission Control Protocol* (TCP) no que diz respeito à taxa de transmissão e controles que garantam melhor qualidade na exibição (AUSTERBERRY, 2005, p. 18).

Para a transmissão de conteúdo ao vivo, como é o caso deste trabalho, Austerberry (2005, p. 20) recomenda que seja utilizado o *Real-Time Streaming Protocol* (RTSP). As características deste protocolo são a possibilidade de utilizar comandos como *play* e *pause* e também a capacidade de anunciar novos *streamings* na transmissão, no caso deste trabalho, novas câmeras conectadas.

2.5 COMPRESSÃO

Para Bhaskaran e Konstantinides (1997, p. 1), os métodos de compressão são extremamente vitais para aplicações que manipulam ou armazenam dados. O objetivo principal é produzir uma representação digital compacta dos sinais recebidos. Quando trata-se de fluxo de vídeo ou sinal de áudio a compressão deve minimizar a taxa de bits desta representação digital.

Um dos fatores utilizados para os esquemas de compressão serem aplicados é a redundância no sinal. Num único quadro pode haver uma correlação significante entre amostras vizinhas, o que é chamado de correlação espacial. Para imagens coletadas de várias câmeras pode haver correlação entre as amostras simultâneas das câmeras, o que é chamado correlação espectral. Por fim, pode haver correlação entre amostras em diferentes segmentos de tempo, o que é chamado correlação temporal (BHASKARAN; KONSTANTINIDES, 1997, p. 3).

Bhaskaran e Konstantinides (1997, p. 5) destacam que os métodos de compressão podem ter diversas prioridades no seu propósito. Entre elas estão a qualidade do sinal, a complexidade de implementação do *codec* e o atraso na comunicação. Por exemplo, num sistema de teleconferência, a prioridade pode ser o atraso na comunicação, ao contrário de um sistema de televisão, onde a prioridade pode ser a qualidade do sinal e a facilidade na implementação.

Outra característica que deve ser analisada na implementação de um *codec* é se o conteúdo codificado pode ou não apresentar perda de dados. Ou seja, se o processo de codificação necessita ser totalmente reversível, sem perda alguma de dados, então chama-se

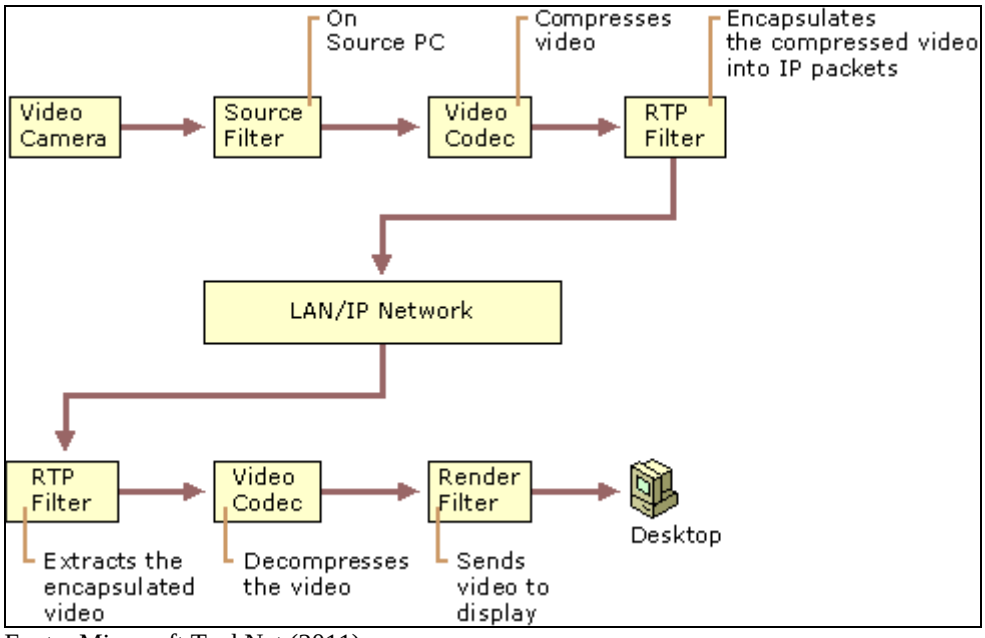
compressão sem perdas. Caso alguma perda seja tolerável, com ganho de velocidade na codificação, o processo chama-se compressão com perdas (BHASKARAN; KONSTANTINIDES, 1997, p. 6).

Na Tabela 1 são apresentadas algumas aplicações e suas respectivas taxas de dados. Na sequência, a Figura 2 mostra os passos de um processo de compressão e descompressão de vídeo.

Tabela 1 – Aplicações de compressão e respectivas taxas de dados

Aplicação	**Taxa de dados**	
	Descomprimido	**Comprimido**
Voz 8k amostras/s, 8 *bits*/amostra	64 KiloBit Por Segundo (KBPS)	2 – 4 KBPS
Vídeo conferência (15 *Frames Per Second* (FPS)) tamanho do quadro 352 x 240, 24 *bits*/*pixel*	30,41 MegaBit Por Segundo (MBPS)	64 – 768 KBPS
Transmissão de vídeo (30 FPS) tamanho do quadro 720 x 480, 24 *bits*/*pixel*	248,83 MBPS	3 – 8 MBPS
Televisão de alta definição (59,94 FPS) tamanho do quadro 1280 x 720, 24 *bits*/*pixel*	1,33 GigaBit Por Segundo (GBPS)	20 MBPS

Fonte: adaptado de Bhaskaran e Konstantinides (1997, p. 3).



Fonte: Microsoft TechNet (2011).

Figura 2 – Passos de um processo de compressão/descompressão de vídeo

Como demonstrado na Figura 2, o vídeo obtido da câmera é capturado por um filtro e depois é comprimido pelo *codec*. Depois os dados são empacotados para a camada de transporte e transmitidos até o cliente onde são desempacotados, as imagens são descomprimidas e por fim executadas.

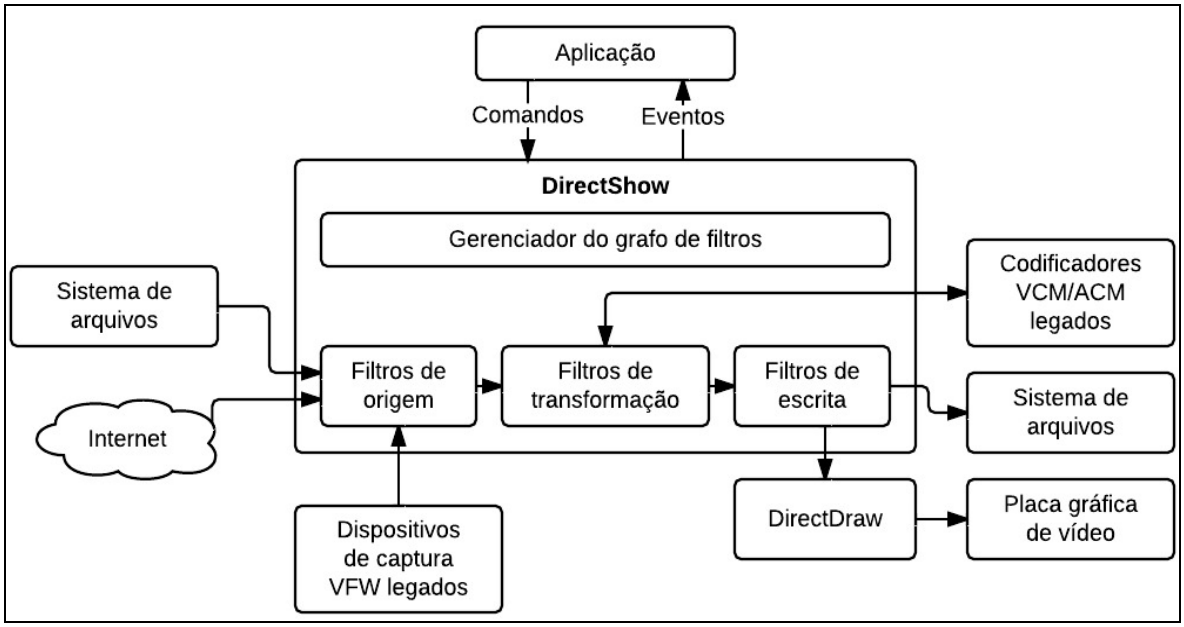
2.6 DIRECTSHOW

O Microsoft DirectShow é uma arquitetura de mídia que suporta vídeo de alta qualidade e execução de *streamings* multimídia. Ele suporta uma ampla variedade de formatos de áudio e vídeo, captura de dispositivos analógicos e digitais e é baseado no COM (MICROSOFT DEVELOPER NETWORK, 2010).

Segundo Microsoft Developer Network (2010), o principal objetivo do DirectShow é simplificar a tarefa de criar aplicações de mídia digital isolando-as da complexidade da camada de transporte de dados, diferenças nos hardwares e da sincronização. Para lidar com a variedade de recursos, dispositivos e formatos possíveis, a tecnologia utiliza uma arquitetura modular, na qual a aplicação combina de diferentes maneiras componentes chamados de filtros.

Neste trabalho é utilizada a biblioteca de classes DirectShow.NET que permite acessar as funcionalidades da arquitetura em aplicações .NET. São na grande maioria enumerações, estruturas e interfaces para objetos COM do DirectShow (DIRECTSHOW.NET, 2010).

A Figura 3 mostra o relacionamento entre uma aplicação, os componentes do DirectShow e alguns componentes de hardware e software que a tecnologia suporta.



Fonte: adaptado de Microsoft Developer Network (2010).

Figura 3 – Relacionamento entre aplicação, DirectShow e componentes suportados

Neste protótipo são utilizados apenas alguns destes componentes. A captura é feita por um dispositivo *Video For Windows* (VFW) suportado pela arquitetura DirectShow e os dados são a entrada para um filtro de origem. Na sequência um filtro de transformação conectado ao

filtro antes citado recebe os dados das imagens da câmera e codifica utilizando um dos codificadores legados. Por fim, o filtro responsável pela escrita grava os dados resultantes deste processo em arquivo.

2.6.1 Filtros

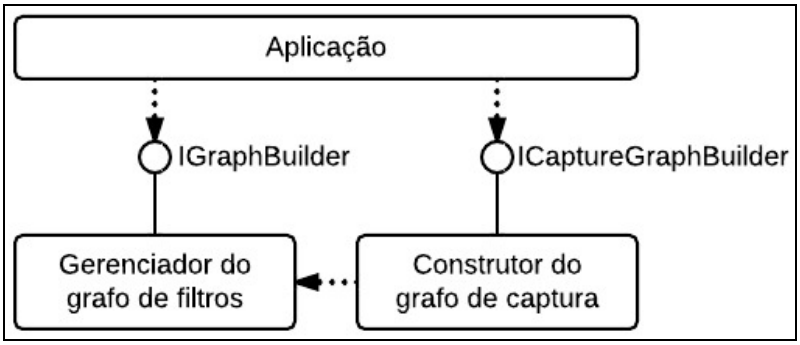
A respeito dos filtros, Microsoft Developer Network (2010) cita que há suporte para captura e sintonização de dispositivos baseados no *Windows Driver Model* (WDM), assim como suporte às placas gráficas legadas e codificadores escritos para o *Audio Compression Manager* (ACM) e o *Video Compression Manager* (VCM).

Os filtros recebem uma entrada e produzem uma saída. Assim sendo, uma aplicação utilizando DirectShow efetua qualquer tarefa conectando filtros, formando uma série, de forma que a saída de um filtro torna-se a entrada para outro. Esta série é chamada grafo de filtros (MICROSOFT DEVELOPER NETWORK, 2010).

2.6.2 Captura

Segundo Microsoft Developer Network (2010), um grafo de filtros que efetua captura é chamado grafo de captura. Para representar esta necessidade, o DirectShow possui um objeto chamado `CaptureGraphBuilder`, que por sua vez implementa a interface `ICaptureGraphBuilder2`, contendo métodos para montar e controlar o grafo de captura.

Na Figura 4 é mostrado o relacionamento que o grafo de captura possui com o grafo de filtros.



Fonte: adaptado de Microsoft Developer Network (2010).

Figura 4 – Relacionamento entre grafo de captura e grafo de filtros

A aplicação acessa as funcionalidades do construtor do grafo de captura através de

uma interface, assim como as funcionalidades do gerenciador do grafo de filtros. Para que o construtor do grafo de captura possa agir em cima do grafo de filtros, ele precisa estar conectado ao gerenciador.

2.7 WINDOWS MEDIA FORMAT SDK

O Windows Media Format SDK oferece aos desenvolvedores o acesso a diversos componentes do Windows Media Format. Isto inclui o contêiner de arquivos *Advanced Systems Format* (ASF), codificadores *Windows Media Audio* (WMA) e *Windows Media Video* (WMV) e capacidades básicas de *streaming* sobre a rede. Os objetos deste SDK manipulam componentes do Windows Media em baixo nível (MICROSOFT DEVELOPER NETWORK, 2009).

Segundo Microsoft Developer Network (2009), o propósito primário do SDK é permitir que sejam criadas aplicações que reproduza, escreva, criptografe e envie arquivos ASF por *streams* de rede. Estes arquivos podem conter áudio e vídeo codificados em WMA e WMV. No entanto, podem também ser encapsulados quaisquer tipos de dados.

Da mesma forma que com o DirectShow.NET, foi utilizado neste trabalho uma biblioteca de classes para acessar as funcionalidades do Windows Media Format SDK em aplicações .NET chamada WindowsMedia.NET.

2.8 TRABALHOS CORRELATOS

Após a invenção da internet, a tecnologia de circuitos fechados de televisão evoluiu, permitindo a mesma utilização das câmeras, mas desta vez através da rede mundial. Diversos produtos foram desenvolvidos para explorar esta nova fronteira na área da tecnologia, entre eles estão *WebCam Monitor* (DESKSHARE INCORPORATED, 2011) e Webcam 1-2-3 (WEBCAM CORPORATION, 2011).

Assim como este, outros trabalhos acadêmicos focaram no monitoramento de imagem de câmeras, porém utilizando outras abordagens. Merege Neto (2004) construiu um protótipo para segurança utilizando câmera e *display* gráfico. Já Carlassara (2009) desenvolveu uma

ferramenta para visualizar vídeos no iPhone gravados por uma aplicação executando em um computador que possui uma câmera conectada.

2.8.1 Protótipo (hardware e software) para segurança predial através de monitoração via câmera digital e *display* gráfico

Merege Neto (2004) aborda a segurança de edifícios utilizando câmeras digitais e *Liquid Crystal Display* (LCD). As imagens são capturadas pela câmera posicionada em ponto pré-definido e, através de interface serial, é conectada com um *display* gráfico LCD.

O desenvolvimento foi dividido em módulos. Sendo assim, há o módulo `Câmera` e o módulo `Monitor`. O módulo `Câmera` constitui hardware e software. O software é escrito na linguagem de programação PicBasic. Já o módulo `Monitor` foi desenvolvido utilizando-se o hardware LCD. O software é escrito na linguagem de programação C (MEREGE NETO, 2004).

O sistema desenvolvido por Merege Neto (2004) funciona da seguinte maneira: a câmera captura cada *frame* de imagem do ambiente onde está localizada e grava este *frame* na memória. O *display* LCD lê este *frame*, converte para ser exibido e, somente após a leitura estiver completa, a câmera grava outro *frame* para ser lido.

2.8.2 Visualização de imagens capturadas em um Circuito Fechado de TeleVisão (CFTV) no iPhone

O trabalho desenvolvido por Carlassara (2009) trata do monitoramento de imagens gerados por câmeras posicionadas em determinados locais e visualizadas num aplicativo do iPhone. A imagem de cada câmera é obtida por um software, chamado `Gerador`, que grava em arquivo e os disponibiliza através da internet. O `Gerador` também pode obter uma foto em tempo real da câmera conectada ao computador e enviá-la ao aplicativo cliente.

O aplicativo cliente que Carlassara (2009) desenvolveu é executado no iPhone. Nele, o usuário pode cadastrar os locais que possuem câmeras em monitoramento e mais tarde acessá-los. Ao acessar este local cadastrado, são exibidos os vídeos já gravados naquele local e o usuário pode escolher qual deseja assistir. A transmissão do vídeo é feita através de *streaming*.

3 DESENVOLVIMENTO DO PROTÓTIPO

Para este trabalho foi desenvolvido um protótipo para demonstrar a utilização da tecnologia RIA em aplicações utilizando *streaming* de vídeo. A finalidade deste protótipo é visualizar a imagem de câmeras, tantas quanto estiverem conectadas ao servidor, para monitorar espaços públicos ou privados com o objetivo de proteger pessoas ou patrimônio.

O protótipo foi desenvolvido com a estrutura cliente/servidor, onde os clientes são uma interface rica para a internet e o servidor é um serviço web executando em um computador que possui as câmeras conectadas.

A seguir serão descritos os detalhes desde a especificação até a implementação destes componentes do protótipo.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Durante a elaboração da proposta deste trabalho, foram levantados os requisitos básicos que deveriam ser desenvolvidos. Os requisitos funcionais são descritos no Quadro 3, bem como os casos de uso relacionados, e os requisitos não funcionais no Quadro 4.

Requisitos funcionais	**Casos de uso**
RF01: A aplicação cliente deve disponibilizar um painel que ocupe o maior espaço possível da janela do navegador onde o usuário poderá interagir.	UC03
RF02: A aplicação cliente deve disponibilizar um quadro para cada *streaming* de câmera que estiver recebendo.	UC01
RF03: A aplicação cliente deve possuir um menu onde podem ser escolhidas as câmeras escondidas que se deseja restaurar.	UC01, UC07
RF04: A aplicação cliente deve permitir redimensionamento e movimentação livres (limitado pelo tamanho do painel) de cada quadro de câmera.	UC03 e UC05
RF05: A aplicação cliente deve permitir fechar cada quadro de câmera.	UC04
RF06: A aplicação cliente deve permitir parar e continuar a exibição das imagens de cada janela de câmera.	UC02
RF07: A aplicação servidora deve ler a entrada de cada câmera conectada ao computador.	UC06
RF08: A aplicação servidora deve processar os dados e criar o *streaming* de vídeo de cada câmera.	UC06
RF09: A aplicação servidora deve disponibilizar os *streamings* para leitura na internet.	UC06

Quadro 3 – Requisitos funcionais

Requisitos não funcionais
RNF01: Desenvolver a aplicação cliente utilizando a linguagem de programação C#.
RNF02: Desenvolver a interface da aplicação cliente utilizando o ambiente Microsoft Expression Blend.
RNF03: Implementar o código da aplicação cliente utilizando o ambiente Microsoft Visual Studio.
RNF04: Desenvolver a aplicação servidora utilizando a linguagem de programação C#.
RNF05: Implementar o código da aplicação servidora utilizando o ambiente Microsoft Visual Studio.

Quadro 4 – Requisitos não funcionais

3.2 ESPECIFICAÇÃO

A especificação dos módulos deste trabalho foi desenvolvida na ferramenta de modelagem Enterprise Architect utilizando a notação *Unified Modeling Language* (UML). Foram então criados os diagramas de caso de uso, de classes e de sequência, que são demonstrados nas seções a seguir e, na seção 3.2.4 é apresentado o rascunho do leiaute deste protótipo.

3.2.1 Diagrama de casos de uso

O diagrama de casos de uso do protótipo desenvolvido mostra os atores e as interações deles com a aplicação. A seguir, é exibida a Figura 5 contendo o diagrama de casos de uso.

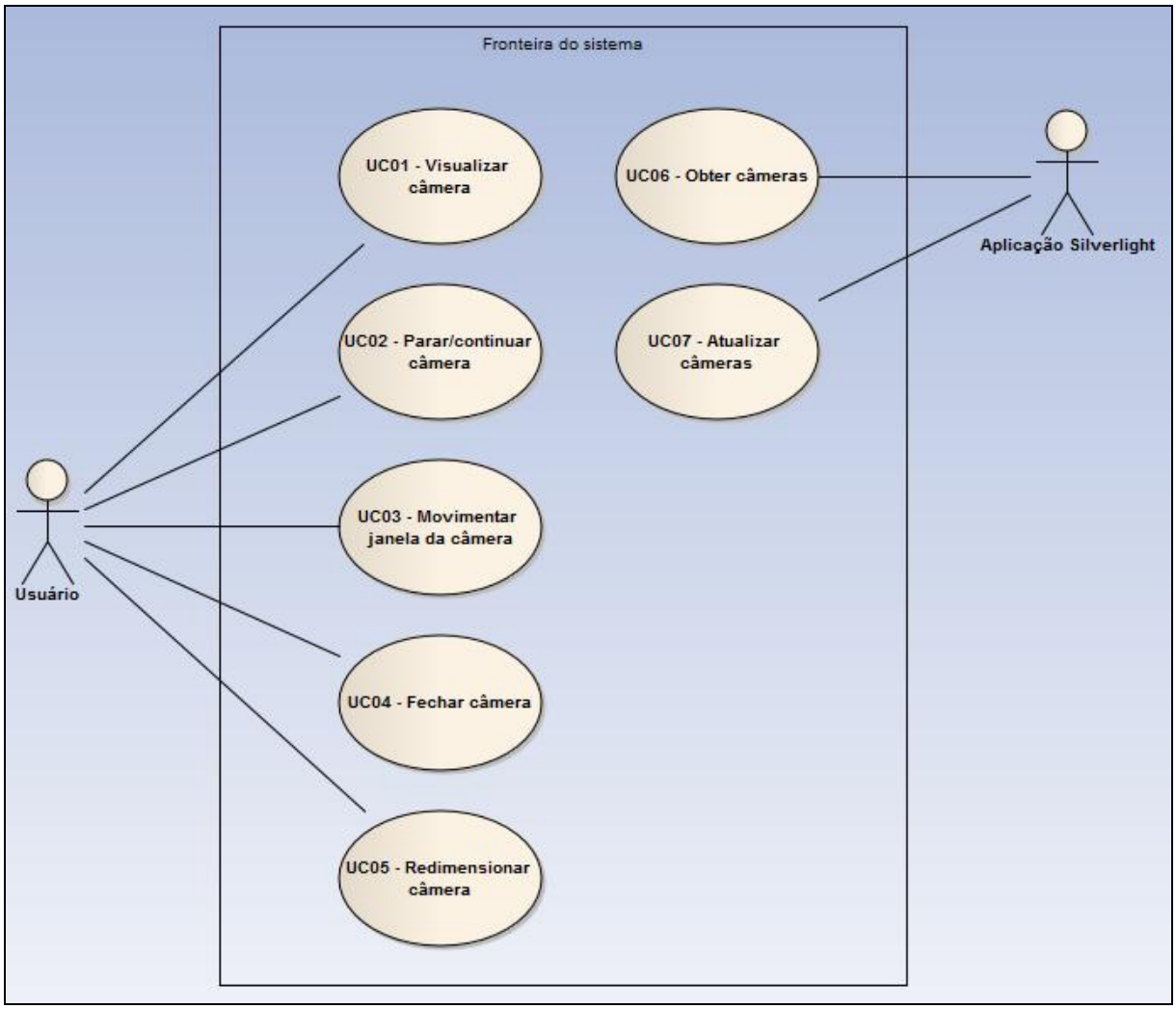


Figura 5 – Diagrama de casos de uso

3.2.1.1 Visualizar câmera

O caso de uso `UC01 – Visualizar câmera` descreve o que o usuário precisa fazer para que possa visualizar as imagens de uma câmera na aplicação `Cliente`. O Quadro 5 mostra os detalhes do caso de uso.

UC01 – Visualizar câmera	
Requisitos atendidos	RF02 e RF03
Pré-condições	Não possui.
Cenário principal	1) O usuário inicia através de seu navegador a aplicação `Cliente`. 2) O usuário seleciona uma câmera na lista de câmeras disponíveis. 3) O usuário clica no botão `Abrir`. 4) A aplicação `Cliente` exclui a câmera selecionada da lista de câmeras disponíveis. 5) A aplicação `Cliente` inicia a janela de exibição da câmera.
Fluxo alternativo 01	Nenhuma câmera disponível: 1) No passo 2 do cenário principal, caso não haja câmera disponível, o fluxo é encerrado.
Pós-condições	Uma janela é aberta exibindo a imagem da câmera.

Quadro 5 – Caso de uso UC01

3.2.1.2 Parar/continuar câmera

O caso de uso `UC02 – Parar/continuar câmera` descreve os passos seguidos para parar ou continuar a execução da imagem de uma câmera na aplicação `Cliente`. No Quadro 6 são exibidos os detalhes deste caso de uso.

UC02 – Parar/continuar câmera	
Requisitos atendidos	RF06
Pré-condições	Deve haver alguma janela de câmera aberta.
Cenário principal	1) O usuário clica no botão `Parar`. 2) A aplicação `Cliente` congela a imagem da câmera.
Fluxo alternativo 01	Continuar exibição: 1) No passo 1, caso a imagem já estivesse congelada, o usuário clica no botão `Continuar`. 2) A aplicação `Cliente` recarrega a imagem da câmera e a reproduz.
Pós-condições	1) Caso a imagem estivesse reproduzindo, deve ficar parada. 2) Caso a imagem estivesse parada, deve reproduzir.

Quadro 6 – Caso de uso UC02

3.2.1.3 Movimentar janela da câmera

O caso de uso `UC03 – Movimentar janela da câmera` descreve o fluxo seguido para que seja movimentada a janela de uma câmera na aplicação `Cliente`. No Quadro 7 descrevem-se os detalhes do caso de uso.

UC03 – Movimentar janela da câmera	
Requisitos atendidos	RF01 e RF04
Pré-condições	Deve haver alguma janela de câmera aberta.
Cenário principal	1) O usuário clica no cabeçalho da janela. 2) O usuário arrasta a janela para o local desejado.
Pós-condições	A janela deve permanecer no local onde foi liberada.

Quadro 7 – Caso de uso UC03

3.2.1.4 Fechar câmera

O caso de uso `UC04 – Fechar câmera` descreve os passos seguidos para fechar a janela de uma câmera na aplicação `Cliente`. No Quadro 8 é descrito o caso de uso com o seu detalhamento.

UC04 – Fechar câmera	
Requisitos atendidos	RF05
Pré-condições	Deve haver alguma janela de câmera aberta.
Cenário principal	1) O usuário clica no botão X (`Fechar`). 2) A aplicação `Cliente` fecha a janela. 3) A aplicação `Cliente` exibe a câmera na lista de câmeras disponíveis.
Fluxo alternativo 01	Câmera não disponível: 1) No passo 3, caso a câmera não esteja mais disponível, não deve aparecer na lista de câmeras disponíveis.
Pós-condições	A janela de câmera deve desaparecer e, se a câmera ainda estiver disponível, deve aparecer na lista de câmeras disponíveis.

Quadro 8 – Caso de uso UC04

3.2.1.5 Redimensionar câmera

O caso de uso `UC05 – Redimensionar câmera` descreve o que o usuário precisa fazer para redimensionar uma janela na aplicação `Cliente`. No Quadro 9 são exibidos os detalhes deste caso de uso.

UC05 – Redimensionar câmera	
Requisitos atendidos	RF04
Pré-condições	Deve haver alguma janela de câmera aberta.
Cenário principal	1) O usuário clica no limite da janela da câmera. 2) O usuário arrasta até o tamanho desejado.
Pós-condições	A imagem da câmera deve redimensionar junto com a janela.

Quadro 9 – Caso de uso UC05

3.2.1.6 Obter câmeras

O caso de uso `UC06 – Obter câmeras` descreve as ações que a aplicação `Cliente` toma para obter as câmeras que estão disponíveis para o usuário visualizar. No Quadro 10 é detalhado o caso de uso UC06.

UC06 – Obter câmeras	
Requisitos atendidos	RF07, RF08 e RF09
Pré-condições	Não possui.
Cenário principal	1) A aplicação `Cliente` solicita ao servidor a lista de câmeras disponíveis. 2) O servidor obtêm as câmeras conectadas e monta o *streaming* de cada uma. 3) O servidor fornece uma lista contendo as câmeras que estão conectadas naquele momento.
Pós-condições	Lista contendo as informações das câmeras conectadas no servidor.

Quadro 10 – Caso de uso UC06

3.2.1.7 Atualizar câmeras

O caso de uso `UC07 – Atualizar câmeras` demonstra o procedimento aplicado pela aplicação Silverlight para atualizar as câmeras disponíveis ao usuário bem como as janelas que permanecem abertas. No Quadro 11 são exibidos os detalhes deste caso de uso.

UC07 – Atualizar câmeras	
Requisitos atendidos	RF03
Pré-condições	Lista contendo as câmeras conectadas ao servidor.
Cenário principal	1) A aplicação `Cliente` atualiza a lista de câmeras disponíveis. 2) A aplicação `Cliente` fecha todas as janelas das câmeras que não estão mais disponíveis. 3) A aplicação `Cliente` remove da lista de câmeras disponíveis as câmeras que possuem janela aberta.
Pós-condições	Lista de câmeras disponíveis e janelas atualizadas.

Quadro 11 – Caso de uso UC07

3.2.2 Diagrama de classes

Para demonstrar a implementação necessária para que a solução do protótipo fosse alcançada, foram desenhados os diagramas de classes para cada módulo desenvolvido. Nesta seção serão descritos os diagramas e as classes envolvidas, bem como os métodos de cada

classe.

3.2.2.1 Diagrama de classes da biblioteca Captura

A biblioteca `Captura` é um conjunto de classes que são utilizadas no servidor para efetuar a captura, compressão e *streaming* das imagens das câmeras conectadas. A Figura 6 mostra as classes `Camera` e `Utils` relacionadas entre si e os seus métodos, propriedades e atributos.

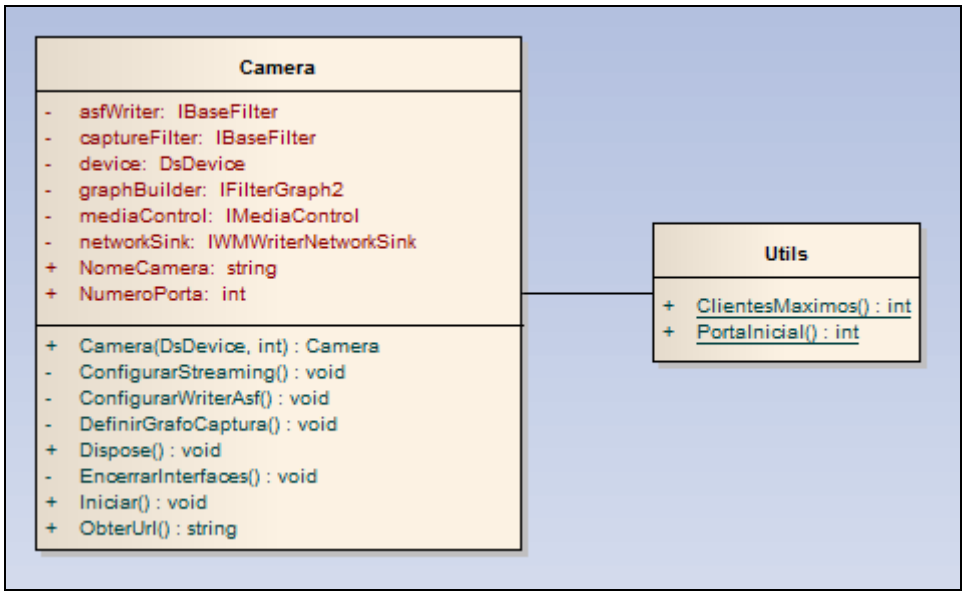


Figura 6 – Diagrama de classes da biblioteca `Captura`

A classe `Camera` possui propriedades, atributos e métodos responsáveis pelo seu funcionamento. As propriedades são:

a) `asfWriter`: filtro responsável pela codificação das imagens da câmera;
b) `captureFilter`: filtro que obtêm as imagens da câmera;
c) `device`: representa o dispositivo de captura de vídeo conectado ao servidor;
d) `graphBuilder`: representa o grafo de filtros;
e) `mediaControl`: responsável pelo controle de fluxo da mídia;
f) `networkSink`: utilizado para despejar os dados das imagens da câmera na rede;
g) `NomeCamera`: obtêm o nome da câmera conectada ao servidor;
h) `NumeroPorta`: obtêm o número da porta definido para a câmera.

Os seguintes métodos estão presentes na classe:

a) `Camera`: construtor responsável por construir o grafo de filtros para a câmera;
b) `ConfigurarStreaming`: método que configura os parâmetros necessário para

iniciar a transmissão das imagens pela rede;

c) `ConfigurarWriterAsf`: método responsável por configurar a codificação das imagens da câmera;

d) `DefinirGrafoCaptura`: define e configura o grafo de captura utilizando os filtros presentes na classe;

e) `Dispose`: método responsável por liberar os recursos utilizados pela câmera no servidor;

f) `EncerrarInterfaces`: método que libera os recursos utilizados pelo grafo de filtros;

g) `Iniciar`: método responsável por iniciar o despejo dos dados da imagem da câmera na porta designada;

h) `ObterUrl`: método responsável por retornar a URL de onde a câmera pode ser acessada na rede.

A classe Utils é responsável por acessar o arquivo de configurações e retornar os valores depositados lá. Os métodos presentes na classe são:

a) `ClientesMaximos`: retorna a quantidade de clientes máximos que pode acessar cada câmera;

b) `PortaInicial`: retorna a primeira porta que deve ser alocada para a uma câmera utilizar.

3.2.2.2 Diagrama de classes da aplicação servidora

Nesta seção será exibido o diagrama das classes `Servico` e `CameraInfo` pertencentes à aplicação `Cliente.Web`. O diagrama está na Figura 7 e depois são descritos os detalhes de cada método e propriedade.

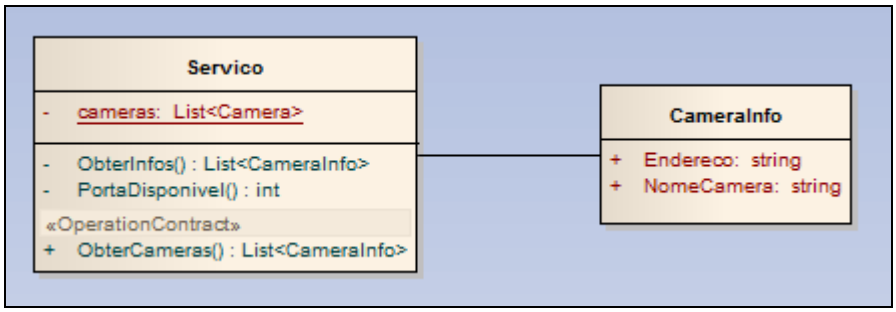


Figura 7 – Diagrama de classes da aplicação `Cliente.Web`

A classe que representa o serviço web deste protótipo possui o atributo `cameras`. Este

atributo é um `List<Camera>` que representa as câmeras conectadas ao servidor. Ainda na classe foram implementados os seguintes métodos:

a) `ObterInfos`: método utilizado para transformar os objetos `Camera` colecionados no atributo `cameras` em objetos `CameraInfo`, para formar uma lista;
b) `PortaDisponivel`: este método retorna a próxima porta disponível para alocar uma câmera;
c) `ObterCameras`: este método é responsável por obter as câmeras conectadas ao servidor onde se encontra o serviço e então retornar uma lista de `CameraInfo`.

A classe `CameraInfo` é responsável por armazenar informações referentes às câmeras conectadas no servidor e possui as propriedades:

a) `Endereco`: esta propriedade é responsável por armazenar a URL referente ao local onde a câmera se encontra na rede;
b) `NomeCamera`: propriedade responsável por armazenar o nome da câmera para exibição na lista de câmeras inativas e na janela da câmera.

3.2.2.3 Diagrama de classes da aplicação Cliente

A Figura 7 mostra o diagrama das classes parciais utilizadas na aplicação Cliente desenvolvida com a tecnologia Silverlight. As classes possuem propriedades, métodos e manipuladores de eventos, que são descritos após a Figura 8.

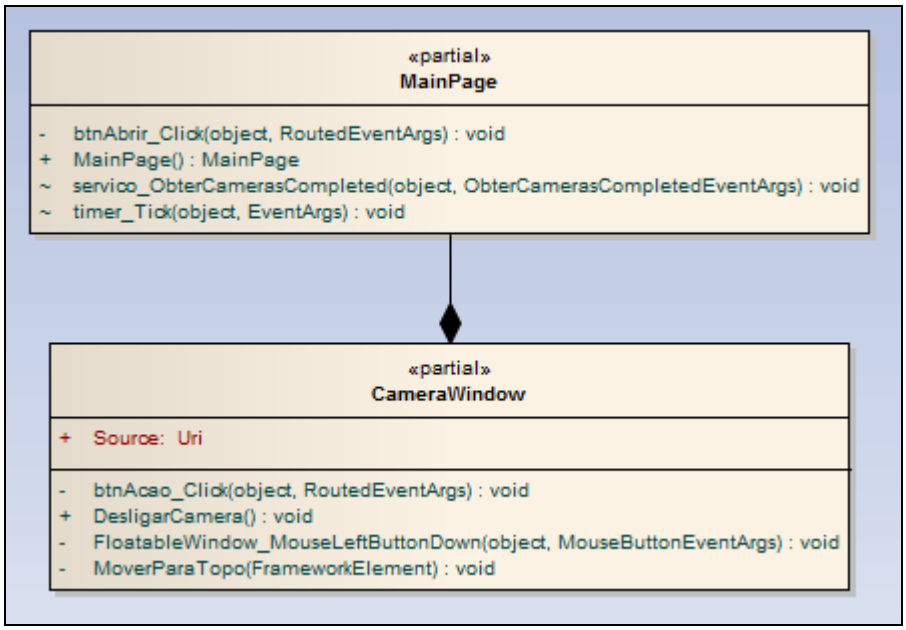


Figura 8 – Diagrama de classes da aplicação `Cliente`

A classe parcial `MainPage` é o *code-behind* da tela principal da aplicação em Silverlight. Ela possui os seguintes métodos e manipuladores de eventos:

a) `btnAbrir_Click`: manipulador do evento de clique no botão responsável por abrir a janela de uma câmera selecionada na lista de câmeras disponíveis;
b) `MainPage`: método construtor que define e inicializa o temporizador de verificação das câmeras;
c) `serviço_ObterCamerasCompleted`: método do serviço web que executa assincronamente para obter as câmeras disponíveis;
d) `timer_Tick`: manipulador do evento *tick* do temporizador.

A classe parcial `CameraWindow` é o *code-behind* da janela responsável por exibir a imagem de uma câmera na aplicação. Esta classe possui a propriedade `Source` que é utilizada para definir o endereço da câmera no componente `MediaElement`. Os métodos e manipuladores de eventos presentes na classe são:

a) `btnAcao_Click`: manipulador do evento de clique do botão responsável pela ação de parar ou continuar a imagem da câmera;
b) `DesligarCamera`: efetua o fechamento da janela flutuante da câmera;
c) `FloatableWindow_MouseLeftButtonDown`: manipulador do evento de clique na janela;
d) `MoverParaTopo`: método responsável por trazer para o topo a janela clicada.

3.2.3 Diagrama de sequência

Para representar a comunicação e troca de mensagens entre as classes de uma aplicação, são utilizados os diagramas de sequência. Nesta seção, na Figura 9 é apresentado o diagrama de sequência que demonstra a funcionalidade do caso de uso UC06.

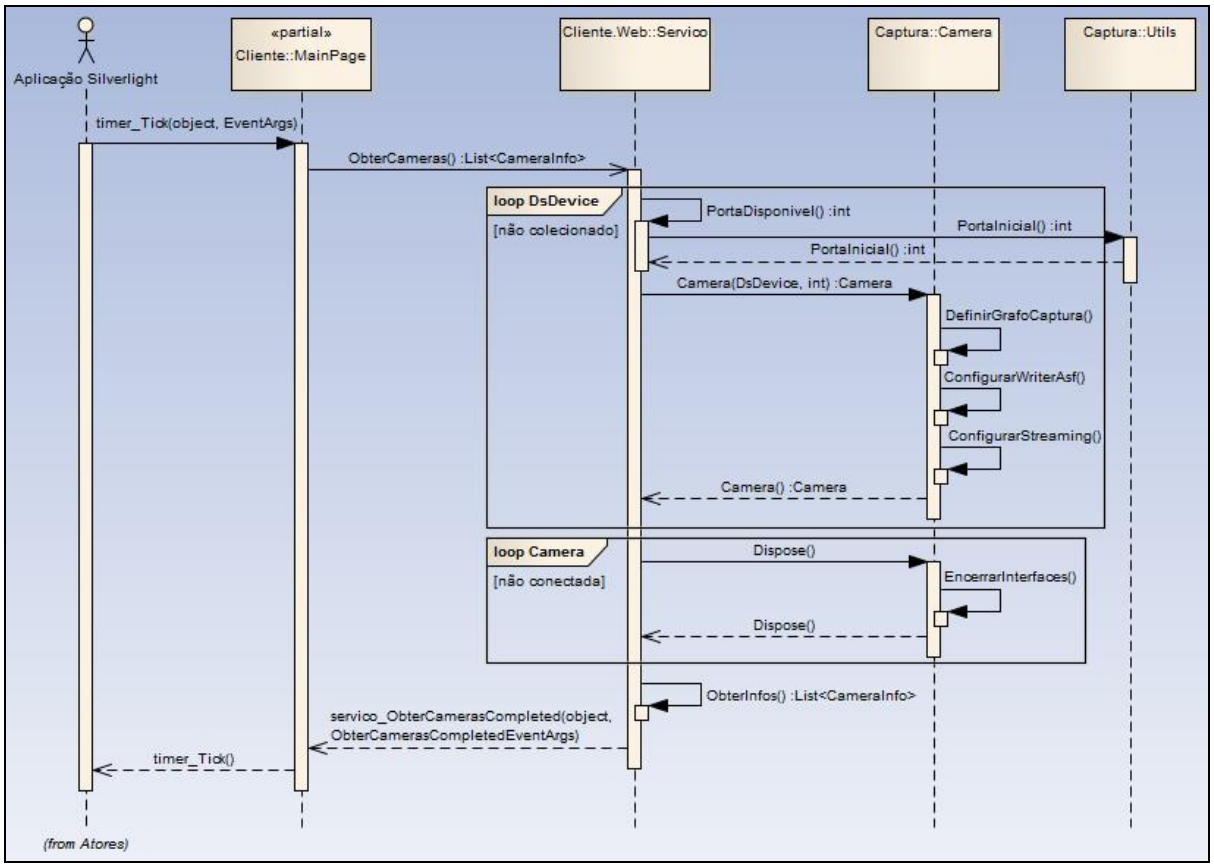


Figura 9 – Diagrama de sequência do caso de uso UC06

Assim que o método `timer_Tick` é ativado pelo evento `Tick` do temporizador na aplicação cliente, é iniciada a obtenção das câmeras conectadas naquele momento no servidor. O grafo de filtros é montado para as câmeras não conectadas previamente e a lista de câmeras – cada uma representada pela classe `CameraInfo` – é retornada para a aplicação cliente no método `servico_ObterCamerasCompleted`.

3.2.4 Layout da interface

Foi elaborado um rascunho para a interface do protótipo prevendo os recursos disponíveis da tecnologia Silverlight. O *layout* pode ser visto na Figura 10.

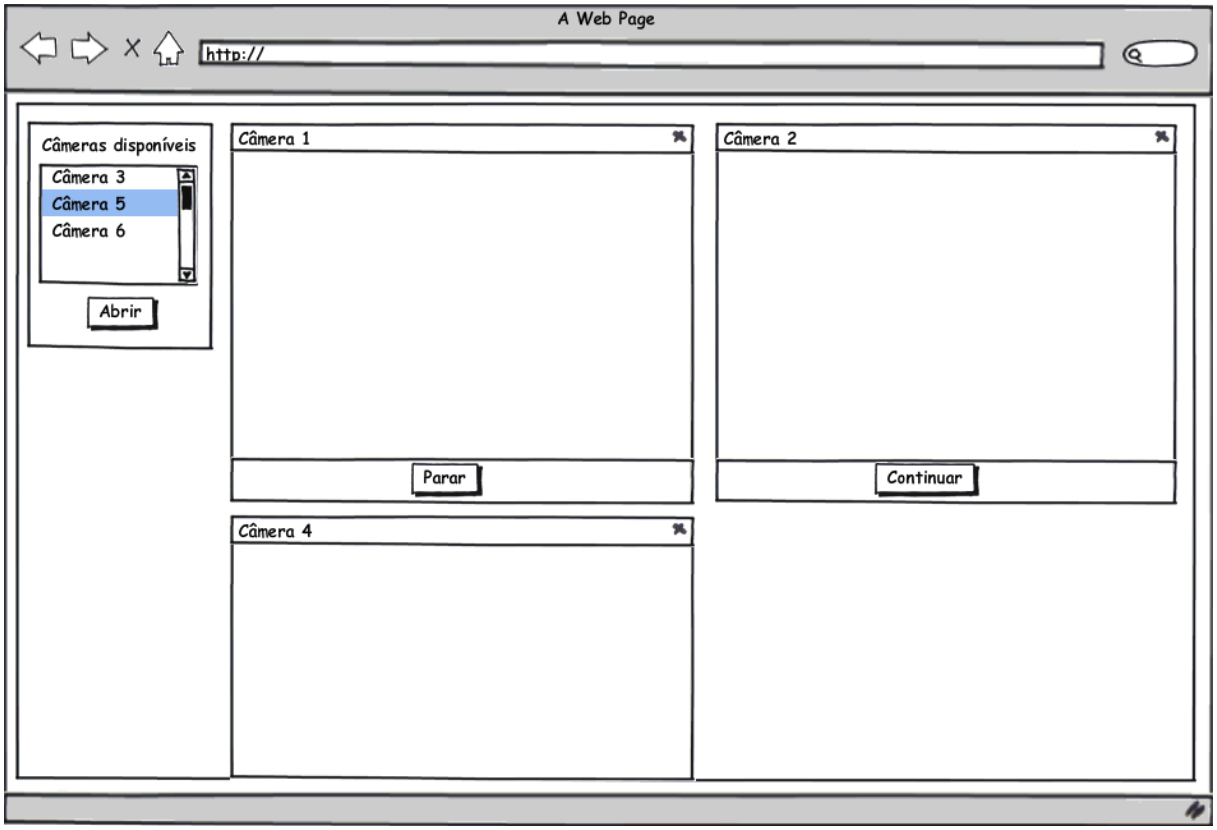


Figura 10 – *Layout* da interface do protótipo

O *layout* apresenta no canto esquerdo superior, um quadro com uma listagem de câmeras que estão conectadas ao servidor, mas que não possuem janela aberta. Logo abaixo, há um botão que é responsável por abrir a janela da câmera selecionada na listagem. No espaço restante da interface ficam dispostas as janelas flutuantes das câmeras. Estas devem ser de livre movimentação e redimensionamento. Cada uma ainda possui um botão para controle da imagem que está vindo da câmera através de *streaming*.

3.3 IMPLEMENTAÇÃO

São descritas nesta seção as técnicas utilizadas para desenvolver o protótipo proposto. Serão mostrados também os módulos implementados e os detalhes do funcionamento de cada um.

3.3.1 Técnicas e ferramentas utilizadas

Em toda a implementação foi utilizado o paradigma da orientação a objetos. O protótipo foi desenvolvido em duas ferramentas IDE. Toda a codificação do protótipo foi desenvolvida na linguagem de programação C# na IDE Microsoft Visual Studio 2008. A interface da aplicação Silverlight foi desenhada com o auxílio da IDE Microsoft Expression Blend 3.

A seguir, nas subseções, são apresentados os detalhes da implementação de cada módulo do protótipo e mostradas as principais classes e métodos.

3.3.1.1 Biblioteca Captura

Nesta biblioteca existem as classes `Camera` e `Utils`, responsáveis pela tarefa de capturar as imagens das câmeras, efetuar a compressão de vídeo e produzir os *streamings* de cada uma, e ainda um arquivo de configuração chamado `app.config`, responsável por armazenar parâmetros utilizados durante o processamento.

Antes de descrevê-las, faz-se necessário citar as classes e interfaces utilizadas para efetuar a captura, compressão e *streaming* das câmeras. Para isto, são utilizadas as bibliotecas DirectShow.NET e WindowsMedia.NET.

A biblioteca DirectShow.NET, como já citado na seção 2.6, possui enumerações, estruturas e interfaces para trabalhar através de classes COM do DirectShow. Assim sendo, o grafo de filtros é construído utilizando-se a interface `IFilterGraph2` e o grafo de captura com a interface `ICaptureGraphBuilder2`.

Os dispositivos de captura de vídeo são representados pela classe `DsDevice`. A interface `IBaseFilter` é a interface base para representar qualquer filtro utilizado para construir o grafo de filtros. Eles são adicionados por um objeto `IFilterGraph2` com o método `AddFilter` e por fim conectados com o método `RenderStream` de um objeto `ICaptureGraphBuilder2`.

A compressão do vídeo é definida com um objeto que implemente a interface `IConfigAsfWriter` da biblioteca WindowsMedia.NET e utilizando o método `ConfigureFilterUsingProfileGuid`. O *Globally Unique Identifier* (GUID) é utilizado no método `ConfigureFilterUsingProfileGuid` para definir o compressor de vídeo. Neste

protótipo o compressor de vídeo utilizado é o WMV 8 da categoria *modem delivery*. Segundo Gill (2001), a categoria *modem delivery* comprime o vídeo com uma taxa de dados de 56 kbps, resolução de 160 x 120 e 15 fps.

Para realizar o *streaming* dos dados, o WindowsMedia.NET fornece a interface `IWMWriterNetworkSink` que, com as configurações de um `IWMWriterAdvanced` define a forma que o *streaming* será realizado.

Com os detalhes anteriores, pode-se entender melhor as classes da biblioteca `Captura`. A classe `Camera` representa cada dispositivo de captura de vídeo conectado ao servidor. Além de efetuar a captura das imagens originadas do dispositivo, ela também é responsável pela compressão e disponibilização destas imagens na rede. No Quadro 12 estão os atributos e propriedades desta classe.

```
                                        ...
16   private IFilterGraph2 graphBuilder = null;
17   private IMediaControl mediaControl = null;
18   private IWMWriterNetworkSink networkSink = null;
19   private DsDevice device = null;
20   private int numeroPorta;
21   private ICaptureGraphBuilder2 captureGraphBuilder = null;
22   private IBaseFilter captureFilter = null;
23   private IBaseFilter asfWriter = null;
24
25   public string NomeCamera
26   {
27      get { return device.Name; }
28   }
29
30   public int NumeroPorta
31   {
32      get { return numeroPorta; }
33   }
                                        ...
```

Quadro 12 – Atributos e propriedades da classe `Camera`

Nas linhas `16` a `23` são definidos os atributos para os objetos utilizados durante a construção do grafo de filtros. Nas linhas `25` a `28`, e `30` a `33`, estão propriedades utilizadas para obter o nome da câmera e o número da porta utilizada pela câmera, respectivamente. No Quadro 13 é mostrado o construtor da classe `Camera`.

```
                                        ...
35   public Camera(DsDevice device, int numeroPorta)
36   {
37       try
38       {
39           this.graphBuilder = (IFilterGraph2)new FilterGraph();
40           this.mediaControl = (IMediaControl)graphBuilder;
41           this.device = device;
42           this.numeroPorta = numeroPorta;
43           DefinirGrafoCaptura();
44       }
                                        ...
50   }
                                        ...
```

Quadro 13 – Construtor da classe `Camera`

O construtor exibido no Quadro 13, acima, recebe dois parâmetros, como pode ser visto na linha `35`. Um deles é um objeto da classe `DsDevice` e o outro é o número da porta que o *streaming* ficará disponível. Das linhas `39` a `42` são atribuídos valores para os atributos previamente mostrados e, na linha `43`, faz-se uma chamada para o método `DefinirGrafoCaptura`, onde o processamento será realizado.

No Quadro 14 é exibido o código do método `DefinirGrafoCaptura` que é chamado no construtor.

```
                                        ...
52   private void DefinirGrafoCaptura()
53   {
54       int hr;
55       try
56       {
57           captureGraphBuilder = (ICaptureGraphBuilder2)new
   CaptureGraphBuilder2();
58           hr = captureGraphBuilder.SetFiltergraph(graphBuilder);
59           DsError.ThrowExceptionForHR(hr);
60           hr = graphBuilder.AddSourceFilterForMoniker(device.Mon, null,
   device.Name, out captureFilter);
61           DsError.ThrowExceptionForHR(hr);
62           ConfigurarWriterAsf();
63           ConfigurarStreaming();
64           hr = graphBuilder.AddFilter(asfWriter, "ASF Writer");
65           DsError.ThrowExceptionForHR(hr);
66           hr = captureGraphBuilder.RenderStream(null, MediaType.Video,
   captureFilter, null, asfWriter);
67           DsError.ThrowExceptionForHR(hr);
68       }
                                        ...
87   }
                                        ...
```

Quadro 14 – Construção do grafo de captura

Na linha `54` é criada uma variável para obter os códigos de retorno das chamadas do DirectShow. Na linha `57` é criado o grafo de captura e na linha `58` o grafo de filtro é ligado ao grafo de captura, como demonstrado na Figura 3. Na linha `59`, caso o código de retorno da chamada for de erro, uma exceção será lançada. Após isto, na linha `60`, é adicionado um filtro

de origem para o grafo de filtros.

Nas linhas 62 e 63 é configurada a compressão das imagens e configurado o *streaming* para transmissão, respectivamente. Depois o filtro de compressão é adicionado ao grafo de filtros, na linha 64, e por fim, os filtros são conectados entre si na linha 66. No Quadro 15 são exibidos os detalhes de como a compressão de vídeo é realizada.

```
                                        ...
143   private void ConfigurarWriterAsf()
144   {
145       int hr;
146       IFileSinkFilter fileSinkFilter = null;
147       try
148       {
149           hr = captureGraphBuilder.SetOutputFileName(MediaSubType.Asf,
   string.Format("{0}.asf", device.Name), out asfWriter, out
   fileSinkFilter);
150           Marshal.ThrowExceptionForHR(hr);
151           WindowsMediaLib.IConfigAsfWriter config =
   (WindowsMediaLib.IConfigAsfWriter)asfWriter;
152           Guid category = new Guid(0x6E2A6955, 0x81DF, 0x4943, 0xBA,
   0x50, 0x68, 0xA9, 0x86, 0xA7, 0x08, 0xF6);
153           config.ConfigureFilterUsingProfileGuid(category);
154       }
                                        ...
163   }
                                        ...
```

Quadro 15 – Definição da compressão de vídeo

Neste procedimento, na linha 146 é criada uma variável para um filtro de gravação de arquivo. Isso é feito para obter a instância do `AsfWriter`, na linha 149, que será utilizado para comprimir o vídeo. Após isso, na linha 151 é definido um `IConfigAsfWriter` a partir do `AsfWriter` obtido anteriormente. Depois é criado um objeto `Guid` para definir o codificador para o vídeo, utilizando o código associado ao WMV 8 categoria *modem delivery*, na linha 152. Por fim, na linha 153 é atribuído o `Guid` para o objeto `IConfigAsfWriter`.

O *streaming* de vídeo de uma câmera é configurado pelo método `ConfigurarStreaming` da classe `Camera`. Este método é exibido no Quadro 16 e descrito na sequência.

```
                                        ...
89   private void ConfigurarStreaming()
90   {
91       IWMWriterAdvanced writerAdvanced = ObterWriterAdvanced();
92       try
93       {
94           RemoverSinks(writerAdvanced);
95           writerAdvanced.SetLiveSource(true);
96           WMUtils.WMCreateWriterNetworkSink(out networkSink);
97           networkSink.SetNetworkProtocol(NetProtocol.HTTP);
98           networkSink.SetMaximumClients(Utils.ClientesMaximos());
99           networkSink.Open(ref numeroPorta);
100          writerAdvanced.AddSink((IWMWriterSink)networkSink);
101      }
                                        ...
110  }
                                        ...
```

Quadro 16 – Configuração do *streaming* de vídeo

O método `ConfigurarStreaming` inicia obtendo-se, na linha `91`, um objeto auxiliar `IWMWriterAdvanced` a partir do objeto `AsfWriter`. Na linha `94` são removidas as saídas para a rede que podem já estar alocadas para o `IWMWriterAdvanced`. Nas linhas `95`, `96`, `97` e `98` é definido que a fonte de vídeo é ao vivo, criada uma nova saída para a rede, definido o protocolo da rede para transmissão e definido o número máximo de clientes para a câmera, respectivamente. Enfim, na linha `99` a saída para a rede é aberta na porta definida para a câmera e na linha `100` esta saída é adicionada ao objeto `IWMWriterAdvanced`.

Para evitar o uso excessivo de recursos do sistema, foram implementados os métodos `Dispose` e `EncerrarInterfaces` para liberar os recursos utilizados pela instância da classe `Camera`. Os métodos são exibidos no Quadro 17, seguidos da descrição de seus procedimentos.

```
                                        ...
175   public void Dispose()
176   {
177       GC.SuppressFinalize(this);
178       EncerrarInterfaces();
179   }
                                        ...
186   private void EncerrarInterfaces()
187   {
188       int hr;
189       try
190       {
191           if (mediaControl != null)
192           {
193               hr = mediaControl.Stop();
194           }
195       }
196       catch (Exception ex)
197       {
198           Debug.WriteLine(ex);
199       }
200       if (networkSink != null)
201       {
202           networkSink.Disconnect();
203           Marshal.ReleaseComObject(networkSink);
204           networkSink = null;
205       }
206       if (graphBuilder != null)
207       {
208           Marshal.ReleaseComObject(graphBuilder);
209           graphBuilder = null;
210       }
211   }
                                        ...
```

Quadro 17 – Liberação dos recursos utilizados

No método `Dispose`, na linha `177` é sinalizado para o coletor de lixo do .NET Framework que este método já está liberando os recursos utilizados e que não é mais necessário invocá-lo posteriormente. A seguir é chamado o método `EncerrarInterfaces` que, na linha `193` pára o processo de captura, compressão e *streaming* da câmera através do grafo de filtros. Após isto, na linha `202`, a saída para a rede é desconectada e na linha `203` o objeto é liberado. Por último o grafo de filtros é liberado, na linha `208`.

O Quadro 18 mostra os métodos responsáveis por fazer todo o procedimento acima descrito começar a funcionar e obter a URL da câmera conectada, respectivamente.

```
                                        ...
165   public void Iniciar()
166   {
167       if (!executando)
168       {
169           int hr = mediaControl.Run();
170           DsError.ThrowExceptionForHR(hr);
171           executando = true;
172       }
173   }
                                        ...
213   public string ObterUrl()
214   {
215       return string.Format("http://localhost:{0}", numeroPorta);
216   }
                                        ...
```

Quadro 18 – Início do processo de captura, compressão e transmissão

O método Iniciar começa verificando, na linha `167`, se o procedimento já está sendo executado. Caso não esteja, na linha `169` o procedimento é ativado. No método `ObterUrl`, retorna-se a URL que será utilizada pela aplicação `Cliente` para conectar o `MediaElement` na câmera.

Os outros dois arquivos presentes na biblioteca `Captura`, a classe `Utils` e o arquivo `app.config` são demonstrados nos Quadro 19 e Quadro 20 com a descrição de cada um na sequência.

```
                                        ...
09   public class Utils
10   {
11       public static int PortaInicial()
12       {
13           return Settings.Default.PortaInicial;
14       }
15
16       public static int ClientesMaximos()
17       {
18           return Settings.Default.ClientesMaximos;
19       }
20   }
                                        ...
```

Quadro 19 – Classe de acesso às configurações da biblioteca

O método `PortaInicial` das linhas `11` a `14` obtêm a primeira porta que deve ser alocada para a primeira câmera conectada. Nas linhas `16` a `19` está o método `ClientesMaximos` que obtêm a quantidade de clientes máxima que pode se conectar à câmera.

```
                                        ...
08   <applicationSettings>
09       <Captura.Properties.Settings>
10           <setting name="PortaInicial" serializeAs="String">
11               <value>50600</value>
12           </setting>
13           <setting name="ClientesMaximos" serializeAs="String">
14               <value>5</value>
15           </setting>
16       </Captura.Properties.Settings>
17   </applicationSettings>
                                        ...
```

Quadro 20 – Configurações da biblioteca codificadas em XML

No Quadro 20 está o trecho de código em XML referente às configurações da biblioteca. Nas linhas `10` a `12` está definida a configuração `PortaInicial` com o valor 50600. A configuração `ClientesMaximos`, nas linhas `13` a `15` é definida com o valor 5.

3.3.1.2 Aplicação Cliente

A interface RIA desenvolvida com a tecnologia Silverlight é a aplicação `Cliente`. A interface desta aplicação foi desenhada utilizando-se dos recursos da IDE Microsoft Expression Blend 3. O *code-behind*, responsável pelo funcionamento destas telas foi desenvolvido através da IDE Microsoft Visual Studio 2008.

Há uma referência para o serviço web, descrito posteriormente, para obtenção das câmeras conectadas ao servidor. Também é utilizada a referência para a DLL `FloatableWindow.dll` que se faz necessária nesta aplicação, pois esta permite que haja interação com o controle pai, neste caso, o painel principal.

O painel principal da interface está escrito no formato de marcação XAML no arquivo `MainPage.xaml` que é exibido no Quadro 21 seguido de sua descrição.

```
01   <UserControl ... x:Class="Cliente.MainPage">
02       <Grid x:Name="LayoutRoot" Background="#FFC2C8DC" >
03           <Grid.ColumnDefinitions>
04               <ColumnDefinition Width="150"/>
05               <ColumnDefinition Width="*"/>
06           </Grid.ColumnDefinitions>
07           <Grid Grid.Column="0" VerticalAlignment="Top" Margin="8">
08               <Border BorderThickness="1" BorderBrush="Black"
   Height="250">
09                   <StackPanel Background="#FFCEE4E3">
10                       <dataInput:Label Content="Câmeras disponíveis"
   Margin="8,8,8,4" HorizontalAlignment="Center"/>
11                       <ListBox x:Name="lstCamerasDisponiveis"
   Margin="8,4,8,4" Height="180"/>
12                       <Button x:Name="btnAbrir" Margin="8,4,8,8"
   Content="Abrir" Click="btnAbrir_Click"/>
13                   </StackPanel>
14               </Border>
15           </Grid>
16           <Grid x:Name="colCameras" Grid.Column="1" Margin="8">
17               <Canvas x:Name="cavCameras" HorizontalAlignment="Stretch"
   VerticalAlignment="Stretch"></Canvas>
18           </Grid>
19       </Grid>
20   </UserControl>
```

Quadro 21 – Marcação XAML do painel principal

Na marcação XAML, na linha `01` é definido o controle pai do painel. Nas linhas `03` a `06` são definidos os tamanhos de cada coluna sendo a primeira coluna onde fica a lista de câmeras disponíveis e a outra, de tamanho livre, onde ficarão as janelas flutuantes. A lista é definida na linha `11` e o botão responsável por abrir as janelas, na linha `12`. Por fim, na linha `17` um `Canvas` onde as janelas são criadas.

O Quadro 22 mostra a classe parcial `MainPage` onde o *code-behind* da tela efetua os processamentos necessários para seu funcionamento.

```
                                    ...
17   public partial class MainPage : UserControl
18   {
19       private static List<CameraInfo> infos = new List<CameraInfo>();
20       private ServicoClient servico = new ServicoClient();
21       private DispatcherTimer timer = new DispatcherTimer();
                                    ...
98   }
                                    ...
```

Quadro 22 – Classe parcial *code-behind* da tela `MainPage`

A classe parcial `MainPage` possui 3 atributos declarados nas linhas `19` a `21`. O atributo `infos` é um `List<CameraInfo>` que coleciona as informações das câmeras conectadas. O `servico` é um objeto referente ao serviço web e o `timer` é o temporizador responsável pela verificação periódica de novas câmeras conectadas. O método construtor da classe é exibido no Quadro 23, seguido de sua explicação.

```
                                    ...
23   public MainPage()
24   {
25       InitializeComponent();
26       servico.ObterCamerasCompleted += new
   EventHandler<ObterCamerasCompletedEventArgs>(servico_ObterCamerasComple
   ted);
27       servico.ObterCamerasAsync();
28       timer.Interval = new TimeSpan(0, 0, 8);
29       timer.Tick += new EventHandler(timer_Tick);
30       timer.Start();
31   }
                                    ...
```

Quadro 23 – Construtor do painel principal

Na linha `26` é criado um manipulador para o evento `ObterCamerasCompleted` do serviço web. Na linha `27` é feita uma primeira chamada para o método web assincronamente com o objetivo de iniciar a aplicação já carregando as câmeras conectadas. O intervalo para a verificação periódica é definido na linha `28` e, na linha `29`, o manipulador para o evento `Tick` do temporizador é criado. Ao final, na linha `30` é iniciado o temporizador.

O Quadro 24 mostra os métodos manipuladores dos eventos `ObterCamerasCompleted` e `Tick`, previamente descritos, seguidos de suas descrições.

```
                                        ...
33   void timer_Tick(object sender, EventArgs e)
34   {
35       servico.ObterCamerasAsync();
36   }
37
38   void servico_ObterCamerasCompleted(object sender,
   ObterCamerasCompletedEventArgs e)
39   {
40       infos = e.Result;
41       lstCamerasDisponiveis.Items.Clear();
42       foreach (CameraInfo info in infos)
43       {
44           foreach (UIElement element in colCameras.Children)
45           {
46               if (elemento is FloatableWindow)
47               {
48                   if ((((FloatableWindow)element).Title as String) ==
   info.NomeCamera)
49                   {
50                       info.Aberto = true;
51                       break;
52                   }
53               }
54           }
55           if (!info.Aberto)
56           {
57               lstCamerasDisponiveis.Items.Add(new ListBoxItem() {
   Content = info.NomeCamera, DataContext = info.Endereco });
58           }
59       }
60       foreach (UIElement element in colCameras.Children)
61       {
62           if (elemento is FloatableWindow)
63           {
64               bool existe = false;
65               foreach (CameraInfo info in infos)
66               {
67                   if ((((FloatableWindow)element).Title as String) ==
   info.NomeCamera)
68                   {
69                       existe = true;
70                       break;
71                   }
72               }
73               if (!existe)
74               {
75                   ((CameraWindow)elemento).DesligarCamera();
76               }
77           }
78       }
79   }
                                        ...
```

Quadro 24 – Manipuladores de eventos do painel principal

No método `timer_Tick`, na linha `35` a cada intervalo de tempo definido no método construtor é chamada a verificação das câmeras através do método `ObterCamerasAsync`. No método `servico_ObterCamerasCompleted`, na linha `40` obtêm-se o resultado do

processamento como um `List<CameraInfo>`. Na linha `41` a lista de câmeras disponíveis é limpa para ser efetuada a atualização com as câmeras conectadas.

Nas linhas `42` a `59` é feita a atualização da lista de câmeras disponíveis. Para cada câmera retornada do método web é verificado se já existe alguma janela aberta e, caso não haja, a câmera é adicionada na lista. Nas linhas `60` até `78` há uma iteração para verificar se a câmera correspondente a cada janela aberta está ainda disponível. Caso não esteja, a janela é fechada com o método `DesligarCamera`.

Outro método da classe `MainPage` é o `btnAbrir_Click` que é responsável por manipular o evento `Click` do botão para abrir janelas de câmeras. O método é exibido no Quadro 25 seguido de seu detalhamento.

```
                                        ...
86   private void btnAbrir_Click(object sender, RoutedEventArgs e)
87   {
88       ListBoxItem selecionado = null;
89       if ((selecionado =
   (ListBoxItem)lstCamerasDisponiveis.SelectedItem) != null)
90       {
91           FloatableWindow janela = new CameraWindow() { Title =
   selecionado.Content, Source = new
   Uri(selecionado.DataContext.ToString()) };
92           janela.ParentLayoutRoot = cavCameras;
93           lstCamerasDisponiveis.Items.Remove(selecionado);
94           janela.Show();
95       }
96   }
                                      ...
```

Quadro 25 – Manipulador do evento `Click` no painel principal

Primeiro é verificado, na linha `89`, se há algum item da lista de câmeras disponíveis selecionado. Caso haja, uma instância da janela é criada, o controle pai é definido como o `Canvas`, o item é removido da lista e a janela é exibida, nas linhas `91`, `92`, `93` e `94` respectivamente.

Agora será detalhada a janela de câmeras. O arquivo que contêm a marcação XAML de uma janela flutuante é o `CameraWindow.xaml`. No Quadro 26 é exibido o conteúdo do arquivo e na sequência são descritos os seus detalhes.

```
01   <controls:FloatableWindow ... x:Class="Cliente.CameraWindow"
   Width="320" Height="240"
   MouseLeftButtonDown="FloatableWindow_MouseLeftButtonDown">
02       <Grid x:Name="LayoutRoot" Background="#FFC7E0C7">
03           <Grid.RowDefinitions>
04               <RowDefinition />
05               <RowDefinition Height="Auto" />
06           </Grid.RowDefinitions>
07           <Grid Grid.Row="0">
08               <MediaElement x:Name="medCamera"
   HorizontalAlignment="Stretch"
   VerticalAlignment="Stretch"></MediaElement>
09           </Grid>
10           <Grid Grid.Row="1">
11               <Button x:Name="btnAcao" Content="Parar" Width="75"
   Height="23" HorizontalAlignment="Center" Click="btnAcao_Click"/>
12           </Grid>
13       </Grid>
14   </controls:FloatableWindow>
```

Quadro 26 – Marcação XAML da janela flutuante

Na linha `01` o controle `FloatableWindow` é declarado com as medidas nos atributos `Width` e `Height` além da definição do manipulador para o evento `MouseLeftButtonDown`. Na linha `08` é definido o controle responsável por exibir as imagens, `MediaElement`. Abaixo, na linha `11` está a marcação do botão que efetua as ações de parar e continuar a exibição das imagens.

No Quadro 27 é mostrada a classe `CameraWindow` com as propriedades e o método `DesligarCamera`, descritos na sequência.

```
                                        ...
16   public partial class CameraWindow : FloatableWindow
17   {
18       private bool executando = true;
19
20       public Uri Source
21       {
22           set { medCamera.Source = value; }
23       }
                                        ...
30       public void DesligarCamera()
31       {
32           this.Close();
33       }
                                        ...
68   }
                                        ...
```

Quadro 27 – Classe `CameraWindow` e método `DesligarCamera`

Na linha `18` é declarado um atributo para saber se a imagem está reproduzindo ou se está parada. Nas linhas `20` a `23` a propriedade `Source` define a URL da câmera para o `MediaElement` e nas linhas `30` a `33` o método `DesligarCamera` fecha a janela. No Quadro 28 está o método `btnAcao_Click`.

```
                                        ...
35   private void btnAcao_Click(object sender, RoutedEventArgs e)
36   {
37       if (executando)
38       {
39           medCamera.Stop();
40           btnAcao.Content = "Continuar";
41           executando = false;
42       }
43       else
44       {
45           medCamera.Play();
46           btnAcao.Content = "Parar";
47           executando = true;
48       }
49   }
                                        ...
```

Quadro 28 – Manipulador do evento `Click` do botão de ação

O método verifica, na linha `37`, se a imagem está reproduzindo. Se estiver reproduzindo, nas linhas `39`, `40` e `41` o `MediaElement` é parado, o texto do botão é alterado para "Continuar" e o atributo `executando` é alterado para `false`, respectivamente. Caso contrário, nas linhas `45`, `46` e `47` o `MediaElement` reproduz, o texto do botão é alterado para "Parar" e o atributo `executando` é alterado para `true`, respectivamente. Por fim, no Quadro 29 é exibido o manipulador do evento `MouseLeftButtonDown` da janela flutuante e o método `MoverParaTopo`.

```
                                        ...
51   private void FloatableWindow_MouseLeftButtonDown(object sender,
   MouseButtonEventArgs e)
52   {
53       FrameworkElement element = (FrameworkElement)sender;
54       MoverParaTopo(element);
55   }
56
57   private void MoverParaTopo(FrameworkElement element)
58   {
59       Panel panel = (Panel)element.Parent;
60       int highestElement = 0;
61       foreach (UIElement elem in panel.Children)
62       {
63           int tmpHighestElement = Canvas.GetZIndex(elem);
64           highestElement = (tmpHighestElement > highestElement) ?
   tmpHighestElement : highestElement;
65       }
66       Canvas.SetZIndex(element, highestElement + 1);
67   }
                                        ...
```

Quadro 29 – Manipulador do evento `MouseLeftButtonDown` e método auxiliar

Na linha `53` obtêm-se a janela que foi clicada e na linha `54` é chamado o método `MoverParaTopo`. O método `MoverParaTopo` é um método auxiliar para fazer com que um elemento fique com um valor acima dos outros no eixo Z. Primeiro é obtido o `Panel` pai das

janelas abertas, na linha 59. Nas linhas 61 a 65 há uma iteração entre os filhos deste Panel para descobrir qual o maior valor no eixo Z. Enfim, é definido para o elemento clicado o maior valor mais 1, na linha 66.

3.3.1.3 Aplicação servidora

Esta aplicação consiste numa aplicação ASP.NET 3.5 que possui uma página que carrega a aplicação Cliente Silverlight, descrita acima, um serviço web responsável por controlar as câmeras conectadas no computador e a classe CameraInfo que armazena os dados de uma câmera conectada. No Quadro 30 é exibida a página Cliente.aspx responsável pelo carregamento da aplicação Cliente, seguida de descrição.

```
                                                ...
17   <body>
18     <table cellpadding="0" cellspacing="0" style="height: 100%; width:
   100%;">
19       <tr>
20         <td valign="middle">
21           <table cellpadding="0" cellspacing="0" style="height: 99%;
   width: 99%; border: solid thin black; margin: auto; text-align:
   center;">
22             <tr>
23               <td>
24                 <object data="data:application/x-silverlight-2,"
   type="application/x-silverlight-2" width="100%" height="100%" >
25                   <param name="source" value="ClientBin/Cliente.xap" />
26                 </object>
27               </td>
28             </tr>
29           </table>
30         </td>
31       </tr>
32     </table>
33   </body>
                                                ...
```

Quadro 30 – Marcação onde é carregada a aplicação Cliente

Na HTML da linha 18 define-se uma table que serve como contêiner para a aplicação Cliente. Na linha 24 é definido um object onde a aplicação desenvolvida em Silverlight será executada. Este object possui um atributo type que configura o tipo de aplicação que será executada nele. Aninhado neste object, na linha 25 há um param que aponta o arquivo de recursos da aplicação Cliente, posicionado na pasta ClientBin da aplicação Cliente.Web. No Quadro 31 é exibida a classe CameraInfo.

```
                                        ...
08    public class CameraInfo
09    {
10        public string NomeCamera { get; set; }
11        public string Endereco { get; set; }
12    }
                                        ...
```

Quadro 31 – Classe para armazenar informações da câmera

A classe `CameraInfo` possui duas propriedades definidas. Na linha `10`, a propriedade `NomeCamera` é utilizada para armazenar o nome da câmera obtido de um objeto `Camera`. A propriedade `Endereco`, definida na linha `11`, armazena o endereço de rede da câmera.

O serviço web que controla as câmeras conectadas no computador foi desenvolvido utilizando a tecnologia *Windows Communication Foundation* (WCF) da Microsoft. A classe `Servico` é exibida no Quadro 32.

```
                                        ...
13    [ServiceContract(Namespace = "")]
14    [AspNetCompatibilityRequirements(RequirementsMode =
    AspNetCompatibilityRequirementsMode.Allowed)]
15    public class Servico
16    {
17        private static List<Camera> cameras = new List<Camera>();
                                        ...
93    }
                                        ...
```

Quadro 32 – Atributos do serviço web

Na linha `13`, o atributo `ServiceContract` define que a classe `Servico` será utilizada como um serviço web. O atributo `AspNetCompatibilityRequirements` define, na linha `14`, que os métodos web da classe `Servico` são compatíveis com aplicações ASP.NET. Na linha `17` criado o atributo `camera` do tipo `List<Camera>` para colecionar as câmeras conectadas no computador servidor. No Quadro 33 está o método web `ObterCameras` seguido de sua explicação.

```
                                        ...
19   [OperationContract]
20   public List<CameraInfo> ObterCameras()
21   {
22       DsDevice[] captureDevices;
23       captureDevices =
   DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice);
24       foreach (DsDevice device in captureDevices)
25       {
26           if (!Existe(device))
27           {
28               Camera novaCamera = new Camera(device, PortaDisponivel());
29               cameras.Add(novaCamera);
30               novaCamera.Iniciar();
31           }
32       }
33       List<DsDevice> devices = new List<DsDevice>(captureDevices);
34       List<Camera> removidas = new List<Camera>();
35       foreach (Camera camera in cameras)
36       {
37           if (!Existe(devices, camera))
38           {
39               removidas.Add(camera);
40           }
41       }
42       foreach (Camera camera in removidas)
43       {
44           camera.Dispose();
45           cameras.Remove(camera);
46       }
47       return ObterInfos();
48   }
                                        ...
```

Quadro 33 – Rotina de obtenção das câmeras conectadas

O método `ObterCameras` inicia sendo definido, na linha `19`, como um método web. Nas linhas `22` e `23` os dispositivos de captura de vídeo são obtidos através do método estático `GetDevicesOfCat` da classe `DsDevice`. O enumerador `FilterCategory` possui diversas categorias de dispositivos dos quais pode-se obter.

Nas linhas `24` a `32` há uma iteração sobre os dispositivos capturados. Na linha `26` é verificado se o dispositivo já está colecionado e, caso não esteja, uma `Camera` é instanciada, na linha `28`, tendo como parâmetros o dispositivo e a próxima porta disponível. A câmera é então colecionada e depois a transmissão é iniciada, nas linhas `29` e `30` respectivamente.

A partir da linha `33` até a linha `46`, são removidas as câmeras que não foram mais detectadas. Detalhe para a linha `44`, onde o método `Dispose` é acionado para liberação dos recursos alocados anteriormente. Por fim, na linha `47` é feito o retorno do método `ObterCameras` criando uma coleção de `CameraInfo` através do método `ObterInfos`. No Quadro 34 é mostrado o método `PortaDisponivel` seguido do seu detalhamento.

```
                                        ...
76   private int PortaDisponivel()
77   {
78       int porta = Captura.Utils.PortaInicial();
79       bool existe = true;
80       do
81       {
82           existe = cameras.Exists(delegate(Camera c)
83           {
84               return c.NumeroPorta == porta;
85           });
86           if (existe)
87           {
88               porta++;
89           }
90       } while (existe);
91       return porta;
92   }
                                        ...
```

Quadro 34 – Obtenção da próxima porta disponível

Na linha `78` é obtida do arquivo de configuração a porta inicial para alocações das câmeras. Verifica-se então, nas linhas `82` a `85` se esta porta já está alocada para alguma das câmeras colecionadas. Repete-se então a verificação após o incremento do número da porta na linha `88`. Por fim, na linha `91` a primeira porta vaga é retornada. O método `ObterInfos` é exibido no Quadro 35.

```
                                        ...
50   private List<CameraInfo> ObterInfos()
51   {
52       List<CameraInfo> infos = new List<CameraInfo>();
53       foreach (Camera camera in cameras)
54       {
55           infos.Add(new CameraInfo() { Endereco = camera.ObterUrl(),
   NomeCamera = camera.NomeCamera });
56       }
57       return infos;
58   }
                                        ...
```

Quadro 35 – Obtenção e criação da lista de informações de câmeras

No método `ObterInfos` é feita uma iteração da linha `53` a `56` sobre as câmeras colecionadas. Para cada uma destas câmeras é instanciado e colecionado um `CameraInfo`, obtendo-se a URL da câmera e o seu nome. Ao final, na linha `57` retorna-se a lista.

3.3.2 Operacionalidade da implementação

Nesta seção são demonstradas as interações que o usuário pode efetuar na interface do protótipo. A operacionalidade é apresentada seguindo os casos de uso onde o ator é o usuário

da aplicação. É necessário destacar que o *plugin* do Silverlight 3 ou superior precisar estar instalado no computador do usuário.

3.3.2.1 Visualizar câmera

A operacionalidade relacionada com o caso de uso `UC01 – Visualizar câmera` é apresentada na Figura 11 e com uma descrição posterior.

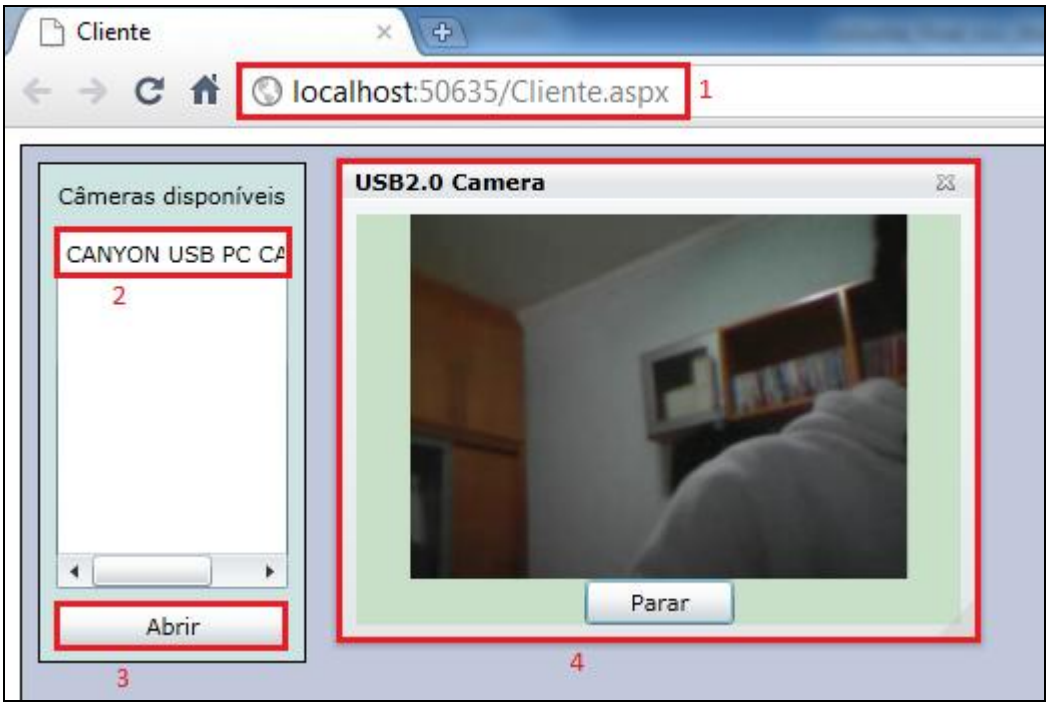


Figura 11 – Operacionalidade do caso de uso UC01

Para visualizar uma câmera, o usuário deve seguir os passos marcados com os números na sequência em vermelho apresentados na Figura 11.

a) passo 1: através de um navegador Microsoft Internet Explorer 8 ou Google Chrome 11, o usuário acessa aplicação;
b) passo 2: o usuário seleciona a câmera desejada na listagem contendo as câmeras que estão conectadas ao servidor, mas que não estão abertas;
c) passo 3: o usuário então pressiona o botão Abrir;
d) passo 4: a janela da câmera é aberta e é iniciada a visualização ao vivo.

3.3.2.2 Parar/continuar câmera

A operacionalidade relacionada com o caso de uso `UC02 – Parar/continuar câmera` é apresentada na Figura 12 seguida de uma descrição correspondente.

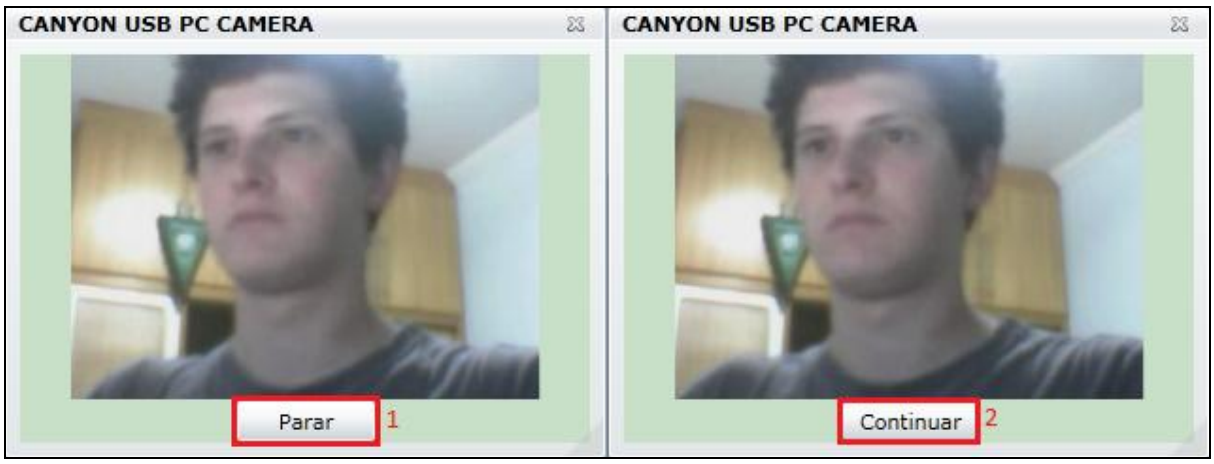


Figura 12 – Operacionalidade do caso de uso UC02

A Figura 12 é uma montagem com duas imagens da mesma janela de câmera, mas em momentos diferentes. Os passos estão marcados com os números na sequência em vermelho.

a) passo 1: enquanto a imagem da câmera estiver reproduzindo, o texto do botão é "Parar", portanto ao clicar nele, a imagem será congelada;
b) passo 2: quando a imagem estiver congelada, o texto do botão é "Continuar", sendo assim, ao clicar nele a imagem carregará e será reproduzida novamente.

3.3.2.3 Movimentar janela da câmera

A operacionalidade relacionada com o caso de uso `UC03 – Movimentar janela da câmera` é mostrada na Figura 13 e com uma descrição posterior.

Figura 13 – Operacionalidade do caso de uso UC03

A Figura 13 consiste numa montagem com duas imagens do painel principal com a mesma janela de câmera, mas em momentos e posições diferentes. Os passos estão marcados com os números na sequência em vermelho.

a) passo 1: o usuário seleciona uma janela flutuante clicando no cabeçalho dela;
b) passo 2: mantendo o botão do mouse utilizado para selecionar pressionado, o usuário arrasta a janela para o local desejado.

3.3.2.4 Fechar câmera

Na Figura 14 é mostrada a operacionalidade do caso de uso `UC04 – Fechar câmera`, seguida de uma descrição para melhor entendimento.

Figura 14 – Operacionalidade do caso de uso UC04

A Figura 14 mostra uma montagem de duas imagens do painel principal da aplicação, mas em momentos distintos. Os passos estão numerados na sequência em vermelho.

a) passo 1: caso o usuário deseja fechar uma janela, deve apontar para o `X` no canto superior direito da janela e clicá-lo;
b) passo 2: após a aplicação verificar que a câmera ainda está disponível, mas não possui nenhuma janela aberta, esta câmera ficará na lista de câmeras disponíveis.

3.3.2.5 Redimensionar câmera

A operacionalidade do caso de uso `UC05 – Redimensionar câmera` é exibida na Figura 15. Logo após são descritos os passos necessários para a execução.

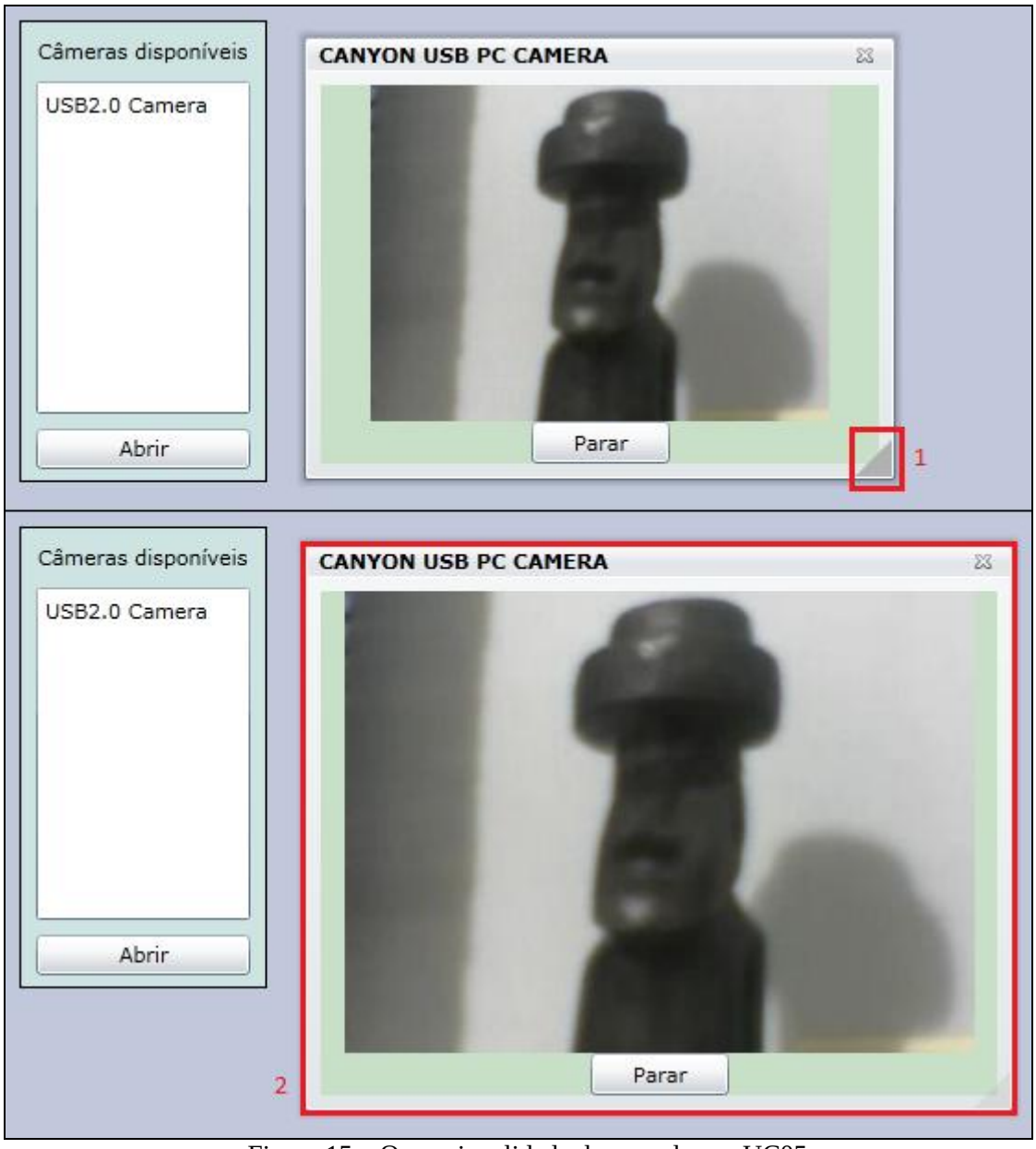


Figura 15 – Operacionalidade do caso de uso UC05

A Figura 15 consiste numa montagem de duas imagens do painel principal da aplicação demonstrando os passos para redimensionar uma janela de câmera. Os passos estão numerados na sequência em vermelho.

a) passo 1: o usuário deve clicar no triângulo presente no canto inferior direito;
b) passo 2: mantendo o botão do mouse responsável pelo clique pressionado, o usuário deve arrastar as bordas da janela até que esta possua o tamanho desejado.

3.4 RESULTADOS E DISCUSSÃO

Os resultados alcançados no desenvolvimento deste protótipo são descritos e discutidos nesta seção. Os requisitos levantados durante a proposta foram todos atendidos como pode ser visto no Quadro 36.

Requisitos funcionais	**Atendido**
RF01: A aplicação cliente deve disponibilizar um painel que ocupe o maior espaço possível da janela do navegador onde o usuário poderá interagir.	Sim
RF02: A aplicação cliente deve disponibilizar um quadro para cada *streaming* de câmera que estiver recebendo.	Sim
RF03: A aplicação cliente deve possuir um menu onde podem ser escolhidas as câmeras escondidas que se deseja restaurar.	Sim
RF04: A aplicação cliente deve permitir redimensionamento e movimentação livres (limitado pelo tamanho do painel) de cada quadro de câmera.	Sim
RF05: A aplicação cliente deve permitir fechar cada quadro de câmera.	Sim
RF06: A aplicação cliente deve permitir parar e continuar a exibição das imagens de cada janela de câmera.	Sim
RF07: A aplicação servidora deve ler a entrada de cada câmera conectada ao computador.	Sim
RF08: A aplicação servidora deve processar os dados e criar o *streaming* de vídeo de cada câmera.	Sim
RF09: A aplicação servidora deve disponibilizar os *streamings* para leitura na internet.	Sim

Quadro 36 – Atendimento dos requisitos

O objetivo do protótipo desenvolvido foi utilizar os benefícios da RIA através da tecnologia Silverlight. O que se conseguiu também foi trabalhar com tecnologias de manipulação de mídia, com o auxílio do DirectShow e do Windows Media Format SDK.

Nos testes realizados foram utilizadas duas câmeras com conexão *Universal Serial Bus* (USB). Os navegadores de internet testados foram o Microsoft Internet Explorer 8 e o Google Chrome 11. Um requisito necessário para a execução da aplicação cliente é o *plugin* Silverlight 3 ou superior instalado no computador. O protótipo foi testado somente no computador de desenvolvimento, que possui as seguintes características:

a) computador *notebook* modelo Dell Vostro 1310;
b) sistema operacional Microsoft Windows 7 Professional 64-*bit* Service Pack 1;
c) processador Intel Core 2 Duo T5670 1.8 Gigahertz;
d) memória *Random Access Memory* (RAM) 4.0 gigabytes *dual-channel Double Data Rate* (DDR) 2 332 Megahertz;
e) placa de video NVIDIA GeForce 8400M GS;

f) resolução de vídeo 1280 x 800 *pixels* com taxa de atualização de 60 Hertz e profundidade de cores de 32 *bits*;

g) disco rígido de 80 Gigabytes;

h) câmera USB com resolução de vídeo 320 x 240 *pixels* e captura de 30 quadros por segundo;

i) câmera USB com resolução de vídeo 640 x 480 *pixels* e captura de 30 quadros por segundo.

Ao executar os testes no protótipo, não foi percebida lentidão ou travamento nos movimentos das janelas ou das imagens das câmeras. Tanto para movimentar as janelas flutuantes quanto para redimensioná-las ou fechá-las mostrou que a aplicação possui um bom tempo de resposta para as ações. O tempo de carregamento da imagem numa janela é, na média, de 6 segundos de *buffer*.

Comparando as funcionalidades do protótipo desenvolvido para este trabalho de conclusão de curso com outras ferramentas desenvolvidas, é possível verificar algumas particularidades entre elas. O Quadro 37 foi criado para comparar o protótipo desenvolvido com o trabalho desenvolvido por Carlassara (2009) e com a ferramenta comercial WebCam Monitor 5.24 da DeskShare Incorporated (2011).

Funcionalidade	Este protótipo	Trabalho de Carlassara (2009)	WebCam Monitor 5.24 de DeskShare Incorporated (2011)
Armazenamento das imagens		X	X
Compressão de vídeo	X	X	X
Detecção automática de novas câmeras	X		
Detecção de movimentos			X
Movimentação remota das câmeras			X
Transmissão através da internet	X	X	X
Transmissão de áudio			X
Transmissão em tempo real	X		X
Visualização de múltiplas câmeras simultaneamente	X		X
Visualização em dispositivo móvel		X	
Visualização no navegador	X		X

Quadro 37 – Comparativo entre ferramentas correlatas

4 CONCLUSÕES

Existem muitas ferramentas no mercado que possuem a intenção de monitorar áreas com câmeras além das apresentadas nesta monografia. No entanto, as tecnologias empregadas são das mais variadas. Este protótipo teve como foco utilizar a tecnologia para aplicações ricas de internet chamada de Silverlight, além de processar as imagens captadas efetuando compressão de vídeo e ainda *streaming* ao vivo para transmitir através da internet.

As aplicações de trabalhos correlatos e de ferramentas comerciais similares apresentados neste trabalho tornaram-se fonte de inspiração para o desenvolvimento e permitiram dar uma ideia mais clara dos objetivos a que se desejava alcançar.

Explorando o que a tecnologia Silverlight é capaz de fazer, desenvolveu-se uma aplicação que permite visualizar diversas câmeras simultaneamente, transmitindo suas imagens em tempo real, e possibilitando livre personalização para movimentação das janelas e redimensionamento das mesmas.

A utilização dos *frameworks* DirectShow e Windows Media Format SDK possui a importância de estas estruturas estarem presentes em grande parte das aplicações de mídia compatíveis com o sistema operacional Windows, o que reflete na farta literatura a seu respeito. Assim sendo, estudar estas tecnologias permite abranger a vasta quantidade de usuários deste sistema operacional.

O principal desafio em utilizar estas tecnologias através das bibliotecas de classes DirectShow.NET e WindowsMedia.NET foi integrá-las, de modo que parte dos procedimentos de captura, compressão e *streaming* efetuados na aplicação eram de uma e outra parte eram de outra. Soma-se a isso a dificuldade em adquirir informações visto que ambas as bibliotecas fazem parte de projetos paralelos da comunidade.

Optou-se por seguir as melhores práticas para a construção de sistemas orientados a objetos, portanto, o protótipo foi dividido em módulos. Utilizando esta estratégia, foi desenvolvida uma biblioteca com classes específicas para as funções centrais de captura, compressão e transmissão das imagens. Também foram desenvolvidas uma aplicação web onde estão a página a ser acessada e o serviço web responsável pela detecção e obtenção da lista de câmeras conectadas, e a aplicação rica de internet desenvolvida com o Silverlight.

Por ser um protótipo e não embutir todas as tecnologias conhecidas na área de captura, processamento de vídeo e monitoramento de câmeras, mesmo em função do tempo disponível para o seu desenvolvimento, esta aplicação pode ser utilizada para incremento de novas

possibilidades de funcionalidades como detecção de faces e/ou objetos, captação e transmissão de áudio, entre outras.

4.1 EXTENSÕES

Ao desenvolver este protótipo, verificaram-se várias possibilidades de extensão e melhorias desde a tarefa da captura de mídia até a interface utilizada. As sugestões levantadas durante o período de especificação e implementação podem ser vistas no Quadro 38.

Extensão
Criar um sistema de reconhecimento de faces a fim de detectar a presença de pessoas procuradas pela polícia.
Criar um sistema de reconhecimento de objetos a fim de detectar potenciais armas sendo transportadas por pessoas no ambiente monitorado.
Criar uma interface da aplicação Cliente utilizando a tecnologia HTML 5.
Criar uma interface da aplicação Cliente voltada para dispositivos móveis.
Incrementar funcionalidade de autenticação para acessar o sistema ou para cada câmera individualmente.
Incrementar funcionalidade de detecção de movimento nas imagens de cada câmera.
Incrementar funcionalidades que permitam a utilização da ferramenta para vídeo conferências.
Incrementar tecnologias de realidade aumentada para interagir com as imagens das câmeras e disponibilizar melhores informações para o usuário.
Permitir ao servidor capturar áudio obtido de microfones conectados e transmiti-lo para execução através da aplicação Cliente.
Permitir ao usuário compartilhar as câmeras conectadas no próprio computador.
Permitir controlar remotamente, através da aplicação Cliente, o ângulo de visão de cada câmera.
Permitir controlar, através da aplicação Cliente, o *zoom* da imagem produzida em cada câmera.
Permitir gravar, através da aplicação Cliente, as imagens obtidas de cada câmera.
Permitir utilizar recursos avançados presentes nas câmeras conectadas.

Quadro 38 – Possibilidades de extensão

REFERÊNCIAS BIBLIOGRÁFICAS

AUSTERBERRY, David. **The technology of video & audio streaming**. Burlington: Focal Press, 2005. Disponível em: <http://books.google.com/books?id=I3kiWVvO7WoC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 31 maio 2011.

BHASKARAN, Vasudev; KONSTANTINIDES, Konstantinos. **Image and video compression standards:** algorithms and architectures. 2nd ed. Norwell: Kluwer Academic Publishers, 1997. Disponível em: <http://books.google.com.br/books?id=dzOV8-sZE9cC&printsec=frontcover#v=onepage&q&f=false>. Acesso em: 31 maio 2011.

BUSCH, Marianne; KOCH, Nora. **Rich internet applications**: state-of-the-art. 2009. 18 f. Technical Report – Programming and Software Engineering Unit, Ludwig-Maximilians-Universität München, München. Disponível em: <http://uwe.pst.ifi.lmu.de/publications/maewa_rias_report.pdf>. Acesso em: 06 jul 2011.

CARLASSARA, Diogo. **Visualização de imagens capturadas em um circuito fechado de televisão (CFTV) no iPhone**. 2009. 69 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2010/341398_1_1.pdf>. Acesso em: 24 maio 2011.

CUBAS, Viviane O. **Segurança privada**: a expansão dos serviços de proteção e vigilância em São Paulo. São Paulo: Associação Editorial Humanitas: Fapesp, 2005. Disponível em: <http://books.google.com/books?id=cgv2flDoKwMC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 31 maio 2011.

DESKSHARE INCORPORATED. **WebCam Monitor 5.24**. [S.l.], [2011?]. Disponível em: <http://www.deskshare.com/lang/po/wcm.aspx>. Acesso em: 24 maio 2011.

DIRECTSHOW.NET. **DirectShowNet library**. [S.l.], [2010]. Disponível em: <http://directshownet.sourceforge.net/about.html>. Acesso em: 24 maio 2011.

FOLLANSBEE, Joe. **Hands-on guide to streaming media**: an introduction to delivering on-demand media. Burlington: Focal Press, 2006. Disponível em: <http://books.google.com/books?id=gEN1GbEgb5AC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 31 maio 2011.

GHODA, Ashish; SCANLON, Jeff. **Accelerated Silverlight 3**. New York: Apress, 2009. Disponível em: <http://books.google.com/books?id=jVwo11e1T94C&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 31 maio 2011.

GILL, Tricia. **An introduction to Windows Media Encoder 7.1**. [S.l.]: Microsoft Digital Media Division, 16 maio 2001. Disponível em: <http://msdn.microsoft.com/en-us/library/ms867158.aspx>. Acesso em: 24 maio 2011.

MACDONALD, Matthew. **Pro WPF in C# 2010:** Windows Presentation Foundation in .NET 4. New York: Apress, 2010. Disponível em: <http://books.google.com/books?id=nYl7J7z3KssC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 31 maio 2011.

MEREGE NETO, Jorge A. **Construção de um protótipo (hardware e software) para segurança predial através de monitoração via câmera digital e display gráfico**. 2004. 62 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2004/279149_1_1.pdf>. Acesso em: 24 maio 2011.

MICROSOFT. **Microsoft lança Silverlight 3 e Expression 3**. São Paulo, 2009. Disponível em: <http://www.microsoft.com/latam/presspass/brasil/2009/julho/silverlight3.mspx>. Acesso em: 24 maio 2011.

_____. **Silverlight for Symbian**. [S.l.], [2011a?]. Disponível em: <http://www.silverlight.net/getstarted/devices/symbian/>. Acesso em: 24 maio 2011.

_____. **Silverlight for Windows Phone**. [S.l.], [2011b?]. Disponível em: <http://www.silverlight.net/getstarted/devices/windows-phone/>. Acesso em: 24 maio 2011.

_____. **Silverlight toolkit samples**. [S.l.], [2010?]. Disponível em: <http://www.silverlight.net/content/samples/sl3/toolkitcontrolsamples/run/default.html>. Acesso em: 24 maio 2011.

MICROSOFT DEVELOPER NETWORK. **Code-behind and XAML**. [S.l.], [2011a?]. Disponível em: <http://msdn.microsoft.com/en-us/library/aa970568%28VS.90%29.aspx>. Acesso em: 24 maio 2011.

_____. **DirectShow**. [S.l.], 2010. Disponível em: <http://msdn.microsoft.com/en-us/library/dd375454%28VS.85%29.aspx>. Acesso em: 24 maio 2011.

_____. **MediaElement**. [S.l.], [2011b?]. Disponível em: <http://msdn.microsoft.com/en-us/library/bb980132%28VS.95%29.aspx>. Acesso em: 24 maio 2011.

_____. **Windows media format 11 SDK**. [S.l.], 2009. Disponível em: <http://msdn.microsoft.com/en-us/library/dd757738%28v=vs.85%29.aspx>. Acesso em: 24 maio 2011.

MICROSOFT TECHNET. **Internet Telephony with H.323**. [S.l.], [2011?]. Disponível em: <http://technet.microsoft.com/en-us/library/cc976949.aspx>. Acesso em: 07 jul 2011.

WEBCAM CORPORATION. **WebCam 1-2-3**. [S.l.], [2011?]. Disponível em: <http://www.webcam123.com/en/ezcast.html>. Acesso em: 24 maio 2011.