Faculdade de Engenharia da Universidade do Porto

Módulo para Gestão Documental na Solução *eResults* da CPC-HS

Joel Tiago Moreira Campos

Relatório de Projecto realizado no Âmbito do
Mestrado Integrado em Engenharia Informática e Computação

Orientador: Professor Jorge Alves da Silva

Julho de 2008

Módulo para Gestão Documental na Solução *eResults* da CPC-HS

Joel Tiago Moreira Campos

Relatório de Projecto realizado no Âmbito do
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Professor José Manuel Magalhães Cruz (Professor Auxiliar)

_____

Arguente: José Carlos Nascimento (Professor Auxiliar)

Vogal: Jorge Alves da Silva (Professor Auxiliar)

15 de Julho de 2008

Resumo

Este documento descreve o projecto Módulo para Gestão Documental na Solução *eResults*, desenvolvido na Companhia Portuguesa de Computadores – *Healthcare Solutions*.

Actualmente, no domínio hospitalar, a produção de exames auxiliares de diagnóstico é, tipicamente, auxiliada por diversas aplicações informáticas especializadas que funcionam localmente. No entanto, a transmissão dos resultados entre os diversos serviços hospitalares segue, depois, um processo frequentemente manual, com diversas lacunas, quer ao nível da rapidez com que este processo é realizado, quer ao nível da eficiência do mesmo, levando a que, muitas vezes, o paciente repita exames desnecessariamente. O propósito do *eResults* é facilitar o arquivamento, distribuição e divulgação dos resultados clínicos, tendo em vista a colmatação destas falhas.

Existindo já há alguns anos, o *eResults* necessita de uma actualização, não só para melhorar o aspecto da interface, como para melhorar as suas funcionalidades. O projecto que aqui se descreve tem como objectivo fundamental o desenvolvimento de dois módulos para o *eResults*, respectivamente, o Módulo de Apresentação de Resultados e o Módulo de Inserção de Conteúdos. Foi determinado que o Módulo de Apresentação de Resultados devia permitir visualizar os resultados em formato multimédia, dentro do próprio módulo e que, em ambos os módulos, deveria ser prestada especial atenção à questão da usabilidade, no sentido de a maximizar.

Por último, um outro objectivo do projecto é testar a viabilidade do uso da tecnologia *Microsoft Silverlight*, para a criação de uma *Rich Internet Application* (*RIA*) de âmbito empresarial.

Os resultados obtidos deixam perceber que a *Silverlight*, embora muito recente, apresenta já as capacidades necessárias para o desenvolvimento de *RIA's* completamente funcionais, sendo portanto viável o seu uso em ambiente empresarial.

Abstract

This document describes the project Documental Management Module for the *eResults* Solution, developed at *Companhia Portuguesa de Computadores* – Healthcare Solutions.

Currently, in the clinical domain, the production of diagnosis exams in the various departments is, typically, managed by specialized computer applications that work specifically within a given department. However, the distribution of the produced exams frequently follows a manual process, with several flaws, both on the speed at which the process is made and on its efficiency, leading to an often unnecessary submission of the patient to the repetition of some exams. The *eResults* purpose is to ease the effort of archival and distribution of the clinical results, thus aiming the solving of these flaws.

Existing since several years ago, *eResults* now needs to be updated, in order to improve the interface look and features. The project here described aims essentially to develop two modules for *eResults,* respectively, the Results Presentation Module and the Contents Insertion Module. It was decided that the Results Presentation Module should allow the visualization of the multimedia results inside the module itself and that special attention should be given to the usability of both modules.

Another aim of the project is to test the viability of the use of *Microsoft Silverlight* technology, in order to create a Rich Internet Application (RIA) of enterprise scope.

The obtained results allow to conclude that, although being very recent, *Silverlight* supplies the means for the development of fully functional RIA's, and that it can be used successfully in enterprise environments.

Definições, Acrónimos e Abreviaturas

CPC-HS	Companhia Portuguesa de Computadores – Healthcare Solutions
DOM	*Document Object Model*
HTML	*HyperText Markup Language*
Id	Identificador
IPSS	Instituições Particulares de Solidariedade Social
MCDT	Meio(s) complementar(es) de diagnóstico e terapêutica
MIEIC	Mestrado Integrado em Engenharia Informática e Computação
RAM	*Random Access Memory*
RIA	*Rich Internet Application*
SGBD	Sistema de Gestão de Base de Dados
URI	*Uniform Resource Identifier*
WPF	*Windows Presentation Foundation*
XAML	*Extensible Application Markup Language*
XML	*Extensible Markup Language*
UI	*User Interface*

Browser	*Software* usado para visualizar páginas *Web*
Cookies	Mensagem enviada a um *browser* por um servidor. O *browser* armazena a mensagem num ficheiro de texto, para que numa fase posterior o possa enviar de volta, quando tentar aceder a uma das páginas do servidor.
Streaming	Consiste na recepção e apresentação de dados de vídeo de forma contínua, possibilitando a sua reprodução mesmo antes da

	recepção dos dados ter sido terminada.
Web	Abreviatura de *World Wide Web*

Agradecimentos

Quero deixar aqui os meus agradecimentos a todos aqueles que de uma forma ou de outra possibilitaram o desenvolvimento deste projecto.

Começo por deixar um agradecimento ao Professor Jorge Alves da Silva pelas suas sugestões e esforço dedicado na orientação deste projecto.

Queria também deixar o meu agradecimento à Companhia Portuguesa de Computadores – *Healthcare Solutions*, pela oportunidade que me proporcionou de realizar este estágio, e à equipa de Investigação e Desenvolvimento *Microsoft* na qual fui inserido, particularmente ao Eng. Nuno Ribeiro e ao Eng. Nuno Mendes, pela orientação e apoio dado ao longo de todo o projecto.

Não poderia deixar de agradecer aos meus pais, irmãos e família, que sempre me apoiaram em toda a minha vida.

Tenho ainda que expressar a gratidão aos meus colegas por me proporcionarem tantos bons momentos durante o curso, vocês sabem quem são.

Por último, deixo ainda um agradecimento à Paula, pelo apoio e compreensão ao longo de todo o curso.

Sem vocês não teria chegado aqui,

Obrigado

Índice

ÍNDICE

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

LISTA DE TABELAS

Capítulo 1

Introdução

O presente documento tem o propósito de descrever as actividades realizadas e os produtos desenvolvidos no âmbito da disciplina de Projecto do Mestrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto (MIEIC-FEUP)**.** O projecto **Módulo para Gestão documental na Solução *eResults* da CPC-HS** consiste na especificação e desenvolvimento de um módulo de visualização e indexação de documentos clínicos, para integrar a aplicação de gestão documental *eResults* da Companhia Portuguesa de Computadores-*Healthcare Solutions* (CPC-HS).

Neste capítulo realiza-se a apresentação do contexto do projecto, descrevendo-se em linhas gerais a aplicação sobre a qual este incide, as fundamentações dos desenvolvimentos, e ainda o que se espera dos mesmos. Por último são descritos os temas abordados ao longo do relatório.

1.1 Âmbito

O projecto que aqui é descrito foi desenvolvido na CPC-HS. A empresa foca a sua actividade nas Instituições de Saúde em funcionamento em Portugal, apresentando já uma carteira de clientes vasta e diversificada, constituída por Hospitais Públicos, IPSS's, Hospitais Privados e Clínicas Médicas.

No domínio dos hospitais, verifica-se que actualmente grande parte dos serviços relacionados com os exames auxiliares de diagnóstico dispõe já de soluções informáticas para auxiliar o tratamento da informação produzida num exame. Essa informação é, no âmbito deste relatório, designada por "resultados".

Esses resultados podem assumir os mais diversos formatos, como por exemplo:

- Uma imagem obtida durante uma endoscopia
- Um vídeo de uma cirurgia
- Os resultados de uma análise clínica armazenados numa base de dados
- Um relatório de um exame

A CPC-HS apresenta várias soluções[1] no seu catálogo, abrangendo as mais diversas áreas e serviços clínicos, como Patologia Clínica, ou Radiologia, para citar algumas. No entanto, tal como acontece em tantos outros cenários empresariais, a divulgação dos conteúdos que aí são produzidos – no caso, resultados clínicos nos mais diversos formatos – segue, frequentemente, um circuito manual ou semi-automático. Este cenário tem diversas lacunas e causa graves problemas quer ao nível da rapidez com que os resultados são partilhados, quer da eficiência, uma vez que, em algumas situações, os exames de um paciente chegam a ser repetidos.

Identificado que foi este problema, a CPC-HS desenvolveu uma solução em ambiente *Web* de Arquivo Centralizado de Resultados Clínicos, o *eResults*, para facilitar o seu arquivamento, distribuição e divulgação junto dos intervenientes neste processo.

Existindo já há alguns anos, a sua interface é muito rudimentar, criando uma má experiência de utilização. Adicionalmente, suporta a apresentação de muito poucos tipos de dados, pelo que se determinou a necessidade do desenvolvimento de uma nova versão.

Este projecto está directamente relacionado com a solução *eResults*, da qual está a ser desenvolvida uma nova versão que será usada, numa fase inicial, no Centro Hospitalar de Vila Nova de Gaia/Espinho, EPE, contudo, prevê-se que o âmbito do

[1] Neste contexto, solução refere-se a um ou vários produtos do domínio informático, usados para a resolução de um problema

projecto venha, muito em breve, a ser alargado a outros produtos da empresa, tal como o *Processo Clínico*, sendo que as diligências para tal começaram já a ser tomadas no decorrer deste projecto.

1.2 Motivações

Actualmente, o aspecto e funcionalidades da solução *eResults* encontram-se desactualizados do ponto de vista da chamada *Web 2.0*, uma vez que a aplicação não tira proveito das capacidades das tecnologias e paradigmas de desenvolvimento que entretanto surgiram no mundo das *RIA* (*Rich Internet Applications*), e nas páginas *Web* em geral. Adicionalmente, um número razoável de necessidades de correcções e melhoramentos foram identificados. Em conjunto, estes factores determinaram a necessidade do desenvolvimento de uma nova versão do *eResults*. Nesta nova versão serão disponibilizadas as seguintes funcionalidades:

- Arquivamento/distribuição de documentos
- Linha de Histórico
- Fornecimento de resultados *On-Line* de soluções Laboratoriais da CPC-HS
- Integração com outros sistemas da CPC-HS
- Integração com sistemas não CPC-HS
- Monitorização e alertas centrados na Distribuição de Resultados
- Controlo de Acessos

Do interesse directo do desenvolvimento do projecto aqui apresentado é a funcionalidade de arquivamento/distribuição de documentos. Esta corresponderá a dois módulos, respectivamente, o Módulo de Apresentação dos Resultados e o Módulo de Inserção de Conteúdos.

1.3 Objectivos do Projecto

Este projecto, para além do desenvolvimento dos módulos de Apresentação dos Resultados e de Inserção de Conteúdos, visava também determinar a viabilidade da utilização da tecnologia *Silverlight* como meio para a criação de aplicações empresariais em ambiente *Web*, uma vez que esta nunca havia sido usada na empresa. A *Silverlight*

ainda se encontra em fase *Beta* e, como tal, não existe um historial sólido de casos de aplicação da mesma a nível empresarial que permita a realização de uma avaliação prévia do seu potencial.

O objectivo deste projecto é tirar partido das novas tecnologias de *RIA* da plataforma *.NET*, especificamente a tecnologia *Silverlight*, para desenvolver módulos capazes de realizar a apresentação de resultados dos exames clínicos e a indexação/armazenamento de informação que lhe está de alguma forma associada.

O Módulo de Apresentação dos Resultados, a ser desenvolvido, deve permitir que todo o tipo de resultado multimédia, como por exemplo imagens ou vídeo, seja tratado no próprio módulo, sendo da sua responsabilidade a disponibilização de meios para a interacção com o resultado. Por sua vez, no que respeita aos restantes tipos de resultados, caberá ao módulo determinar qual o tratamento que deverá ser feito para que estes sejam encaminhados para a aplicação que fará a apresentação mais adequada ao utilizador.

Por sua vez, o Módulo de Inserção de Conteúdos deve permitir definir diversa meta-informação, de acordo com o tipo de conteúdo e âmbito do documento a inserir.

Comum a ambos os módulos é a necessidade de proporcionar máxima usabilidade, pelo que é fundamental uma elevada capacidade de resposta da aplicação em tempo útil, com *feedback* simples e claro para o utilizador. Assim, operações como o carregamento de vídeos e imagens de grandes dimensões devem ser abordadas recorrendo a *streaming* e operações assíncronas, para que se consiga disponibilizar a usabilidade referida. Estas necessidades representam um desafio dado que o desenvolvimento dos módulos será feito em ambiente *Web*, e a cargo deste projecto ficará também a questão do tratamento do *streaming* do lado do servidor.

Pretende-se também que, para além da disponibilização dessas funcionalidades, o desenvolvimento seja feito de forma modular, tendo em vista o reaproveitamento dos módulos desenvolvidos para a sua reutilização em *Windows Presentation Foundation* (*WPF*), tirando partido das semelhanças existentes entre a *Silverlight* e a *WPF*.

1.4 Organização do Presente Relatório

Este relatório encontra-se dividido em sete capítulos e inclui ainda um anexo. Neste primeiro capítulo descreveu-se o contexto no qual o projecto se insere e os seus objectivos.

No segundo capítulo apresenta-se um estudo das tecnologias existentes no mercado para o desenvolvimento de *RIA's*, feito com o intuito de determinar aquela cujo uso traria mais vantagens ao desenvolvimento do projecto.

Por sua vez, no terceiro capítulo descrevem-se diversos aspectos do *eResults* que são de alguma forma relevantes para a descrição deste projecto, bem como o modo de funcionamento geral que é desejado nos módulos a desenvolver. A esta descrição segue-se o quarto capítulo onde se apresenta a especificação dos requisitos da solução.

No quinto capítulo apresentam-se pormenorizadamente os módulos desenvolvidos, sem referência a qualquer detalhe de implementação.

No sexto capítulo descreve-se a estrutura dos módulos desenvolvidos focando a interacção entre os diversos componentes que os compõem, e são apresentados os principais detalhes de implementação da solução.

Por último, apresentam-se as conclusões do projecto, resumindo o trabalho efectuado, descrevendo as principais dificuldades encontradas e apontando aspectos a melhorar nos módulos, em desenvolvimentos futuros.

Em Anexo é feita a descrição do formato *XML* usado, pelo Módulo de Inserção de Conteúdos, para o envio ao *WebService ER2Indexer* das meta-informações associadas a um documento.

Capítulo 2

Revisão Tecnológica

O problema que aqui se estuda é um problema de desenvolvimento de *RIA*'s. Muito embora o projecto possua outras vertentes, a sua vertente fundamental é esta e, como tal, será esta a receber o principal foco ao longo deste capítulo.

Uma *RIA* é uma aplicação *Web* que apresenta as funcionalidades de aplicações locais tradicionais. Tipicamente uma *RIA* transfere o processamento da interface do lado do utilizador para o do cliente, mas mantém os diversos dados necessários ao seu funcionamento no servidor.

Este modo de funcionamento surge em oposição ao modo tradicional de funcionamento das aplicações *Web*. No modo tradicional, as comunicações são realizadas de acordo com uma arquitectura cliente-servidor, em que o servidor realiza todo o processamento e o cliente se limita a apresentar conteúdo *HTML* estático. Numa *RIA*, porém, as aplicações usam uma tecnologia específica do lado do cliente, permitindo que o processamento de dados se realize deste lado. É assim eliminada muita da latência causada pelas constantes trocas de dados e carregamentos de páginas entre cliente e servidor.

A eliminação de latência decorre da transformação de operações que numa arquitectura tradicional seriam síncronas, em operações assíncronas, ou da sua completa transferência para o lado do cliente, dando ao utilizador final um *feedback* mais rápido, se não imediato. Adicionalmente, surgem vantagens como a redução do esforço de

instalação da aplicação, que muitas das vezes passa a ser nulo dependendo da tecnologia *RIA* usada. Verifica-se ainda que, com a disponibilização da aplicação num *browser*, qualquer computador com ligação à internet pode servir como ponto de acesso à aplicação e que o esforço de migração da aplicação para o funcionamento em diferentes sistemas operativos é irrisório ou mesmo nulo, quando comparado com aplicações nativas.

A estas vantagens, com a segunda geração das *RIA*'s, juntou-se o alargamento da capacidade do programador para criar aplicações muito mais interactivas e poderosas em ambiente *Web*, apresentando mesmo um aspecto em tudo semelhante ao de aplicações nativas.

Contudo, existem também diversas desvantagens do uso das *RIA*'s. A mais significativa será que, por questões relacionadas com a manutenção da segurança do computador e privacidade do utilizador, certas funcionalidades são limitadas nas *RIA*'s, como, por exemplo, o acesso aos ficheiros locais no cliente, o que consequentemente limita as capacidades das aplicações. Há ainda a exigência típica de que o cliente permita *scripting* para o funcionamento das aplicações.

Também relevante é o facto de que, apesar das comunicações entre cliente e servidor serem significativamente reduzidas, o tamanho da página com as informações necessárias ao funcionamento da *RIA* é aumentado, o que se reflecte inevitavelmente no tempo de carregamento da página.

Por último, verifica-se ainda a necessidade da existência de uma ligação à internet/rede. Muito embora algumas tecnologias permitam o funcionamento *offline*, com sincronizações ocasionais, este modelo não se adequa a todos os cenários.

2.1 Tecnologias Adequadas ao Desenvolvimento da Solução

O estudo das tecnologias adequadas ao desenvolvimento da solução fica condicionado à partida pelas restrições impostas pela empresa onde o produto foi desenvolvido. Neste caso em particular, as opções de desenvolvimento estavam

determinadas à partida. A empresa já havia escolhido a *Microsoft Silverlight* como a opção de desenvolvimento, pelo que esta é apresentada em maior detalhe.

Existe já um número considerável de tecnologias que suportam o desenvolvimento de *RIA*'s, contudo, as que serão aqui referidas são aquelas com uma percentagem significativa de adopção no mercado: a *Adobe Flash* e suas congéneres, as *Java Applets*, a *JavaScript* e, como não podia deixar de ser, a *Microsoft Silverlight*.

2.1.1 *Microsoft Silverlight*

Com cerca de um ano de vida, a sua entrada no mercado, embora lenta, tem sido progressiva. A *Silverlight* veio dar à área das *RIA* alguma vivacidade e aumento da concorrência, dado que por muitos anos a *Flash* foi, e ainda é, a tecnologia soberana nesta área.

Silverlight é um *plugin* multi-*browser*, multi-plataforma e multi-dispositivo que permite criar aplicações *Web* com uma riqueza inigualável usando *HTML* padrão. Tal deve-se ao seu amplo suporte de áudio, vídeo e imagens, aos meios que fornece para a realização de animações, e ainda à grande variedade de controlos para a interface do utilizador disponibilizados, que proporcionam as condições para a criação de uma *RIA* com grande suporte multimédia.

Por ser um subconjunto reduzido das funcionalidades de uma outra tecnologia da mesma empresa, a *WPF*, as semelhanças entre estas duas tecnologias são enormes.

Para que uma aplicação desenvolvida em *Silverlight* possa funcionar num qualquer computador é necessário proceder à instalação do seu *plugin*. A *Microsoft* definiu que o tamanho do instalador do *plugin* da *Silverlight* deveria ser tão reduzido tanto quanto possível e de muito fácil instalação, para que a adopção da tecnologia pelos utilizadores finais seja facilitada.

Daqui resulta que a *Silverlight* é uma *lightweight class library*, mas, se excluirmos as restrições impostas na *Silverlight* que são maioritariamente fruto de questões de segurança e privacidade, a grande diferença entre a *Silverlight* e a *WPF*, reside no facto de não serem disponibilizadas tantas funcionalidades para o programador.

Abordando a questão da caracterização multi-*browser* e multi-plataforma da *Silverlight*, verifica-se que existem diversos sistemas operativos e *browsers* que ainda não são suportados, como por exemplo o *Linux* e o *Opera*, respectivamente.

Contudo, tal não constitui um problema para o projecto a ser desenvolvido dado que este foi contratualizado para um ambiente de sistema operativo *Microsoft Windows* e *browser Internet Explorer 6* ou superior.

Algumas das principais capacidades do *Silverlight* são:

- Capacidade de reprodução de *WMV – Windows Media Video*, *WMA – Windows Media Audio* e *MP3 – MPEG-1 Audio Layer 3*
- Na versão 2.0, o código do programa pode ser escrito em qualquer linguagem da *Framework .NET*
- Suporte para *streaming* de ficheiros audiovisuais
- Mais de 30 controlos nativos para a interface
- Programação assíncrona
- Manipulação do *DOM* do *HTML* a partir do *managed code*
- *Deep Zoom*[2]
- Acesso controlado ao sistema de ficheiros do utilizador e *isostorage* que permite guardar, mediante a autorização do utilizador, uma certa quantidade de dados fora da *cache*, no computador do cliente
- Desenvolvimento suportado pelo *Microsoft Visual Studio 2008* e *Microsoft Expression Blend*, sendo o segundo mais direccionado a *designers*.

2.1.1.1 Modelo de Funcionamento

Convém referir que nesta secção visa-se explicar apenas o modo de funcionamento de uma aplicação em *Silverlight*, pelo que muitos dos conceitos referidos na descrição que se segue são detalhados somente na secção seguinte.

Uma aplicação *Silverlight* é composta por um ou vários *user controls*. Um *user control* é uma classe que quando instanciada pode ser encarada como o conjunto de um

[2] A *Deep Zoom* é uma nova tecnologia para uso em *Silverlight* que permite a navegação e a ampliação de imagens de alta resolução, sem ser necessário o carregamento completo da imagem no cliente

objecto visual e da lógica que lhe está associada, e que o programador pode reutilizar como entender. Para criar um *user control* um programador criará um ficheiro *XAML* ao qual estará associado um outro denominado de *code-behind*. Tipicamente o ficheiro *XAML* irá especificar o *layout* e o aspecto da página, e o ficheiro de *code-behind* servirá para programar a interacção do *user-control*. É possível que um *user control* use na sua definição outros *user controls*.

Para todas as aplicações *Silverlight* existe uma classe especial responsável pelo arranque da aplicação, vulgarmente denominada *App,* cuja responsabilidade é definir, no arranque da aplicação, qual será o *user control* que será instanciado.

Quando uma aplicação de *Silverlight* é compilada, o resultado da compilação é comprimido e armazenado num ficheiro de extensão ".xap". Para inserir a aplicação numa página *Web*, basta incluir nela um *tag object*, apontando para o ficheiro ".xap". Ao ser carregada no cliente, a página fará com que o *plugin* do *Silverlight* descarregue o ficheiro e instancie a aplicação nessa mesma página.

2.1.1.2 Modelo de Programação

A *Silverlight* preconiza dois modelos de programação, um que usa *managed code*, outro que usa *unmanaged code*. A diferença entre estes dois tipos de código reside no facto do *managed code* correr sobre uma máquina virtual, fornecendo simultaneamente maior segurança e versatilidade.

Como o *unmanaged code* é usado apenas na versão 1 da *Silverlight*, e a versão usada no desenvolvimento do projecto foi a 2, apresentarei apenas a segunda versão.

A programação em *Silverlight* assenta na *Extensible Application Markup Language* (*XAML*). A *XAML* é uma linguagem declarativa baseada em *XML,* cujo principal objectivo é definir os elementos e *layout* da interface do utilizador.

Os objectos que são declarados em *XAML* são directamente mapeados para uma instância de um objecto e os conjuntos de objectos são definidos como árvores de objectos. Associado ao ficheiro de *XAML* existe tipicamente um ficheiro de *code-behind,* escrito numa qualquer linguagem da *Framework .NET,* onde se realizam operações de resposta a eventos e manipulações de objectos declarados em *XAML* [2].

Um ficheiro de *code-behind* refere-se a um ficheiro que contém o código que complementa o código criado quando um ficheiro de *XAML* é compilado para uma aplicação [3].

A figura 2-1 apresenta um exemplo simples do conteúdo de um ficheiro escrito em *XAML*.

```
<UserControl x:Class="Cpchs.Examples.HelloWorld"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid >
        <StackPanel>
            <TextBlock x:Name="firstLabel" Text="Hello"/>
            <TextBlock x:Name="secondLabel" Text="World!"/>
        </StackPanel>
    </Grid>
</UserControl>
```

Figura 2-1 – Ficheiro XAML de exemplo

Como se pode notar, este *UserControl* definido em *XAML*, contém um objecto raiz, no caso uma *Grid*. Por sua vez, este poderá conter diversos objectos filhos. Esses filhos podem também, consoante o seu tipo, conter outros objectos, constituindo-se assim um conjunto de objectos na forma de uma árvore de objectos, tal como foi referido.

Do leque de linguagens disponíveis para a escrita do *code-behind* optou-se pelo uso da *C#*. O ficheiro de *code-behind* correspondente ao ficheiro *XAML* apresentado na Figura 2-1 seria, na sua versão mais simples, o que se apresenta na Figura 2-2.

```
using System.Windows.Controls;

namespace Cpchs.Examples
{
    public partial class HelloWorld : UserControl
    {
        public HelloWorld ()
        {
            InitializeComponent();
        }
    }
}
```

Figura 2-2 – Exemplo de code-behind, em C#

No conteúdo deste ficheiro poderiam ser efectuadas as mais diversas alterações aos objectos declarados no ficheiro *XAML*. Se, por exemplo, se desejasse alterar o

conteúdo da primeira *label* que no *XAML* aparecia como **Hello** para **Silverlight** bastaria acrescentar no construtor:

```
firstLabel.Text = "Silverlight";
```

Como nota final sobre este tópico, refira-se que a *Silverlight* tem versatilidade suficiente para que os objectos declarados no *XAML* sejam gerados exclusivamente no ficheiro de *code-behind*, contudo esse não é o modo típico de execução, e salvo casos pontuais não traz qualquer vantagem ao programador. Adicionalmente, cria-se deste modo uma separação entre a camada de apresentação e a camada lógica.

2.1.1.3 Animações

O suporte que a *Silverlight* apresenta para a criação de animações em objectos visuais auxilia em grande medida o trabalho do programador nessa tarefa, disponibilizando, para isso, diversos métodos.

De forma a generalizar o seu uso e facilitar a sua aplicação a qualquer objecto, as animações disponibilizam mecanismos para indicar o objecto e a propriedade desse objecto sobre a qual a animação irá ser realizada. Assim o programador pode muito facilmente animar um objecto, variando a sua largura, a sua posição, ou qualquer outra propriedade desde que esta seja representada por um valor numérico. Uma vez iniciada, a animação encarregar-se-á de fazer variar o valor da propriedade do objecto que lhe foi indicada, ao longo de um período de tempo também ele indicado, criando-se assim a animação visual do objecto.

O modo mais complexo de criação de animações, disponibilizado pela *Silverlight*, permite a criação de animações que reproduzem o comportamento de objectos físicos, como por exemplo aceleração e desaceleração. Este método usa uma s*pline*[3] para a definição da animação.

Todas as *spline's* usadas em *Silverlight* usam uma curva de *Bézier* definida com o ponto de início (0,0) e ponto de fim (1,1), deixando ao cargo do programador a definição dos pontos de controlo [4]. A curva de *Bézier* assim produzida definirá a

[3] Uma *spline* refere-se a uma curva definida, por troços, através de funções polinomiais

forma como as animações irão variar, mapeando-se a curva produzida para a duração e valores de início e fim da mesma. A inclinação da curva da *spline* em cada momento, definirá a aceleração com que a animação varia, como tal, uma grande inclinação na curva reflectirá uma grande variação na animação e uma pequena inclinação traduzir-se-á numa pequena variação.

Para ilustrar este processo temos a Figura 2-3. Na figura ilustra-se a *spline* com os pontos de controlo (0,1) e (1,0), assinalados, respectivamente, com as cores vermelha e azul. Uma animação produzida com esta *spline* durante 10 segundos, variaria muito rapidamente a propriedade animada nos primeiros 3 segundos, depois muito lentamente durante 4 segundos, voltando a variar muito rapidamente nos últimos 3 segundos.

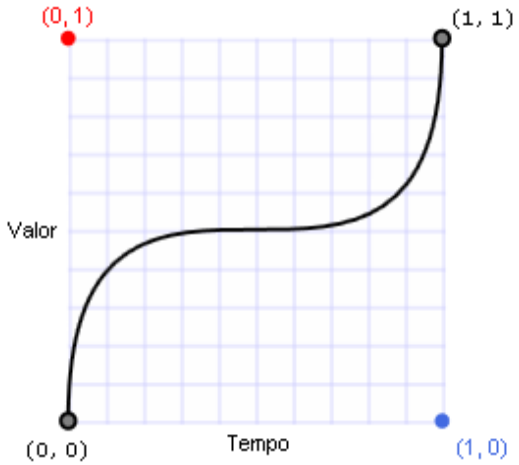

Figura 2-3 – Ilustração de uma spline criada em Silverlight

2.1.2 *Adobe Flash*

A *Adobe Flash* é uma tecnologia multimédia que surgiu em 1996 pela mão da *Macromedia* e foi entretanto adquirida pela *Adobe*. A *Flash* tornou-se muito popular para inserir conteúdos animados ou aplicações com grande interactividade em páginas *Web*. No passado mais recente, as suas capacidades estenderam-se ao suporte de criação de *RIA*'s e *streaming* de áudio e vídeo.

Para poderem visualizar conteúdos desenvolvidos em *Flash*, os clientes têm que possuir o *plugin Adobe Flash Player* instalado no seu *browser*. A sua instalação é simples e rápida, facto que contribui para que, segundo a *Adobe*, em Março de 2008 o *Adobe Flash Player* esteja instalado em 98.8% dos computadores com acesso à internet,

tornando a *Flash* na plataforma de *software* mais difundida do mundo. Para isso muito contribui que a mesma seja suportada na maioria dos sistemas operativos e *browsers*. A lista de sistemas operativos suportados inclui diversas versões do *Microsoft Windows*, *Mac OS* e *Linux*. A *Flash* é ainda suportada em alguns telemóveis, através do *Adobe Flash Lite* [12].

A programação da *Flash* é feita em *ActionScript*, uma linguagem de programação com a mesma sintaxe que o *JavaScript* mas que pertence a uma *framework* completamente distinta.

Um uso da *Flash* que se tem difundido muito é a apresentação de vídeos em páginas *Web*. O sucesso que a Flash tem tido com esta funcionalidade tem mais que ver com a difusão que a tecnologia tem em diferentes *browsers* e sistemas operativos do que com a qualidade do vídeo que apresenta.

Uma das principais críticas que se faz a esta tecnologia é a grande capacidade computacional necessária, que faz com a mesma não apresente um bom desempenho em máquinas desactualizadas. São também feitas críticas à sua usabilidade, acessibilidade, segurança, e à impossibilidade dos motores de pesquisa indexarem os conteúdos produzidos nesta tecnologia.

A *Adobe* possui dois produtos específicos para o desenvolvimento de *RIA*'s baseados na *Flash*, nomeadamente o *Adobe Air* e o *Adobe Flex*.

2.1.2.1 *Adobe Flex*

O *Adobe Flex* surgiu para combater a dificuldade que os programadores tradicionais sentiam em adaptar-se ao modelo de animação baseado em *frames* da tecnologia *Flash*. Para tal foi criado um modelo de programação mais familiar a estes programadores, usando *MXML* e *ActionScript.* A *MXML* é uma linguagem declarativa baseada em *XML* para a construção de *user interfaces* (*UI*'s); *ActionScript* é a linguagem usada para a programação da interacção.

O *Software Developer Kit* (*SDK*) da *Flex* vem munido de vários controlos como botões ou *data grids* para auxiliar o programador na construção da interface.

2.1.2.2 *Adobe Air*

O *Adobe Air* permite a construção de aplicações usando *Adobe Flex, HTML* e *Ajax* e já se distancia um pouco das *RIA*'s mais comuns, uma vez que o seu objectivo é permitir reutilizar aplicações *Flash* para desenvolver aplicações nativas.

O funcionamento do *Air* é o de uma máquina virtual que funciona de forma independente dos *browsers,* permitindo o desenvolvimento de *RIA*'s que podem ser implementadas como aplicações nativas. Esta aproximação ao desenvolvimento de *RIA*'s acarreta a necessidade de instalação da aplicação no computador dos utilizadores, o que, apesar de fornecer acesso ilimitado ao sistema de ficheiros do computador e permitir o funcionamento da aplicação em modo desligado da rede, elimina muita da versatilidade que sempre caracterizou as *RIA*'s.

2.1.3 *Java Applets*

Uma *applet* é um componente de *software* que corre no contexto de um outro programa. No caso de uma *Java Applet* trata-se de uma aplicação *Java* a correr dentro de um *browser*. Para tal, o utilizador final necessita de instalar uma *Java Virtual Machine* no seu computador. As *Java applets* são escritas na linguagem de programação *Java*. Uma das principais características desta linguagem é a sua independência da plataforma, o que se reflecte na diversidade de sistemas operativos e *browsers* que suportam esta tecnologia.

Contudo, esta tecnologia apresenta diversas lacunas do ponto de vista do seu uso para *RIA*'s, como a dificuldade de instalação da *Java Virtual Machine*, os tempos de carregamento das *applets* no *browser*, o fraco suporte para multimédia e a dificuldade em permitir o acesso da *applet* ao sistema de ficheiros[4] são algumas das principais.

Para colmatar algumas destas falhas foi desenvolvida a *Java Web Start* que permite correr aplicações, lançadas directamente de páginas *Web* numa máquina virtual, independente dos *browsers,* com menos restrições, nomeadamente no acesso ao sistema de ficheiros. Contudo, para que as aplicações desenvolvidas nesta tecnologia possam ser

[4] Por omissão é impossível às *applets* acederem ao sistema de ficheiros, mesmo que de forma limitada. Embora existam formas de contornar esta limitação, trata-se de um processo moroso.

usadas é necessário descarregá-las e instalá-las. Algumas das funcionalidades adicionais desta tecnologia são a possibilidade de correr aplicações em modo desligado e a disponibilização de actualizações automáticas das mesmas, quando o utilizador se conecta à rede.

2.1.4 *JavaScript*

A *JavaScript* é uma linguagem de *scripting* de uso muito frequente para programação do lado do cliente, no desenvolvimento de aplicações *Web*. Caracteriza-se por ser dinâmica e fracamente tipificada, e foi desenhada para ter uma sintaxe semelhante à da *Java*, mas mais simples, para que os que não são programadores a pudessem usar. É suportada sem a necessidade de qualquer instalação pela maioria dos *browsers* actuais.

O uso mais frequente da *JavaScript* é a escrita de funções que são incluídas em páginas *Web* e que interagem com o *DOM* da página. A grande vantagem do seu uso resulta da sua capacidade de gerar respostas imediatas às acções do utilizador, o que não é possível fazer em simples *HTML*.

A *JavaScript* foi a primeira tecnologia para clientes capaz de correr código do lado do cliente em arquitecturas cliente-servidor suportada pela maioria dos *browsers*. Embora muito limitada nos seus primórdios, com o aparecimento do conceito de *Dynamic HTML* – uma página *HTML* em que uma linguagem de *script* altera conteúdos da página, dinamizando o que de outra forma seria *HTML* estático – e o recurso a bibliotecas de *software*, é possível usá-la para o desenvolvimento de uma *RIA*.

O uso destas técnicas de desenvolvimento está associado ao conceito de *Asynchronous JavaScript and XML* (*Ajax).* Apesar de ser muito difícil criar aplicações de grande dimensão com *Ajax*, o seu uso é cada vez mais usual, facto facilitado pelas diversas *frameworks* de *Ajax* que entretanto foram desenvolvidas.

2.2 Sumário do Estudo

Após a realização do estudo, ficou claro que qualquer uma das tecnologias consideradas representa uma alternativa viável para o desenvolvimento de *RIA*'s.

Contudo é necessário levar em linha de conta as necessidades da aplicação a desenvolver e não apenas o universo genérico das *RIA*'s. Como tal, a primeira alternativa a ser rejeitada teria de ser o uso da *Ajax*, uma vez que de entre as tecnologias consideradas é a que apresenta um suporte mais limitado a meios audiovisuais e animações, algo que é amplamente desejado para o desenvolvimento dos módulos do projecto.

Outra alternativa seria o uso de *Java Applets*. Apesar de estas já suportarem animações, o seu suporte de multimédia é muito rudimentar, e as dificuldades que apresentam na instalação do *plugin* e carregamento das aplicações são incontornáveis e não desejáveis para o projecto a desenvolver. O facto do acesso ao sistema de ficheiros ser muito limitado cria ainda um impedimento ao módulo de inserção de conteúdos, que necessita dessa funcionalidade para poder desempenhar o seu propósito.

Em relação à *Adobe Flash* fica claro que se trata da tecnologia que apresenta, para já, o maior rol de funcionalidades, a que se junta a maturidade de uma tecnologia com provas dadas no mercado. Muito embora as capacidades da *Adobe Air* sejam extensas, estas afastam-se dos requisitos fundamentais do projecto, devido à necessidade de instalação das aplicações, mesmo que essa instalação seja feita a partir da *Web*. Por sua vez, o suporte que a *Adobe Flex* apresenta, tanto para controlos da *UI* como para as funcionalidades que têm mais que ver com animações e interactividade herdadas da *Flash*, torna-a numa tecnologia que encaixa perfeitamente nas necessidades do projecto.

Efectuando uma comparação entre a *Adobe Flex* e a *Microsoft Silverlight*, poder-se-á dizer que as funcionalidades que uma e outra apresentam são muito semelhantes pelo que, qualquer uma delas poderia ter sido utilizada neste projecto, mas este não é o único factor a levar em conta.

Analisando outros factores, a favor da *Flex* temos o facto de se tratar de uma tecnologia estável, contrariamente à *Silverlight*, que de momento ainda se encontra em fase beta, e em grande evolução mas que eventualmente terá também uma versão final.

Por outro lado, a *Silverlight* já apresenta suporte ao seu desenvolvimento no *Microsoft Visual Studio* e grande integração na plataforma *.NET*, integração essa que tem tendência para aumentar ainda mais no futuro. Este facto, por si só, apresenta

enormes benefícios para empresas que, como a CPC-HS, usam um grande leque de tecnologias da *Microsoft*, privilegiando claramente a facilidade de integração. Este facto é tão importante que foi o que levou a empresa a optar pela *Silverlight* como opção de desenvolvimento, sem necessitar de efectuar um estudo aprofundado.

Capítulo 3

Descrição do Problema

3.1 *eResults*

Para perceber melhor as necessidades do módulo a desenvolver, importa esclarecer melhor o modo de funcionamento do *eResults*. Como uma ferramenta de gestão documental, o objectivo do *eResults* é centrar numa única aplicação os dados relativos a resultados de exames, oriundos das diferentes aplicações que actuam nas diversas áreas do hospital onde se realizam exames auxiliares de diagnóstico. Estas aplicações tratam e podem armazenar os resultados produzidos localmente, e não têm de ser aplicações cujo desenvolvimento esteja a cargo da CPC-HS. Cada aplicação é responsável pelo envio dos resultados que produz para o *eResults*.

O *eResults* é acessível a partir de qualquer ponto da rede interna do hospital, dado que funciona em ambiente *Web*. Assim, qualquer médico, desde que autenticado, pode facilmente consultar o resultado de exames actuais ou antigos do seu paciente, consoante o seu interesse.

3.1.1 Conceitos

Nesta secção explicam-se alguns conceitos importantes para se perceber o modo de funcionamento dos módulos desenvolvidos.

- **Doente/Utente/Paciente** – pessoa que foi tratada/consultada no hospital

- **Episódio/Visita –** compreende todas as actividades a que um doente é submetido num determinado período de tempo; essas actividades podem ser por exemplo, uma consulta de medicina geral ou uma cirurgia
- **Nº Doente** – identificador único atribuído a cada doente, pela aplicação
- **Nº Processo** – tem um papel muito semelhante ao número de doente e é também um identificador único; a sua necessidade decorre da existência de situações em que o nº de doente pode não existir, como, por exemplo, numa entrada de um doente na urgência
- **Nº SNS –** número do Serviço Nacional de Saúde
- **Documento –** no domínio do *eResults* um documento não se refere necessariamente a um documento no sentido corrente; qualquer resultado que seja inserido no *eResults* tem associado um documento, pelo que um documento pode referir tanto um relatório de um exame como uma imagem obtida num exame
- **Nº Documento** – identificador do documento no *eResults*
- **Identificador Externo –** um identificador externo é o identificador do documento na aplicação que o produziu
- **Validação –** a validação corresponde ao momento em que um qualquer resultado é validado na aplicação onde foi produzido; o acto de validação varia consoante a aplicação onde ocorre, mas, tipicamente, o acto corresponde à verificação e aceitação dos resultados do exame por uma pessoa a quem essa responsabilidade é delegada
- **Aplicação de Origem** – refere-se à aplicação na qual o documento foi criado
- **Serviço Requisitante** – refere-se ao serviço que requisitou a realização de um determinado exame auxiliar de diagnóstico
- **Serviço Executante** – refere-se ao serviço que realizou um determinado exame auxiliar de diagnóstico

3.1.2 Arquitectura Física

O *eResults* poderá apresentar diversas configurações ao nível da sua arquitectura física. Na figura 3-1 esquematiza-se aquela que será a mais usual.

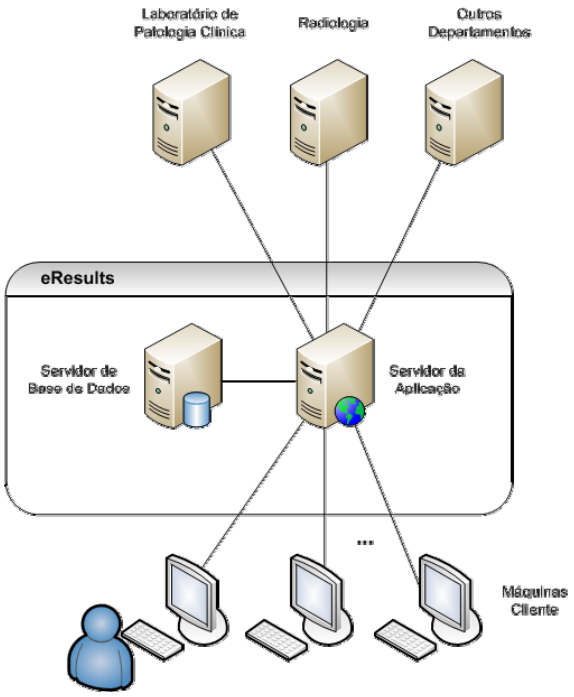

Figura 3-1- Diagrama da arquitectura física

Os constituintes físicos do sistema são:

- **Máquinas das diversas áreas hospitalares** – máquinas onde os resultados dos exames são obtidos e comunicados ao *eResults*
- **Servidor da aplicação** – máquina que fornece o acesso às páginas do *eResults* e aos *WebServices* que serão usados para o funcionamento dos módulos a desenvolver; o papel dos *WebServices* na aplicação é descrito em maior detalhe na secção 6.1.1
- **Servidor de Base de dados** – máquina que fornece o acesso ao SGBD da aplicação
- **Máquina do Cliente** – máquina pela qual o utilizador acede à aplicação

3.1.3 Actores do Sistema

Muito embora os utilizadores primordiais do *eResults* sejam médicos, estes não serão os únicos actores no sistema. A lista completa de utilizadores é a seguinte:

- **Funcionário da secção de informática do hospital** – é o administrador do sistema; o seu papel é, fundamentalmente, o da criação de utilizadores, mas na qualidade de administrador acaba por ter acesso a todas as funcionalidades.

- **Administrativo** – funcionário cujo papel no sistema depende completamente das permissões de acesso que o médico responsável lhe atribuir
- **Médico –** tem acesso a todas as informações dos seus pacientes e pode permitir o acesso de administrativos à aplicação, usando funcionalidades específicas para tal, pode ainda realizar a inserção de resultados na aplicação
- **Director de serviço** – é um médico que tem acesso às informações de todos os pacientes do serviço pelo qual é responsável.

A existência destes tipos de utilizadores acarreta a necessidade da tomada de diversos cuidados, ao nível do controlo de acessos aos diversos módulos do *eResults*, contudo, esse controlo sai do âmbito deste projecto, sendo realizado num módulo completamente independente dos módulos desenvolvidos neste projecto.

3.2 Descrição Geral dos módulos

3.2.1 Módulo de Apresentação dos Resultados

O papel do Módulo de Apresentação dos Resultados é o de permitir que o utilizador consulte os resultados de exames.

Para especificar os resultados que pretende visualizar, o utilizador tem ao seu dispor diversos campos de pesquisa. Estes subdividem-se em dois grupos, os campos de pesquisa simples e os campos de pesquisa avançada, e são enumerados de seguida:

- Pesquisa Simples
  - Nome Doente
  - Nº Doente
  - Processo
  - Nº SNS
- Pesquisa avançada
  - Data Nascimento
  - Sexo
  - Tipo Episódio
  - Episódio
  - Id Externo

- Nº Documento
- Tipo de Documento
- Aplicação
- Datas do Documento/Episódio/Validação

A pesquisa funciona a dois níveis. No primeiro nível determina-se o doente cujos dados se pretende consultar; no segundo, determinam-se quais os resultados desse doente que serão apresentados. Uma vez que apresentar toda a informação associada a um resultado não traria vantagens práticas, por ser demasiada, para escolher qual a informação que deveria ser apresentada para cada resultado, foram consultados os utilizadores chave da aplicação e determinou-se que a informação apresentada seria:

- Id do documento
- Tipo de Episódio
- Data do documento
- Serviço Requisitante
- Serviço Executante

A apresentação de resultados pode ser feita numa de duas formas: através de uma listagem simples ou através de uma grelha de caixas, onde cada caixa corresponde a um resultado. Qualquer um dos modos de apresentação conterá os dados enumerados previamente, mas o segundo conterá também uma imagem de antevisão do resultado. Cada um dos resultados pode, posteriormente, ser visualizado com maior detalhe, usando diferentes abordagens, consoante o tipo de resultado.

3.2.1.1 Carregamento de Ficheiros

Uma vez que os resultados, e as imagens de pré-visualização dos resultados, estão armazenados na base de dados, e que a *Silverlight*, sendo uma tecnologia que funciona no cliente, não pode efectuar operações directamente sobre a base de dados, o carregamento de ficheiros no cliente representa um problema porque tem de ser mediado pela máquina aplicacional do *eResults*.

Mediante a arquitectura física do *eResults* já apresentada, ao realizar as operações de carregamento de dados da forma tradicional, esta processar-se-ia da seguinte forma:

1. Cliente faz o pedido ao servidor aplicacional
2. Aplicação faz interrogação à base de dados
3. Base de dados determina e devolve os resultados
4. Resultado da interrogação é totalmente carregado na aplicação
5. Resultado é enviado ao cliente

A latência resultante da realização do ponto 4 será tanto maior quanto maior o volume de dados que devem transitar entre aplicação e base de dados. Se os dados a serem transmitidos forem imagens ou vídeos aquele volume pode atingir valores na ordem das centenas de megabytes, o que acarretaria, segundo testes realizados na empresa, latências de dezenas de segundos, mesmo usando máquinas muito rápidas, ligadas por redes com velocidades de 10 megabit/segundo. Por isso, este processo não se adequa à realização de operações de carregamento de ficheiros em *streaming.*

Impõe-se, portanto, que os carregamentos dos ficheiros no cliente se realizem sem que uma transmissão completa dos resultados ocorra entre base de dados e aplicação, antes da transmissão para o cliente. Como tal, o processo realiza-se da seguinte forma:

1. Cliente faz o pedido ao servidor aplicacional
2. Aplicação faz interrogação à base de dados
3. Base de dados determina e devolve um apontador para o resultado
4. Aplicação realiza um ciclo em que obtém os dados do ficheiro em pequenas partes e os vai enviando para o cliente até acabar de os ler

Esta abordagem ao problema reduz a latência da interrogação para valores aceitáveis, permitindo a realização da transmissão do ficheiro de forma virtualmente directa, entre a base de dados e o cliente.

Os dois modos de funcionamento são esquematizados da Figura 3-2.

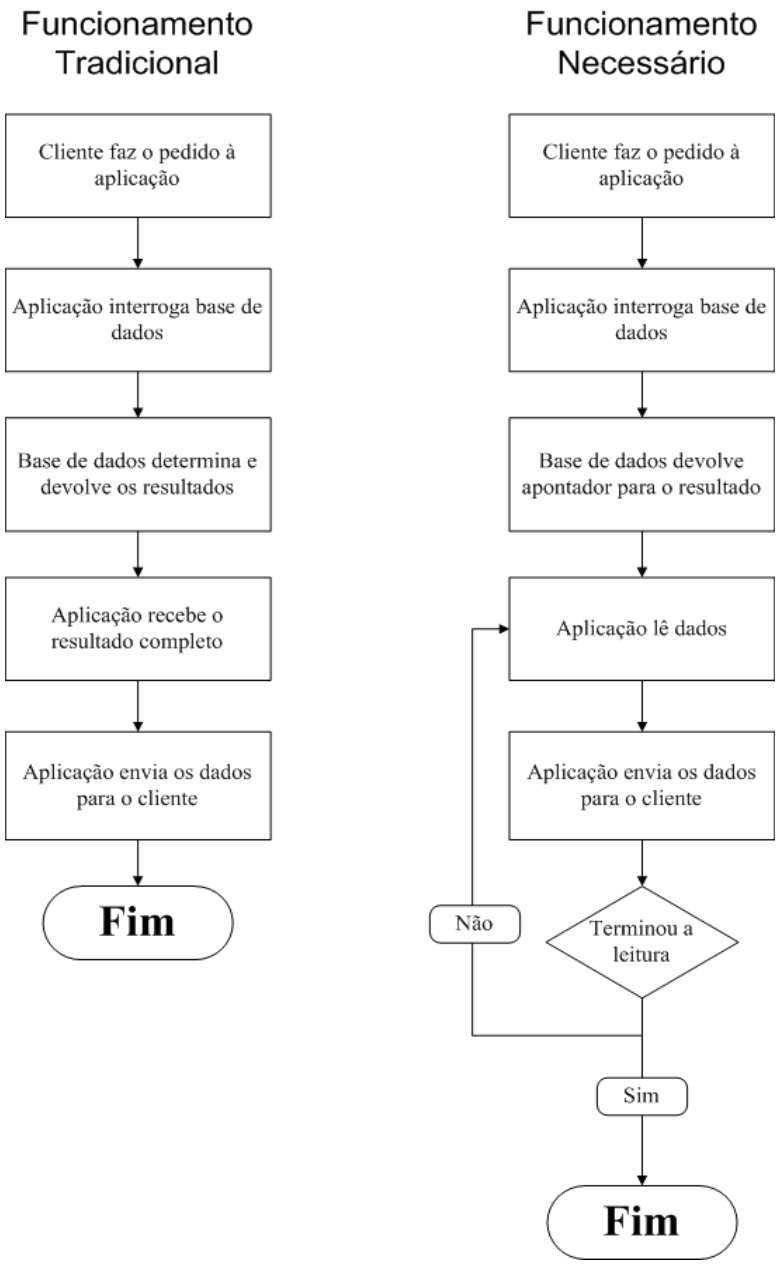

Figura 3-2 – Modos de funcionamento da transmissão de ficheiros

3.2.2 Módulo de Inserção de Conteúdos

O propósito deste módulo é permitir que conteúdos que não sejam provenientes de uma aplicação produtora de resultados possam ser inseridos na aplicação. Isto pode servir para que, por exemplo, um médico possa adicionar a um episódio de um doente uma foto que tirou numa cirurgia.

Para poder realizar uma acção deste tipo, o utilizador tem que inserir um conjunto de informações destinadas a identificar devidamente o documento que vai inserir no sistema.

Esta inserção de dados tem de ser feita em três passos, primeiro é necessário identificar o doente. Para isso é fornecido um mecanismo de pesquisa, onde o utilizador pode especificar uma ou várias das informações seguintes para pesquisar o doente:

- Nº Doente
- Nome do Doente
- Idade
- Data de Nascimento
- Sexo
- Nº SNS
- Telefone
- Morada
- Código Postal

Uma vez identificado o doente, é apresentada uma listagem dos episódios que lhe estão associados. Para cada episódio será apresentado:

- Identificador do Episódio
- Tipo Episódio
- Data Início
- Data Fim

Depois de seleccionado o episódio, o utilizador terá que inserir as informações relativas ao documento que vai inserir:

- Título
- Data
- Tipo
- Origem

Assim que toda esta informação esteja preenchida, o utilizador seleccionará o ficheiro que vai ser inserido, usando uma caixa de diálogo de abertura de ficheiros.

Este processo poderá ser repetido tantas vezes quantas o utilizador desejar. À medida que os ficheiros vão sendo seleccionados, será criada uma listagem com um aspecto semelhante à usada na apresentação de resultados e, se o conteúdo do ficheiro for suportado pelo módulo, será dada ao utilizador a possibilidade de visualizar o seu conteúdo no próprio módulo.

Quando o utilizador terminar o processo de selecção dos ficheiros a inserir na aplicação, poderá submeter todos os dados ao *eResults*, sendo-lhe apresentada, em cada momento, a percentagem do processo que já está concluída.

Capítulo 4

Especificação de Requisitos

4.1 Requisitos Não-funcionais

4.1.1 Requisitos de Interface Externos

4.1.1.1 Interface com Utilizadores

Do ponto de vista do utilizador, a interface será feita exclusivamente através de um *browser*, muito embora para o objectivo do *eResults* – a gestão documental – possam haver outros modos de interacção com o sistema, através da inserção de dados no sistema por outras aplicações, tais interfaces serão complementares e não estão no âmbito deste projecto pelo que não são aqui referidas.

Assim, do ponto de vista dos módulos efectivamente desenvolvidos neste projecto, temos uma interface para a inserção de dados e uma outra para a visualização de dados. É um requisito fundamental que ambas apresentem um aspecto sóbrio mas que sejam simultaneamente visualmente apelativas, muito funcionais e simples.

4.1.1.2 Interface de *Hardware*

Do ponto de vista do utilizador final, para a utilização do produto não será necessário mais que um computador.

4.1.1.3 Interface de *Software*

Para a utilização do módulo desenvolvido será necessário que o computador do utilizador possua a *Silverlight 2.0 Beta 2* instalada, sem a qual o funcionamento da aplicação é impossível.

Relativamente ao *browser*, este deverá ser o *Internet Explorer* 6 ou superior, muito embora a aplicação também funcione noutros *browsers*.

O módulo desenvolvido tem de interagir com a página onde está inserido através do *DOM*. Este processo é amplamente suportado pela *Silverlight*, embora não seja completamente transparente à mesma. O módulo tem de interagir com a página onde está inserido, para obter do utilizador os dados a pesquisar, uma vez que os campos de pesquisa estão alojados na página *Web* e não directamente no módulo.

Por último, para a obtenção, em tempo de execução, dos dados necessários ao funcionamento do módulo, este recorrerá a chamadas a *WebServices* da plataforma *.Net*. O formato em que os dados são transferidos para o *WebService* é *XML*. Para tal, aquando das chamadas aos procedimentos dos *WebServices*, os parâmetros da chamada ao procedimento, juntamente com todos os outros dados necessários à realização da chamada são convertidos para *XML* e remetidos em seguida ao endereço onde o serviço é alojado. Do ponto de vista do programador essa transformação de dados é transparente uma vez que a *Silverlight* suporta nativamente as chamadas aos *WebServices* usados.

4.1.1.4 Interface de Comunicação

Tipicamente todo o sistema estará disponível apenas na rede interna do hospital onde for instalado, pelo que é fundamental a existência de uma ligação do cliente a esta rede. Adicionalmente, e assumindo que o SGBD da aplicação será alojado num outro servidor, é também imprescindível uma ligação entre essa máquina e aquela que serve as páginas *Web* e os *WebServices*.

A capacidade de ligação entre os diversos departamentos onde os exames complementares de diagnóstico são realizados e o servidor aplicacional do *eResults*, também é necessária para que os resultados possam ser inseridos no sistema.

4.1.2 Requisitos de Ambiente

Dos requisitos de ambiente aquele que importa referir é o que diz respeito ao ambiente técnico. Dependendo das imagens a serem apresentadas, a aplicação pode ocupar um espaço considerável de memória RAM. Contudo, tendo em conta a utilização prevista, é expectável que um computador com 1GB de RAM permita o uso de todas as funcionalidades da aplicação, com um desempenho aceitável para o utilizador. Requer-se portanto que, para a usar a aplicação em pleno, os computadores possuam as características referidas.

Um outro ponto de relevo é o tempo de resposta do sistema. Para que a aplicação apresente um desempenho fluído, sem exigir longas esperas ao utilizador, quer para o carregamento, quer para a obtenção de resultados, o servidor do *eResults* e dos clientes devem fazer parte da mesma rede interna, para que os tempos de carregamento da aplicação sejam muito reduzidos, da ordem de 1 a 2 segundos.

4.1.3 Requisitos de Qualidade

O ambiente de operação em que o *eResults* funciona é crítico pois lida diariamente com dados clínicos onde qualquer falha pode ter consequências imprevisíveis para os pacientes. Como tal faz sentido analisar os principais requisitos de qualidade dos sistemas críticos:

- **Fiabilidade** – De entre todos os requisitos de qualidade é o que o sistema melhor deve cumprir. A apresentação de informação sobre um paciente que não corresponda aquela que lhe diz verdadeiramente respeito pode resultar num diagnóstico errado do médico ou, em última análise, causar fatalidades ou mazelas graves em pacientes. Por isso, este tipo de falhas é intolerável.
- **Disponibilidade** – A indisponibilidade do sistema causa perturbação na produtividade dos profissionais que dependem desta ferramenta para realizar as suas actividades. Se a indisponibilidade se estender a alguns dias, poderá levar a atrasos no diagnóstico de doentes que poderão ter consequências nefastas. Apesar disso, falhas de disponibilidade não serão tão graves como as falhas de fiabilidade.

- **Segurança** – Este requisito está directamente relacionado com o da fiabilidade. O sistema por si, como entidade física, não é causador de ameaças à vida humana, contudo se levarmos em consideração as consequências do seu funcionamento anormal no que respeita à fiabilidade dos dados, o mesmo requer um alto nível de segurança. É portanto necessário prevenir acessos não autorizados ao sistema, e garantir que os dados não são destruídos ou alterados maliciosamente.
- **Protecção** – Como qualquer sistema que lida com dados pessoais e mesmo confidenciais, a protecção da consulta desses dados por terceiros será sempre fundamental. Impõe-se assim que o acesso aos dados seja controlado, limitando o tipo de acesso dos diferentes tipos de utilizadores.

4.1.4 Requisitos de Usabilidade

Desde o início do desenvolvimento dos módulos, foi sempre muito claro que este deveria ser realizado tendo em vista a criação duma interface altamente usável, simples do ponto de vista do utilizador e apresentando a informação de forma clara. Esta necessidade, que deve ser sempre tida em conta no processo de desenvolvimento de *software,* torna-se então particularmente importante, pois é um meio de facilitar a utilização do módulo. Será necessário ter em atenção diversos aspectos que são analisados nas sub-secções seguintes.

4.1.4.1 Aprendizagem

Pretende-se que a aplicação seja de aprendizagem fácil, permitindo que os utilizadores facilmente realizem as tarefas básicas, logo na primeira vez que usam a aplicação. A satisfação deste critério influencia em grande parte a aceitação, ou não, da aplicação pelos utilizadores. As capacidades técnicas dos futuros utilizadores justificam a absoluta necessidade do cumprimento deste requisito.

4.1.4.2 Eficácia

A aplicação deve contribuir para o aumento da produtividade dos seus utilizadores. Para isso, o conjunto de acções requeridas para concluir um determinado caso de uso terá que ser tão reduzido quanto possível.

4.1.4.3 Objectivos de Experiência dos Utilizadores

Dos diversos objectivos de experiência dos utilizadores, a sensação de ajuda, a satisfação e um *design* apelativo são aqueles a que deve ser dada especial atenção.

4.1.5 Requisitos de Informação

No desenvolvimento da nova versão do *eResults* foi dedicado um cuidado especial à selecção do modo de armazenamento da informação que seria mais adequado, envolvendo questões como performance, extensão da informação a ser armazenada e o melhor método para a sua estruturação. Note-se, no entanto, que, a especificação deste requisito não faz parte do projecto, tendo sido utilizado o modelo que tinha sido previamente definido.

Uma vez que o modelo de dados é enorme, e apresenta diversos campos sem relevância para este projecto, apresenta-se na Figura 4-1, uma parte do mesmo com as tabelas que foram efectivamente de relevo para o projecto.

Para o desenvolvimento do projecto, espera-se que operações de baixa complexidade sejam efectuadas sobre a base de dados. As operações incidem essencialmente sobre as seguintes tabelas:

- ER_DOCUMENTO
- ER_ELEMENTO
- ER_FICHEIRO
- ER_DOCUMENTO_PREVIEW
- ER_TIPO_DOCUMENTO

Na tabela ER_DOCUMENTO são armazenados todos os documentos inseridos no sistema. Cada documento poderá ser composto por um ou mais sub-documentos, podendo cada um deles ter um conjunto de detalhes associados, da mesma forma que cada documento pode estar associado a um ou vários elementos (ER_ELEMENTO). No caso de serem vários, a entidade documento pode ser encarada como uma espécie de pasta.

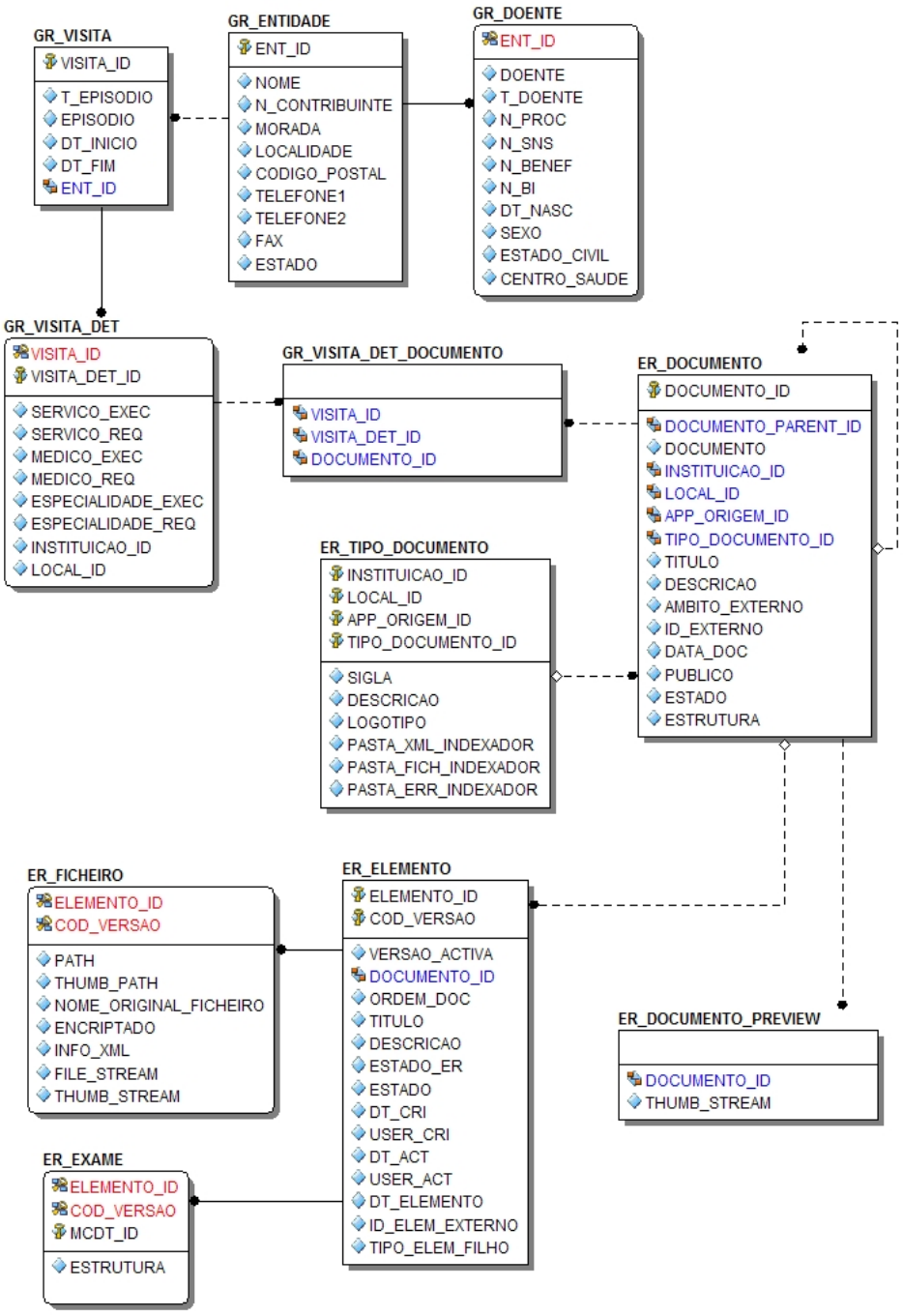

Figura 4-1 – Modelo de dados do eResults

No caso de o elemento ser relativo a um ficheiro o conteúdo deste será armazenado na tabela ER_FICHEIRO. No caso de ser relativa a um exame, o detalhe será inserido na tabela ER_EXAME.

Cada documento pode ter armazenada uma antevisão, isto é, uma imagem que deixa perceber qual é conteúdo do documento; nesse caso, a tabela ER_DOCUMENTO_PREVIEW conterá a entrada respectiva. Se, por exemplo, o

documento for relativo a uma imagem, nesta tabela pode ser armazenada uma miniatura da imagem.

No caso de o documento não possuir uma antevisão associada directamente, por omissão, a antevisão que será apresentada para este documento será uma imagem específica do seu tipo de documento. Esta estará armazenada na tabela ER_TIPO_DOCUMENTO.

4.2 Requisitos Funcionais

Após uma análise detalhada do problema, foram identificados os seguintes requisitos funcionais cuja listagem está separada por módulo:

A) Módulo de Apresentação dos Resultados

- O módulo deverá permitir efectuar pesquisas de resultados, de acordo com diversos parâmetros.
- Quando uma pesquisa é realizada, se esta não identificar inequivocamente o doente de que se pretende ver os resultados, deve ser apresentada uma lista de doentes na qual o utilizador poderá escolher o doente.
- Devem ser disponibilizados dois tipos de visualizações de resultados, um em modo de listagem e outro sob a forma de grelha de miniaturas, com uma imagem de pré-visualização do resultado[5].
- Os resultados do modo de visualização em listagem devem poder ser ordenados de acordo com qualquer um dos dados apresentados.
- O modo de visualização dos resultados em miniaturas deve disponibilizar um mecanismo de redimensionamento de miniaturas.
- O modo de visualização dos resultados em miniaturas deve permitir filtrar resultados apresentados com uma pesquisa local.
- Os resultados que corresponderem a imagens e vídeos devem poder ser abertos e manipulados no próprio módulo; os resultados analíticos devem ser abertos numa outra página do *eResults*, especializada na visualização desse tipo de

[5] Para simplificar a referência a este modo de visualização, de agora em diante passarei a referir-me a ele como o modo de visualização de resultados em miniaturas

resultados; os restantes resultados devem ser abertos numa nova janela do *browser* delegando-lhe a escolha do tratamento que deve dar ao ficheiro.

B) Módulo de Inserção de Conteúdos

- O módulo deverá permitir a inserção de documentos no *eResults* através da sua associação a um episódio.
- Deve ser possível especificar a informação a associar ao documento a inserir.
- Deve ser possível preparar a inserção de vários documentos e consumar a sua inserção de forma simultânea.
- Se o documento a inserir for uma imagem ou vídeo, e este já estiver listado, deve ser possível efectuar a sua apresentação dentro do módulo.
- Quando os documentos estão a ser submetidos ao *eResults* deve ser apresentada uma percentagem do progresso da operação.

Capítulo 5

Descrição da Solução

Neste capítulo é efectuada a descrição dos módulos desenvolvidos sem atentar aos pormenores de desenvolvimento, que serão descritos no Capítulo 6.

5.1 Módulo de Apresentação dos Resultados

5.1.1 Aspecto Inicial

O aspecto da página do *eResults* que contém o módulo de apresentação dos resultados é apresentado na Figura 5.1; na Figura 5.2 é apresentado o detalhe da pesquisa avançada.

Um utilizador poderá preencher qualquer um dos campos de pesquisa apresentados para efectuar pesquisas. A significância de cada um dos campos de pesquisa disponibilizados já foi abordada no capítulo 3.1.1.

A funcionalidade de pesquisa menos intuitiva é a que permite definir que a pesquisa deverá levar em conta apenas as datas que estiverem incluídas num determinado intervalo. Este intervalo pode ser definido para levar em conta as datas dos documentos, dos episódios ou da validação. De resto, os dados serão inseridos usando caixas de texto, caixas de selecção e caixas de selecção de datas.

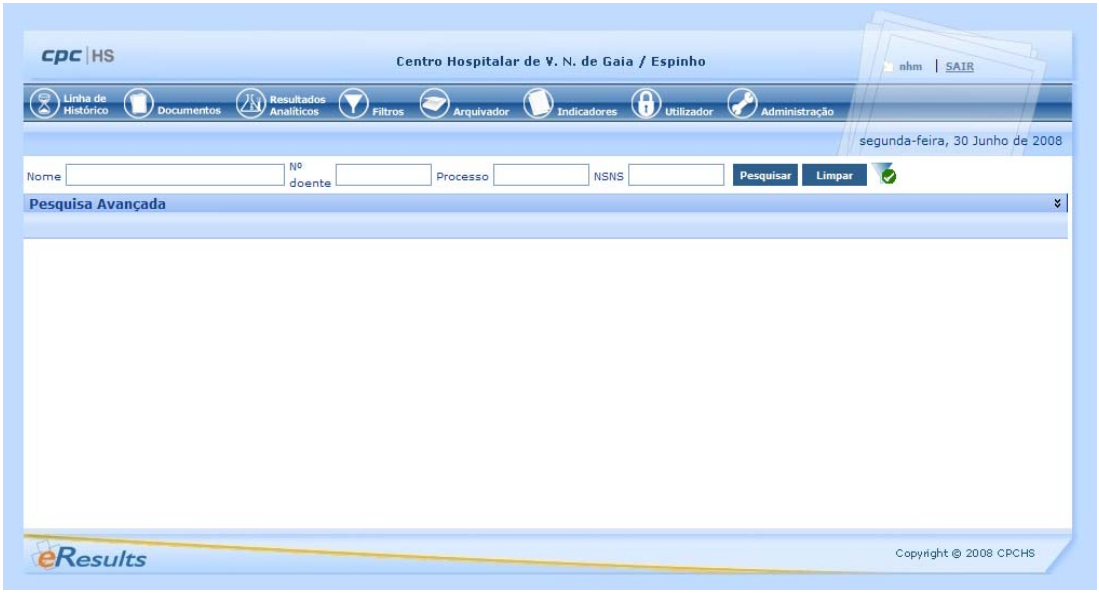

Figura 5-1 – Aspecto inicial do módulo de apresentação de resultados

Figura 5-2 – Aspecto das opções de pesquisa avançada

As acções associadas aos botões “Pesquisar” e “Limpar” são óbvias: o botão “Pesquisar”, quando clicado, dá início à acção de pesquisa, e o “Limpar” limpa não só os resultados apresentados como todos os valores contidos nos campos de pesquisa.

5.1.2 Selecção do Doente

Uma vez que de uma pesquisa podem resultar múltiplos doentes, é necessário que em alguns casos o utilizador indique qual o doente, de entre todos os que verificam os critérios da pesquisa, cujos resultados pretende visualizar. A pesquisa terá o aspecto apresentado na Figura 5.3.

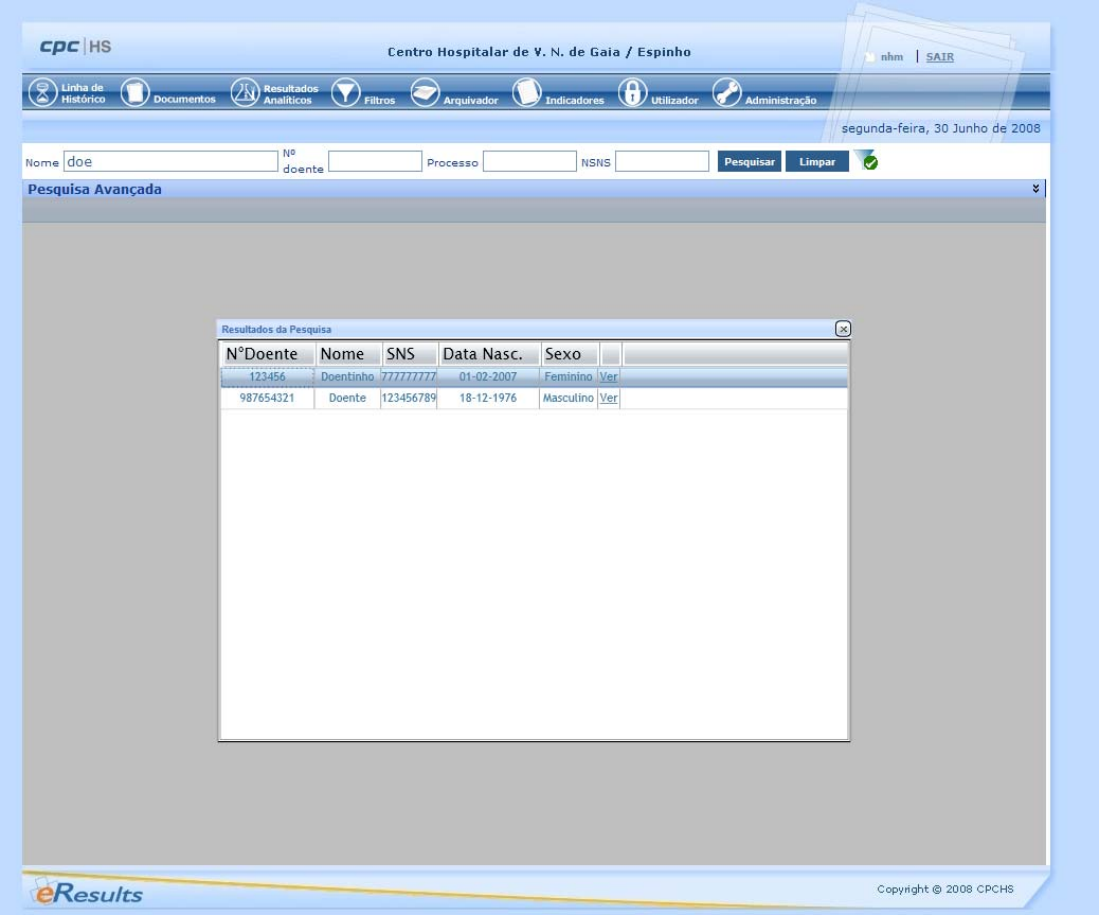

Figura 5-3 – Selecção do doente

Para seleccionar o doente o utilizador terá apenas que clicar na linha em que o doente é apresentado.

5.1.3 Visualização de Resultados

Quando um utilizador efectua uma pesquisa, os resultados serão apresentados numa janela com o aspecto apresentado na Figura 5.4 em que o modo de visualização de resultados é o de miniaturas. A Figura 5.5 ilustra a visualização em modo de listagem.

Para alternar entre os dois modos o utilizador tem apenas que clicar num dos botões apresentados na interface com o texto “Lista” e “Thumb”. A alternância da apresentação dos dados é imediata.

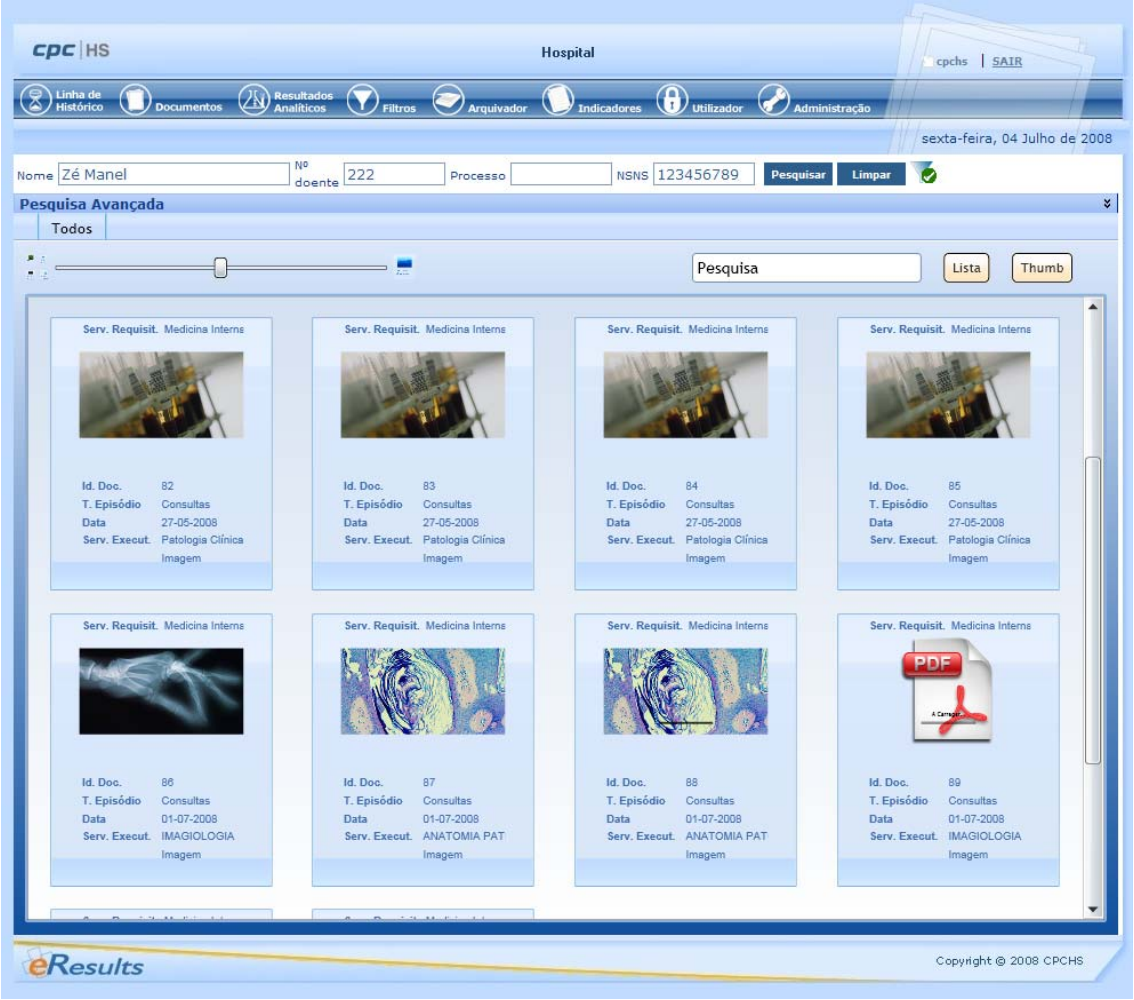

Figura 5-4 – Aspecto do modo de visualização com miniaturas

No modo de visualização de listagem, a lista pode ser ordenada de acordo com qualquer um dos campos de informação. Ainda relativamente a este modo de visualização, para poder abrir o resultado, o utilizador tem apenas que clicar no ícone que aparece na última coluna de cada linha da listagem.

No outro modo de visualização, o das miniaturas, que se pode ver na Figura 5.4 , cada resultado retornado pela pesquisa será apresentado numa caixa que agrupa a informação relevante e a imagem de pré-visualização do resultado. Como uma pesquisa pode retornar muitos resultados, as imagens de pré-visualização são carregadas sequencialmente.

Esta caixa pode ser seleccionada com um clique, passando a ter um realce à sua volta, e, quando o utilizador passa o cursor por cima da mesma, o seu tamanho cresce ligeiramente, voltando ao normal quando o cursor abandona a sua área.

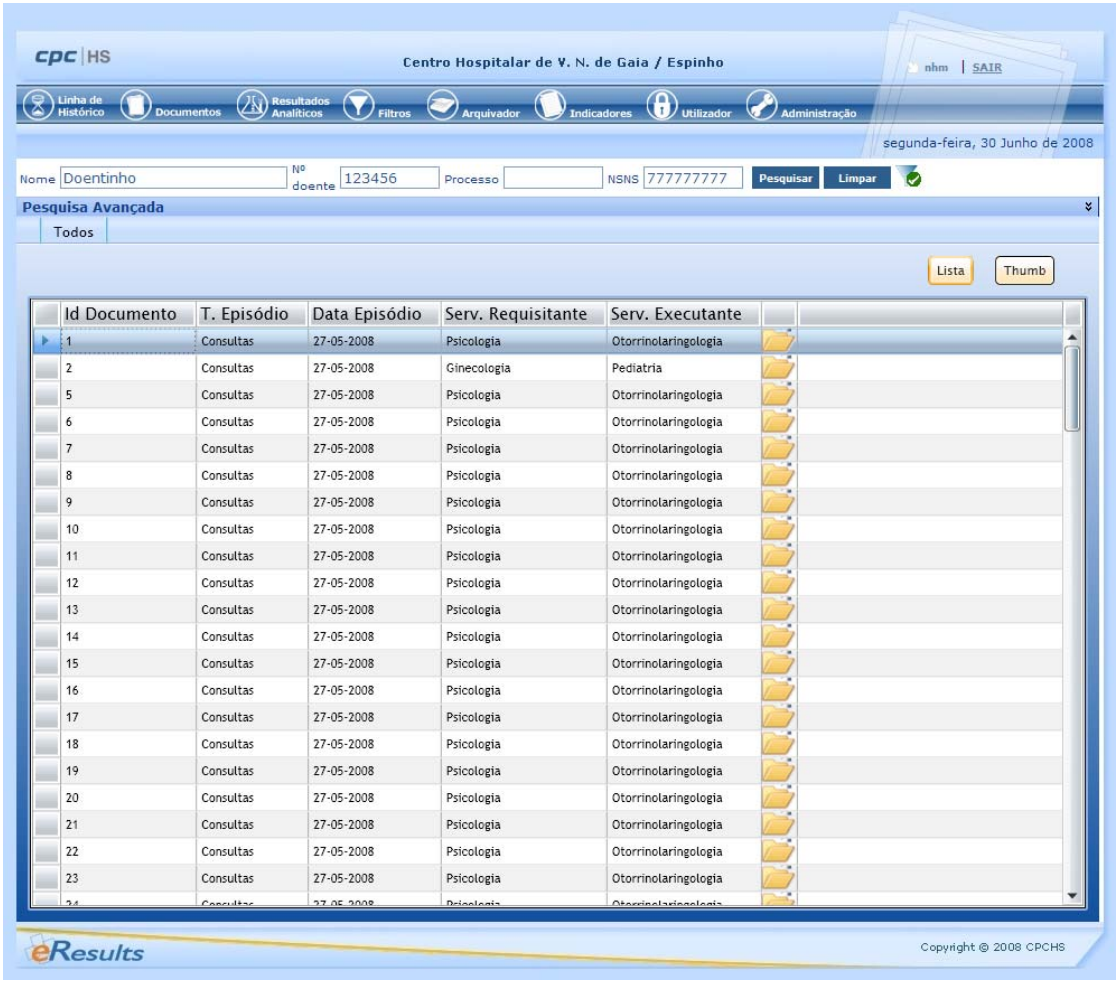

Figura 5-5 – Aspecto do modo de listagem

É ainda possível aumentar e diminuir os tamanhos das miniaturas usando o *slider* apresentado no canto superior esquerdo, como se pode verificar pela Figura 5.6.

Usando a caixa de texto que contém o texto “Pesquisa”, o utilizador pode filtrar localmente os resultados apresentados. Se o utilizador digitar texto nesta caixa, passam a ser apresentados somente os resultados que contêm o texto em pelo menos um dos seus campos de informação. Este comportamento é ilustrado pela Figura 5.7. Para voltar a apresentar os resultados que eram apresentados antes da realização da filtragem basta apagar o texto digitado na caixa.

Neste modo de visualização, para abrir o resultado, o utilizador necessita somente de executar um duplo clique sobre a imagem de pré-visualização do resultado. O procedimento de abertura do resultado é apresentado nos pontos seguintes.

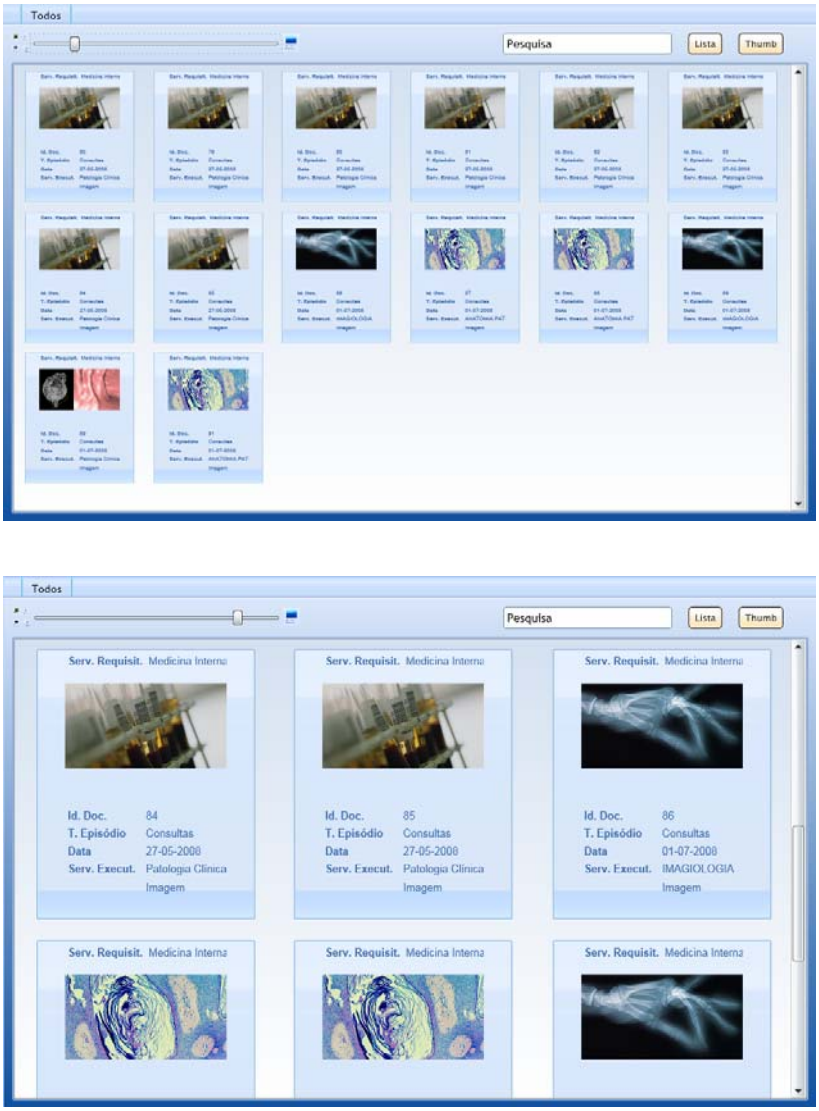

Figura 5-6 – Aumento e diminuição do tamanho das caixas de resultados

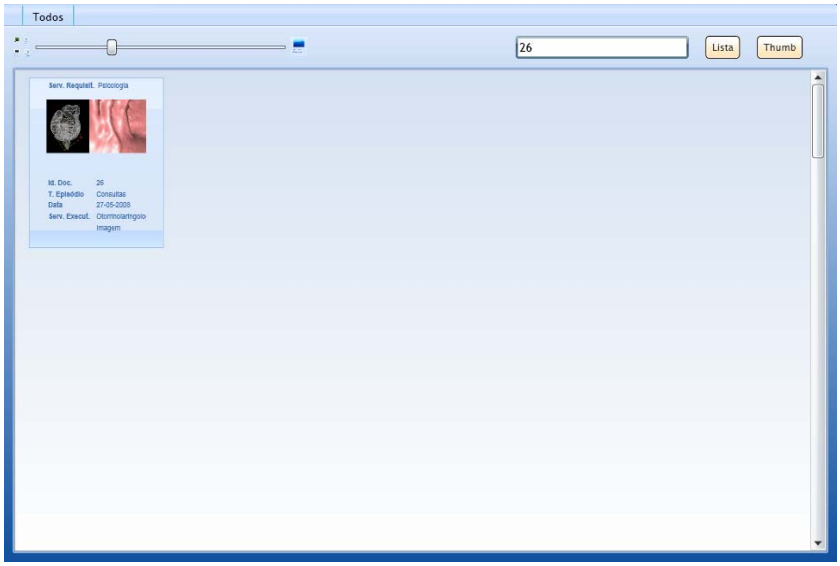

Figura 5-7 – Filtragem local de resultados

5.1.3.1 Imagem

Se o resultado a apresentar for uma imagem esta será aberta dentro do próprio módulo, como se apresenta na Figura 5.8. Ao realizar a abertura da imagem é usada uma animação que introduz a imagem no ecrã e que desfoca a listagem de resultados redimensionando-a para um tamanho menor e dando-lhe um efeito translúcido.

Para permitir manipular a imagem, quando o utilizador passa o cursor por cima da mesma é apresentado em cada canto da imagem, um quadrado semi-transparente que se o utilizador clicar e arrastar com o cursor, pode usar para redimensionar e rodar a imagem. Se, por outro lado, o utilizador clicar em qualquer outra parte da imagem poderá movê-la.

Figura 5-8 – Visualização de uma imagem

Uma imagem que seja manipulada pode então apresentar o aspecto demonstrado na Figura 5.9.

Figura 5-9 – Visualização de uma imagem manipulada no módulo

Ao efectuar um duplo clique sobre a imagem o utilizador regressa ao modo de visualização de resultados, sendo essa transição mediada por um efeito contrário ao do aparecimento da imagem no ecrã.

Resta referir que o carregamento das imagens, vídeos e miniaturas é realizado de forma sequencial. No entanto, como o desejo do utilizador visualizar um resultado se sobrepõe à necessidade do carregamento das imagens de pré-visualização, a partir do momento em que a ordem de abertura de uma imagem é dada, esta será iniciada assim que o carregamento que está a decorrer terminar.

5.1.3.2 Vídeo

Se o resultado a apresentar for um vídeo este será aberto dentro do próprio módulo, como se apresenta na Figura 5-10.

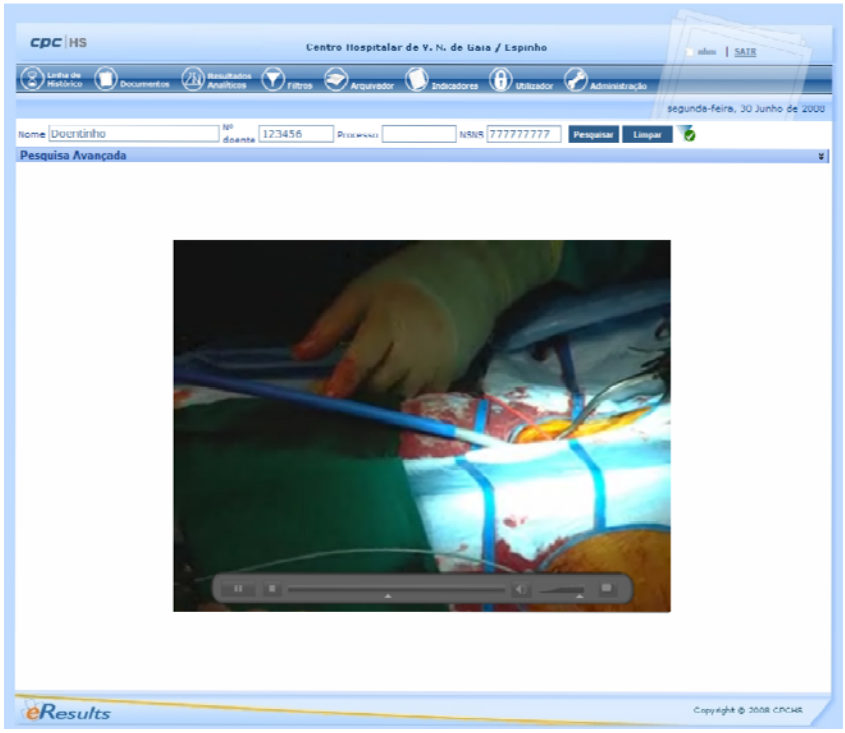

Figura 5-10 – Visualização de um vídeo

O reprodutor de vídeo terá um aspecto semelhante a muitos outros. Apresenta botões para parar, pausar e iniciar a reprodução, um botão para activar ou desactivar o som, um botão para activar a reprodução em ecrã inteiro, uma barra que apresenta o progresso de carregamento do vídeo e o progresso de reprodução do vídeo, e ainda uma outra barra para o controlo do volume.

Para fechar a visualização do vídeo bastará que, de forma semelhante à imagem, se faça duplo clique sobre a janela do vídeo. A abertura e fecho do vídeo usa o mesmo tipo de animações que as imagens.

5.1.3.3 Resultado Analítico

A abertura de resultados analíticos é realizada numa página do *eResults* específica para esse efeito. Essa abertura deve ser feita navegando-se da página do módulo que aqui se descreve para a página dos resultados analíticos. Nesse sentido, quando o utilizador decide abrir um resultado deste tipo, a visualização dos resultados é enegrecida, tal como se apresenta na Figura 5-11, e é apresentada uma espécie de círculo no centro da imagem a rodar continuamente, para informar o utilizador que o processo de abertura está a decorrer.

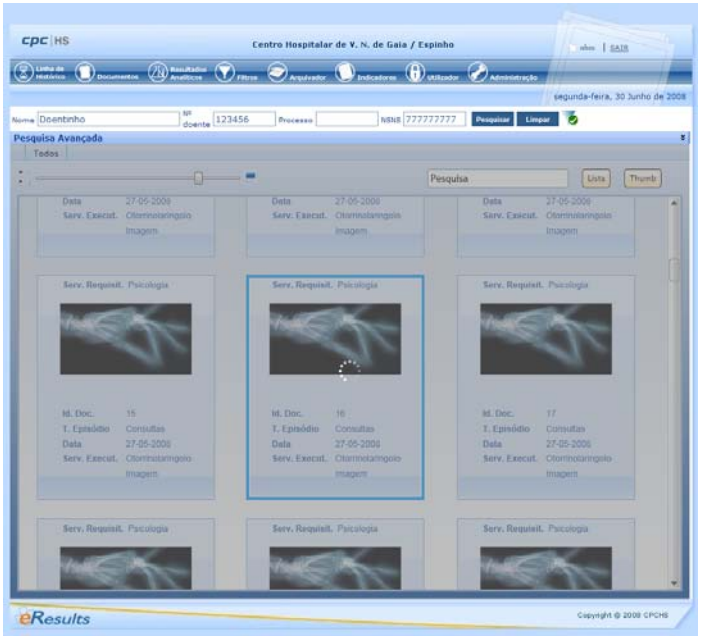

Figura 5-11 – Abertura de um resultado analítico

5.1.3.4 Outros Resultados

A abertura de qualquer outro tipo de resultado é realizada numa nova janela do *browser*, aberta para esse efeito; o *browser* ficará então encarregado do tratamento do ficheiro. Na figura 5-12 é apresentada a abertura de um documento do tipo *pdf* que foi delegada ao *browser*, e que este abriu numa nova janela.

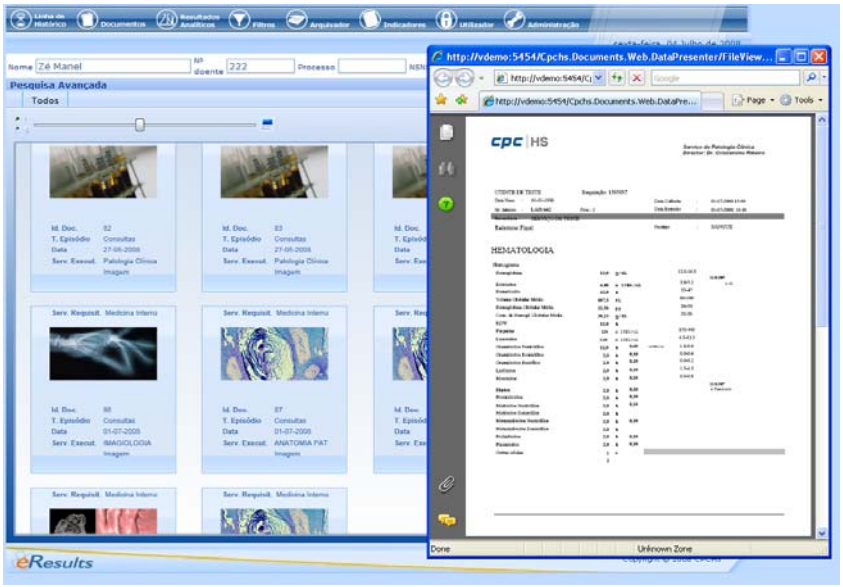

Figura 5-12 – Abertura de um resultado delegada ao 'browser'

5.2 Módulo de Inserção dos Conteúdos

Este módulo, do ponto das funcionalidades de interacção com o utilizador, nada acrescenta, para além do formulário que o utilizador deve preencher para se permitir a submissão dos dados ao servidor.

Este módulo, tal como o módulo apresentado no capítulo anterior, permite a visualização de imagens e vídeos dentro do próprio módulo.

O aspecto do módulo integrado no *eResults* é apresentado nas figuras 5-13 e 5-14.

Os documentos cujas informações já foram preenchidas, e que são apresentados nas miniaturas por baixo do formulário, têm associado um botão com uma cruz de cor vermelha que permite remover o item do conjunto de documentos a inserir no *eResults* aquando do clique no botão com o texto “Submeter”.

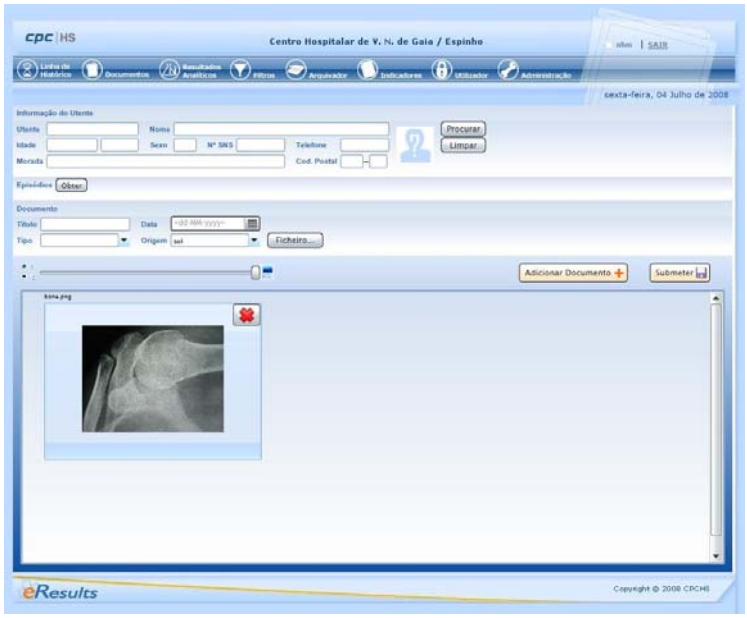

Figura 5-13 – Aspecto global do módulo de inserção de conteúdos

Figura 5-14 – Detalhe do formulário de inserção de conteúdos

Capítulo 6

Detalhes do Desenvolvimento

6.1 Arquitectura Lógica

A arquitectura lógica da aplicação é apresenta na Figura 6-1. Pela sua análise notam-se de imediato as três camadas que constituem a aplicação, a camada do cliente, a camada da aplicação, e finalmente a camada da base de dados.

Na camada do cliente, o *browser* realiza pedidos ao *WebSite* do *eResults*, que são respondidos com o envio de páginas *HTML*. Por sua vez, o *plugin* do *Silverlight* comunica com a camada da aplicação, através dos *WebServices*. Finalmente todos os pedidos de acesso à base de dados são intermediados pelos *WebServices*, sejam os do *WebSite*, ou os do *plugin*.

A camada do cliente constituiu o principal foco ao longo do desenvolvimento deste projecto. Nesta inclui-se o módulo `Cpchs.Modules.DocumentsManagement.UI` que diz respeito ao Módulo de Visualização de Resultados, e o módulo `Cpchs.Modules.Indexer.UI` que corresponde ao Módulo de Inserção de Conteúdos.

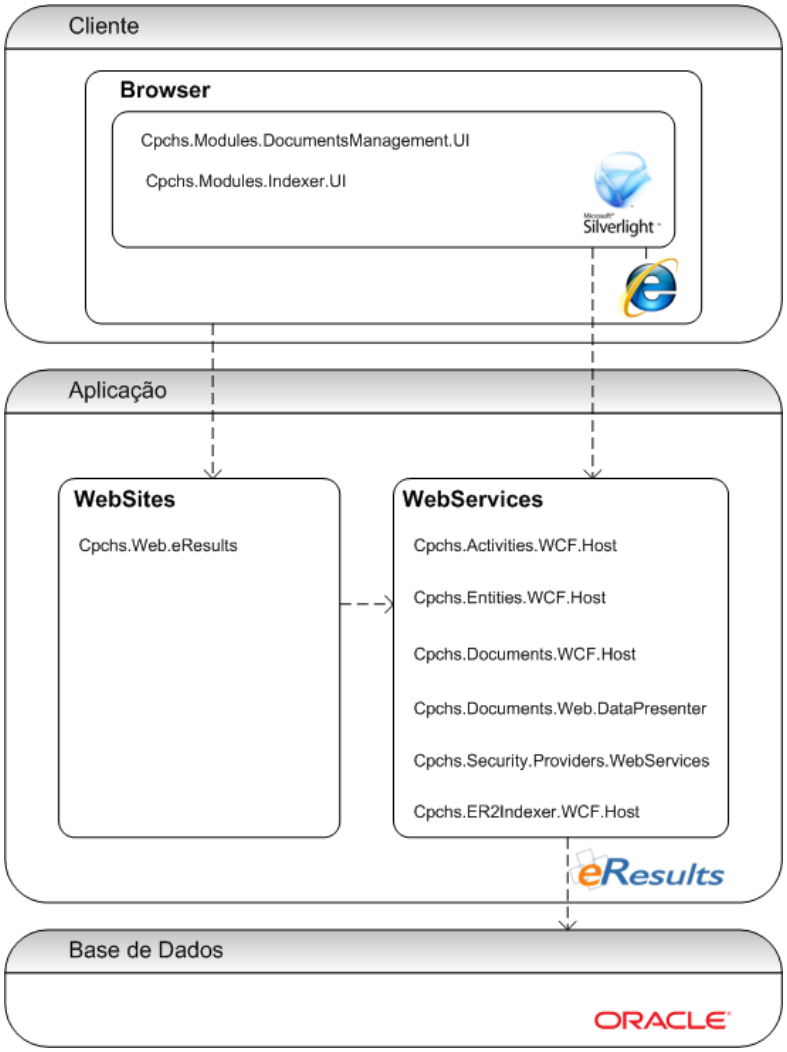

Figura 6-1 – Diagrama da arquitectura lógica do eResults

A camada da aplicação pode ser agrupada em dois grupos lógicos, o do *WebSite* e o dos *WebServices*. No *WebSite* existe apenas o módulo `Cpchs.Web.eResults` que é responsável pela apresentação das páginas solicitadas pelo cliente através do *browser*. Nos *WebServices* existem quatro módulos que correm efectivamente como *WebServices*:

- `Cpchs.Activities.WCF.Host` – concentra os serviços relativos a actividades de um paciente na unidade hospitalar, como por exemplo a realização de uma visita ou de um exame
- `Cpchs.Documents.WCF.Host` – define os serviços de gestão de documentos e elementos

- `Cpchs.Entities.WCF.Host` – agrega todos os serviços relacionados com entidades do modelo de domínio, como por exemplo, um doente ou uma instituição
- `Cpchs.Security.Providers.WebServices` – fornece serviços relacionados com a segurança da aplicação, como por exemplo a autenticação do utilizador.
- `Cpchs.ER2Indexer.WCF.Host` – fornece um único serviço para a inserção de conteúdos no *eResults*.

O outro módulo incluído neste grupo lógico, mas que não funciona efectivamente como um *WebService,* é o `Cpchs.Documents.Web.DataPresenter`, que na realidade é um *WebSite*. A sua existência neste formato deve-se às necessidades da *Silverlight* que permitem mais facilmente gerir a realização de um *download* se este for realizado a partir de um página *Web*, do que se este for realizado através do *download* da informação por um *WebService*. Apesar disso, o seu funcionamento assemelha-se ao de um *WebService,* na medida em que recebe um pedido de dados e somente essa informação é retornada na página de resposta criada. Por esse motivo foi deliberado que esse módulo seria agrupado logicamente com os restantes *WebServices* usados na aplicação. Resta referir que este módulo é composto por duas páginas, uma delas é responsável pela mediação da transmissão de resultados que correspondam a ficheiros, e a outra fornece as imagens de pré-visualização dos resultados.

6.1.1 *WebServices* usados

Para além do módulo `Cpchs.Documents.Web.DataPresenter`, somente os módulos `Cpchs.Documents.WCF.Host`, `Cpchs.Entities.WCF.Host`. e `Cpchs.ER2Indexer.WCF.Host` foram usados no projecto. É de referir que o seu desenvolvimento não ficou a cargo do autor deste projecto, mas devido ao uso extenso que lhes é dado pela aplicação, uma descrição dos mesmos é importante. Para tal, será feita a apresentação do modelo do contrato de serviço[6] e do modelo do contrato de dados[7], focando as partes que foram de algum modo relevantes para este projecto.

[6] O modelo do contrato de serviço refere-se ao modelo usado para definir a interface do serviço, especificando as operações que o serviço suporta e as mensagens a trocar entre o serviço e o cliente.
[7] O modelo do contrato de dados refere-se ao modelo onde se define os tipos de dados não primitivos a usar no serviço.

Do ponto de vista do módulo de apresentação de resultados, estes *WebServices* são essencialmente uma interface para a base de dados. Os serviços relativos aos documentos e às entidades, do ponto de vista do uso que lhe é dado neste projecto, têm um comportamento muito semelhante, podendo ser sumariado nos seguintes passos:

1. Recepção dos dados a pesquisar na chamada do método do serviço
2. Realização do pedido de resultados à base de dados com os dados recebidos na chamada
3. Realização de algumas operações sobre a base de dados com os dados recebidos
4. Devolução do resultado à entidade que realizou o pedido

O módulo do indexador tem um comportamento ligeiramente diferente dos anteriores, dado que o seu propósito é o da realização de inserções de dados na base de dados e, como tal, não retorna resultados.

6.1.1.1 Módulo *Entities*

Este módulo agrega todos os serviços relacionados com entidades do modelo. No caso aqui retratado a entidade de interesse é o paciente.

A) Contrato de Serviço

Muito embora o conjunto de operações associadas a este método seja maior, aqui apresenta-se o único método que foi usado pelo módulo de apresentação dos resultados. O contrato de serviço é, então, o apresentado na Figura 6-2.

O método `SimplePatientFind` é usado pelo módulo de apresentação dos resultados. Este devolve uma lista de doentes que possuem documentos associados, restringindo essa mesma lista, aos critérios de pesquisa recebidos na chamada do método.

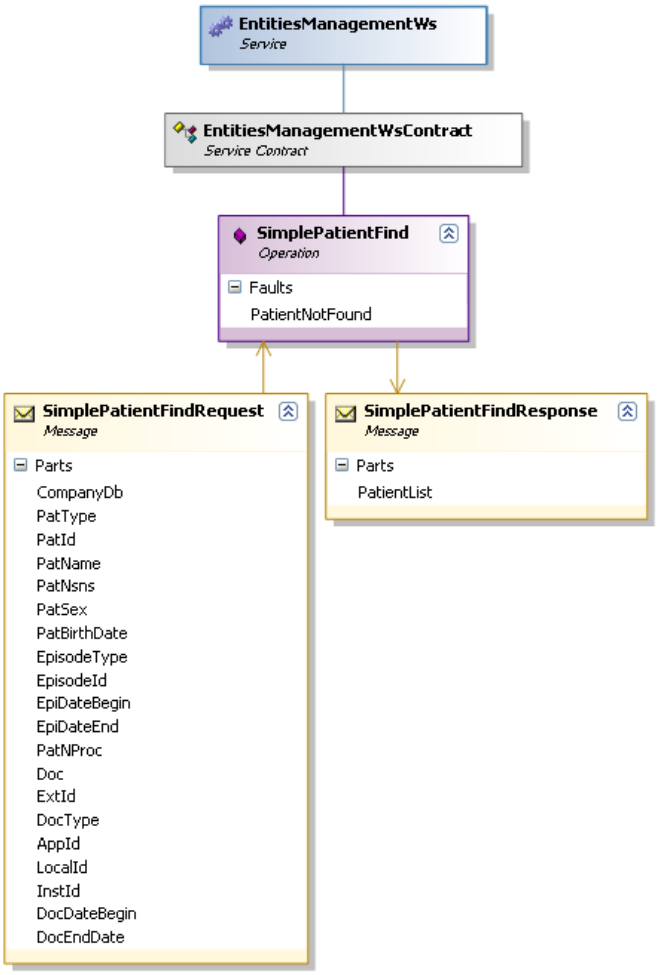

Figura 6-2 - Diagrama do contrato de serviço do serviço 'entities'

Em relação aos parâmetros recebidos pelo método, pouco há a dizer para além de que estes são praticamente uma listagem dos campos de pesquisa disponíveis para o utilizador no módulo de apresentação de resultados, excepção feita ao `CompanyDB`, que tem que ver com o modo de funcionamento do *eResults* como um todo, mas que não é relevante para este relatório.

B) Contrato de dados

O contrato de dados deste serviço também é consideravelmente mais extenso que a parte que se apresenta na Figura 6-3.

Relativamente a este contrato pouco há a dizer; para cada paciente existe um conjunto de dados que podem ser preenchidos. A lista de pacientes a devolver pelo método será do tipo `Patients` que é uma colecção de `Patient`'s.

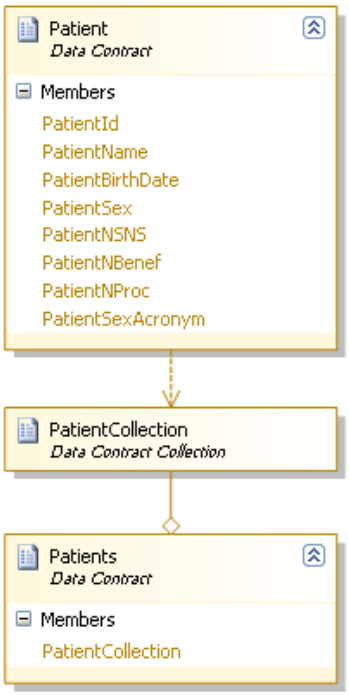

Figura 6-3 – Diagrama do contrato de dados do serviço 'entities'

6.1.1.2 Módulo *Documents*

Tal como já foi referido, este módulo é responsável pela gestão de documentos.

A) Contrato de Serviço

O diagrama do contrato de serviço deste serviço é apresentado na figura 6-4.

O método `GetPatientDocuments` do serviço *documents* retorna os documentos associados a um paciente, e à semelhança do método `SimplePatientFind` do serviço *entities*, restringe essa lista, aos critérios de pesquisa recebidos na chamada do método.

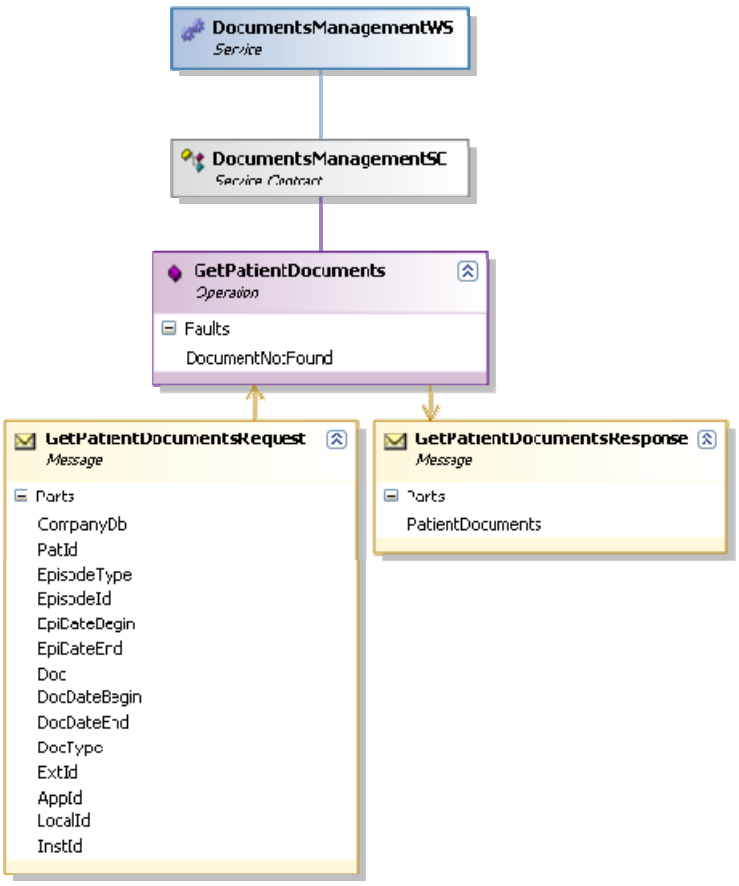

Figura 6-4 - Diagrama do contrato de serviço do serviço 'documents'

B) Contrato de dados

O diagrama do contrato de dados dos documentos é apresentado na Figura 6-5. No diagrama verifica-se que cada documento tem diversas informações associadas, necessárias, no cliente, para a realização do seu tratamento adequado.

Contudo alguns atributos carecem duma explicação, nomeadamente o `DocFileExt`, que virá preenchido apenas se o documento possuir efectivamente um ficheiro associado – podia ser um resultado analítico por exemplo. O segundo atributo a merecer realce é o `DocInfo`, que é um dicionário utilizado para devolver diversas informações de um documento a serem apresentadas, como, por exemplo, o episódio médico a que se refere.

Por último, o `DocElemType` é um atributo que indica simplesmente se o documento é um ficheiro ou um resultado analítico.

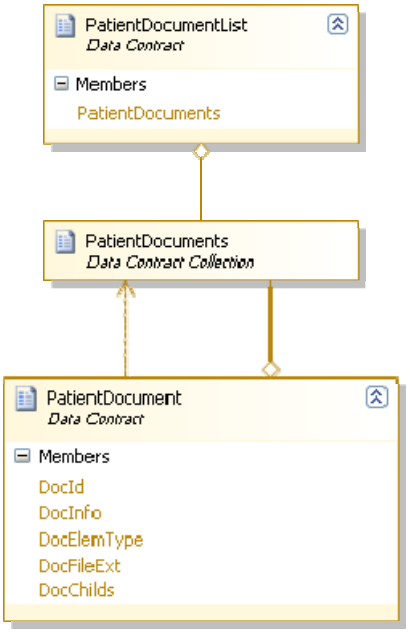

Figura 6-5 – Diagrama do contrato de dados do serviço 'documents'

6.1.1.3 Módulo ER2Indexer

A) Contrato de Serviço

O diagrama do contrato de serviço deste serviço é apresentado na Figura 6-6.

Muito embora o contrato de operações deste serviço seja muito simples, as operações que o mesmo realiza são bastante complexas. O único método disponibilizado recebe na sua chamada um objecto do tipo `EResult2`, que irá conter tanto o ficheiro a inserir na base de dados como as meta-informações que devem ser associadas a este.

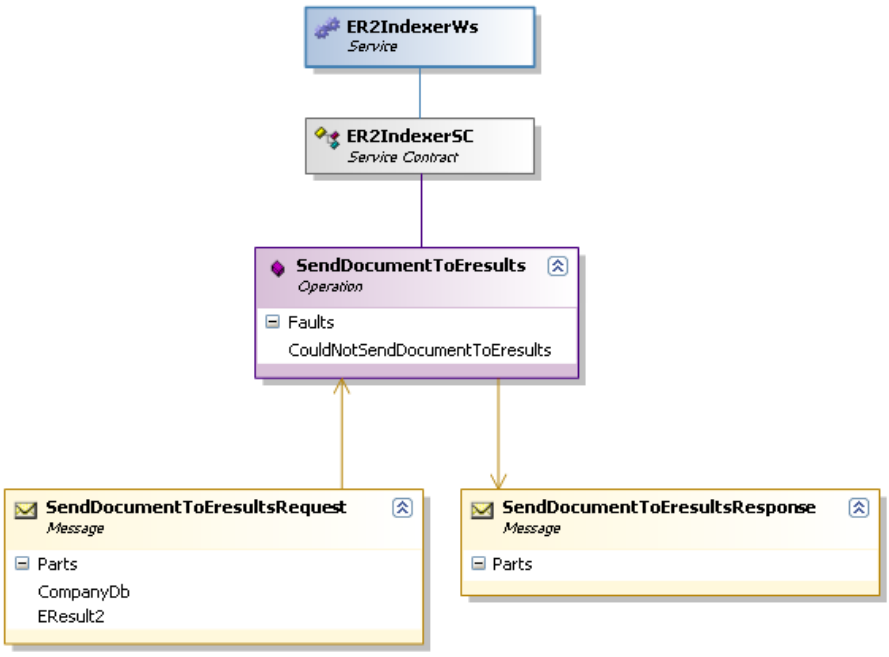

Figura 6-6 – Diagrama do contrato de serviço do serviço 'ER2Indexer'

B) Contrato de Dados

O diagrama do contracto de dados deste serviço é apresentado na figura 6-7.

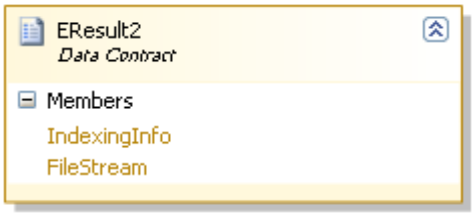

Figura 6-7 – Diagrama do contrato de dados do Serviço 'ER2Indexer'

Conforme ressalta da descrição do contrato do serviço, este tipo de dados é composto pela informação de indexação e pelo ficheiro a ser enviado para a base de dados. O atributo `IndexingInfo` contém a informação de indexação num *XML*, construído de acordo com um formato acordado na empresa e cujo conteúdo é descrito no anexo A. O atributo `FileStream` contém o ficheiro. Ambos os atributos são do tipo `Object`.

6.2 Estrutura dos Módulos

6.2.1 Módulo de Apresentação dos Resultados

O diagrama da figura 6-8 esquematiza as classes criadas para a implementação do Módulo de Apresentação dos Resultados.

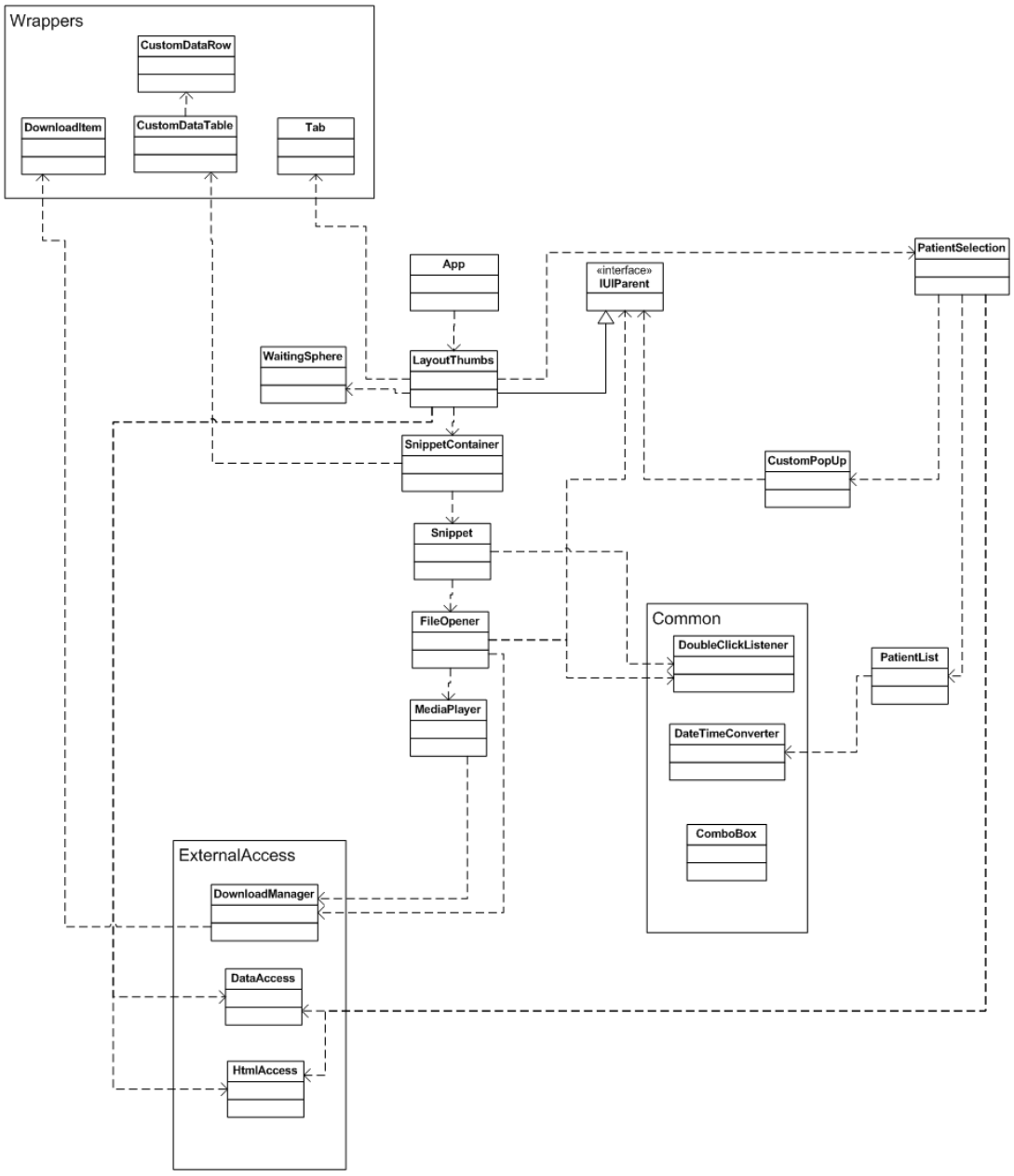

Figura 6-8 – Diagrama de componentes do Módulo de Apresentação dos Resultados

Da análise do diagrama, notam-se de imediato a existência de três sub-módulos internos, cujo propósito se explica de seguida:

- ***Common*** – Inclui classes necessárias para o desenvolvimento do projecto mas que podiam facilmente ser usadas num qualquer outro projecto desenvolvido na *Silverlight.*
- ***External Access*** – Inclui as classes que realizam acessos da aplicação ao domínio externo da aplicação; isso inclui acesso a recursos da página onde a aplicação é executada, abertura de outros recursos *Web* ou chamadas aos *WebServices.*
- ***Wrappers*** – Inclui classes de encapsulamento de dados, para facilitar o tratamento dos mesmos por outras classes da aplicação.

Verifica-se também a existência de um conjunto central de classes: *LayoutThumbs*, *SnippetContainer*, *Snippet*, *FileOpener* e, por último, *MediaPlayer*. Explica-se de seguida, o papel de cada uma no módulo desenvolvido.

- ***LayoutThumbs*** **–** Visa criar uma classe de nível superior aos *SnippetContainer's*, permitindo que uma instância da aplicação suporte diversos *container's* usando *Tabs.* Simultaneamente abstrai os *container's* das operações de acesso aos *WebServices.*
- ***SnippetContainer*** **–** Trata-se da classe que cria efectivamente o *layout* da interface com o utilizador. Tal como o próprio nome indica, é o "contentor" das *Snippet's*, tendo a responsabilidade de as gerir.
- ***Snippet*** **–** Corresponde à classe que permite a geração de uma miniatura de um resultado, e contém uma classe do tipo *FileOpener* para apresentar a imagem da miniatura.
- ***FileOpener*** **–** Classe responsável pela abertura de qualquer tipo de ficheiro na aplicação, seja internamente ao módulo, ou recorrendo a uma nova janela do *browser*.

6.2.2 Módulo de Inserção de Conteúdos

O diagrama da figura 6-9 esquematiza as classes criadas para a implementação do Módulo de Inserção de Conteúdos.

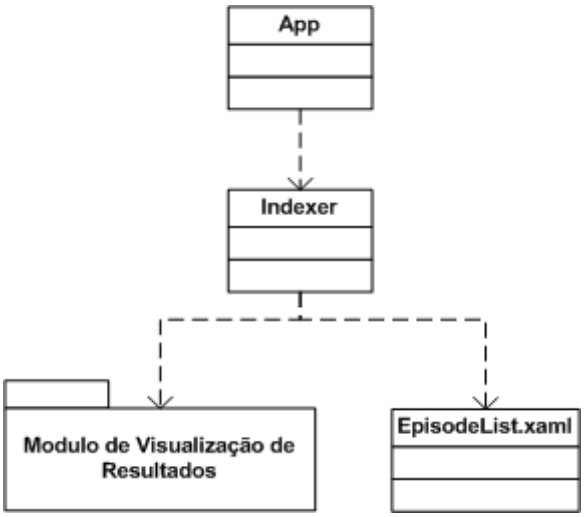

Figura 6-9 - Diagrama de componentes do Módulo de Inserção de Conteúdos

Como se pode ver pelo diagrama, o módulo de inserção de conteúdos tem estrutura muito mais simples, uma vez que reutiliza em grande parte o código do Módulo de Visualização de Resultados. Como tal, neste módulo existe uma classe, onde é desenhado o formulário para a inserção de conteúdos no documento a inserir na aplicação, o *EpisodeList*, ao passo que a classe *Indexer* agrupa o *SnippetContainer* à *EpisodeList*, criando assim o *layout* deste módulo, cabendo-lhe ainda as funções de gestão das chamadas aos *WebServices*.

6.3 Detalhes da Implementação

Em seguida apresentam-se alguns detalhes da implementação que foram considerados de alguma forma relevantes. Para criar uma estruturação dos tipos de componentes a que dizem respeito, a explicação dos diferentes detalhes de implementação foi agrupada de acordo com os seguintes critérios: temos primeiro os detalhes da implementação de componentes que fazem sentido como uma entidade única e que poderiam em teoria ser facilmente reutilizados num outro projecto da *Silverlight.* Depois temos os detalhes de implementação que têm mais que ver com a tecnologia *Silverlight* em si, do que com os módulos desenvolvidos, a que se seguem os detalhes cuja explicação só faz sentido no âmbito dos módulos deste projecto. Por último, temos a explicação de como se realiza a transferência de dados da base de dados para o cliente, sem obter completamente o resultado da base de dados, no servidor aplicacional.

6.3.1 Componentes

6.3.1.1 *Tab*

A classe *Tab* permite a criação de *Tabs* na aplicação, a classe apresenta o modo de funcionamento visto em tantas outras aplicações que usam este tipo de componentes visuais.

Relativamente à classe *Tab* importa ainda referir que, apesar de uma análise do esquema fazer crer que esta deveria ser incluída no grupo *Common*, o seu nível de desenvolvimento não permite o funcionamento da classe como uma entidade completamente independente, daí a decisão de a colocar no grupo *Wrapper*.

6.3.1.2 *ComboBox*

Uma *ComboBox* é um componente visual usado na construção de interfaces com o utilizador, composta por uma caixa de texto e uma lista de dados. A lista de dados é apresentada quando o utilizador clica sobre a caixa de texto. Quando o utilizador selecciona um item da lista, o conteúdo desse item passa a ser apresentado na caixa de texto.

Componentes deste tipo são facilmente encontrados na maioria das tecnologias que permitem o desenvolvimento de interfaces com o utilizador. Contudo, porque a *Silverlight* se encontra ainda em estado *beta*, este componente não é disponibilizado de momento. Apesar disso a necessidade deste controlo determinou que o mesmo tivesse que ser desenvolvido pela empresa para poder integrar os módulos a desenvolver.

O aspecto do controlo desenvolvido é apresentado na figura 6-10.

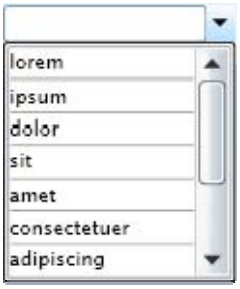

Figura 6-10 – Aspecto da 'ComboBox' desenvolvida

Este controlo foi desenvolvido recorrendo aos controlos `TextBox`, `PopUp` e `ListBox` que já fazem parte dos controlos da interface suportados pela *Silverlight.*

6.3.1.3 *DoubleClickListener*

Esta classe faz a detecção de eventos de duplo clique num objecto visual. Para usá-la basta realizar a sua instanciação, passando-lhe como parâmetros o objecto e a função que deve ser chamada quando o duplo clique ocorrer sobre o objecto.

Para funcionar, esta classe regista-se como receptora dos eventos de clique sobre o objecto passado na instanciação e usa um temporizador de 500 milissegundos. O temporizador é activado quando um clique ocorre no objecto, e, se um segundo clique ocorrer antes de o alarme ser disparado, é chamada a função que se registou como receptora do evento de duplo clique, aquando da instanciação desta classe.

6.3.2 Detalhes Gerais à Tecnologia Silverlight

6.3.2.1 *Download* de Ficheiros

Uma vez que uma pesquisa de resultados pode retornar um grande número de resultados, foi necessário criar um mecanismo para o controlo dos *downloads* que permitisse a sua realização de forma sequencial, ao invés de iniciar o *download* de todas as imagens de forma simultânea quando os resultados são apresentados. Se este desenvolvimento não tivesse sido feito a aplicação apresentaria uma grande degradação na performance.

A classe desenvolvida para este fim foi a *DownloadManager*, que é responsável pela gestão do carregamento de todas as imagens e vídeos na aplicação. O principal desafio da implementação desta classe foi realizar a gestão dos *downloads* localmente, permitindo ao mesmo tempo que os objectos que requisitam os *downloads* recebam informação sobre o progresso do descarregamento e, obviamente, o objecto descarregado, quando o *download* termina.

A classe *DownloadManager* é referenciada de forma estática existindo apenas uma instância da mesma para gerir todos os *downloads*. Para o uso desta classe basta chamar um dos métodos cujo protótipo é apresentado na figura 6-11.

```
public void Queue(Priority p,
                  Uri dlUri,
                  DownloadProgressReport dlProgressChangeReport,
                  DownloadEndedReport dlCompletedReport);
public void Queue(Priority p,
                  Uri dlUri,
                  MediaDownloadProgressReport dlProgressReport,
                  MediaElement player);
```

Figura 6-11 – Protótipos das funções de 'queuing' de 'downloads'

Dos métodos apresentadas na Figura 6-11 resta ainda referir que são usados para inserir na fila de espera respectivamente, um *download* de uma imagem e o *download* de um vídeo.

Internamente, esta classe cria um objecto do tipo `DownloadItem` para cada objecto que se pretende descarregar, o qual é depois inserido numa `List<DownloadItem>` que será gerida pela classe, num comportamento que se assemelha a uma fila de prioridades.

A gestão dos *downloads* tem que considerar a prioridade e tipo de *download* que se pretende fazer. A prioridade é definida aquando da inserção do *download* na classe através da enumeração `Priority`. O mapeamento das prioridades é simples, e é realizado tal como se descreve na tabela 6-1.

Objecto a descarregar	Prioridade
Imagem de pré-visualização do resultado	Baixa
Resultado na forma de uma imagem	Alta
Resultado na forma de um vídeo	Alta

Tabela 6-1 – Mapeamento das prioridades de 'download'

Quando se insere um *download* na fila de espera, se a sua prioridade for elevada este será inserido à cabeça da lista, sendo o próximo *download* a ser realizado; pelo contrário, se a prioridade for baixa, o *download* é inserido no final da fila de espera. Este critério permite que os pedidos de visualização de um resultado do tipo imagem ou vídeo, que são abertos dentro do módulo, sejam sempre atendidos o mais rapidamente possível.

O *download* das imagens em *Silverlight* dispõe de mecanismos que permitem realizar a gestão do progresso do *download* de uma imagem ou de um vídeo de forma simples.

O código para a criação do *download* de uma imagem é apresentado na Figura 6-12.

```
//image download
webClient = new WebClient();
webClient.DownloadProgressChanged +=
    new DownloadProgressChangedEventHandler();
webClient.OpenReadCompleted +=
    new OpenReadCompletedEventHandler(ImageDownloadCompleted);
webClient.OpenReadAsync(DlUri);
```

Figura 6-12 – Criação do 'download' de imagem na Silverlight

Este código regista um método `ImageDlProgressReport` que será chamado quando o progresso do *download* for alterado de forma significativa, e um outro `ImageDownloadCompleted`, que será chamado quando o *download* terminar, permitindo obter o objecto descarregado e dar-lhe o tratamento adequado.

Por sua vez, o *download* de um vídeo é exemplificado na Figura 6-13.

```
//video download
MediaElement player = new MediaElement();
player.Source = DlUri;
player.DownloadProgressChanged +=
    new RoutedEventHandler(PlayerDownloadProgressChanged);
```

Figura 6-13 – Criação do 'download' de vídeo na Silverlight

Aqui, como a realização do *download* é delegada directamente ao reprodutor de vídeo da *Silverlight*, o número de instruções é mais reduzido, bastando indicar ao reprodutor de vídeo qual o *URI* do vídeo a descarregar para iniciar o *download*. O reprodutor, num processo executado internamente, faz o *streaming* do vídeo, e inicia a reprodução do mesmo quando possuir em memória uma porção do vídeo que considere suficiente. De forma semelhante ao que acontece no *download* de imagens, um método `PlayerDownloadProgressChanged` será chamado quando o progresso do *download* for alterado de forma significativa.

6.3.2.2 Animações

As animações foram usadas para que:

- Quando um cursor entra e sai de uma miniatura, se aumente e diminua o tamanho dessa miniatura
- Quando é aberto um resultado do tipo imagem ou vídeo, a listagem de resultados seja encolhida e tornada translúcida, ao mesmo tempo que à imagem ou vídeo a ser aberto é aplicado um efeito de entrada no ecrã
- Se produzam efeitos diversos na barra de pesquisa e botões do reprodutor de vídeos

Dos métodos de produção de animações que a *Silverlight* disponibiliza, o modo usado foi o mais complexo, permitindo por exemplo, que se desse uma sensação de que os objectos animados, estavam a sofrer o efeito da aceleração e desaceleração. A criação destas animações, sem o auxílio da *Silverlight* teria sido uma tarefa morosa e muito propícia a erros.

No código da figura 6-14 demonstra-se como as animações são aplicadas na prática, este código é uma porção simplificada do código que realiza as animações que as miniaturas são alvo quando o cursor passa sobre as mesmas.

```
<Grid.RenderTransform>
    <TransformGroup>
        <ScaleTransform     x:Name="scaleTransformControl"
                            ScaleX="1" ScaleY="1"/>
    </TransformGroup>
</Grid.RenderTransform>

<Grid.Resources>
    <Storyboard x:Name="mouseIn">
        <DoubleAnimationUsingKeyFrames
                Storyboard.TargetName="scaleTransformControl"
                Storyboard.TargetProperty="ScaleX">
            <SplineDoubleKeyFrame
                Value="1.1"
                KeySpline="0.0,0.7 0.35,1.0" KeyTime="0:0:1"/>
        </DoubleAnimationUsingKeyFrames>
        <DoubleAnimationUsingKeyFrames
            Storyboard.TargetName="scaleTransformControl"
            Storyboard.TargetProperty="ScaleY">
            <SplineDoubleKeyFrame
                Value="1.1"
                KeySpline="0.0,0.7 0.35,1.0" KeyTime="0:0:1"/>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>
</Grid.Resources>
```

Figura 6-14 – Aplicação de uma animação a um objecto

As animações são definidas usando `Storyboard's`. Uma `Storyboard` pode ser constituída por diversos tipos de animações mas, neste caso, é apresentada apenas a `DoubleAnimationUsingKeyFrames`, que permite a animação de objectos usando diferentes `KeyFrame`'s. A vantagem deste método é que permite que uma animação possa ser realizada em vários passos, adoptando em cada passo um comportamento diferente. Os `KeyFrame`'s podem também ser de diferentes tipos; no caso apresentado na Figura 6-14 os `SplineDoubleKeyFrame`, que têm a particularidade de permitir criar animações usando uma s*pline*.

A animação definida no código apresentado na figura 6-14 afecta a escala do objecto. Cada uma das `DoubleAnimationUsingKeyFrames` usa a propriedade `x:Name`, definida na `ScaleTransform`, para referenciar esse mesmo escalamento como o objecto da animação. As `DoubleAnimationUsingKeyFrames` usam depois, a propriedade `SplineDoubleKeyFrame` para indicar a *spline* a ser usada na animação e o valor para o qual o escalamento deve ser animado.

A curva produzida usando os pontos de controlo (0, 0.7) e (0.35, 1), e consequentemente, a animação produzida, é ilustrada pela Figura 6-15.

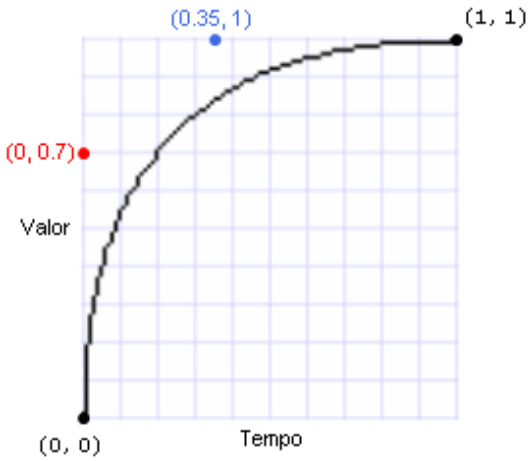

Figura 6-15 – Spline usada na animação das miniaturas

6.3.2.3 *Garbage Collection*

A *Silverlight* apresenta-se como uma das diversas tecnologias que realizam a *garbage collection*[8] como um processo gerido internamente, libertando o programador dessa tarefa. De acordo com a documentação existente, para que um objecto se torne elegível para a sua recolha e eliminação basta que todas as referências existentes para o mesmo sejam eliminadas.

Ao testar os módulos aqui apresentados, verificou-se, de forma inesperada, que a memória não era gerida de forma correcta, pois muito embora as referências aos objectos estivessem a ser eliminadas, somente o recarregamento da página no *browser* libertava a memória usada pelo *plugin*.

Após alguma investigação sobre o assunto verificou-se que este era um problema já conhecido, causado pela má gestão da subscrição de eventos por parte da *Silverlight*. Como tal, para contornar o problema criou-se o método `Dispose` em todas as classes desenvolvidas, onde se anula a referência a outros objectos e a subscrição de eventos.

6.3.2.4 Reprodutor de Vídeos

O reprodutor de vídeos desenvolvido para o projecto, é conceptualmente um objecto visual simples, possuindo as funcionalidades descritas em detalhe no capítulo 5.1.3.2. Uma vez que a descodificação do vídeo é totalmente tratada internamente pela *Silverlight*, o processo que aqui se descreve é o da construção da interface do reprodutor.

O reprodutor de vídeos foi desenvolvido num único *user control* que foi denominado `MediaPlayer`. Este apresenta as características clássicas de um controlo desenvolvido em *XAML* em que todo o seu *layout* é definido no *XAML* e o tratamento de eventos é realizado no *code-behind.*

[8] A *garbage collection*, ou recolha de lixo em português, é um termo usado no domínio do *software*, para se referir ao processo recolha e eliminação de recursos de memória que já não estão a ser usados por programas.

Ao controlo foram aplicadas diversas animações em todos os botões e ainda à própria barra de pesquisa, para criar o efeito de que os botões estão a ser de facto premidos quando clicados, ou realçar o botão quando o cursor entra na sua área.

6.3.3 Específicas dos Módulos Desenvolvidos

6.3.3.1 Interacção com a página *HTML*

O desenvolvimento do módulo de apresentação de resultados precedeu o desenvolvimento do controlo da `ComboBox`. Esse facto determinou que para colmatar a inexistência de um controlo desse tipo em *Silverlight* se usassem controlos *HTML* para obter os valores a serem pesquisados. Esses valores são depois obtidos pelo módulo usando a capacidade da *Silverlight* poder aceder ao *DOM* da página onde está inserida.

Muito embora pela descrição, este processo possa parecer simples, dada a quantidade de campos de pesquisa existentes, ele é muito propício à ocorrência de erros. Este facto, aliado a uma certa indecisão na determinação dos campos que devem ser disponibilizados para a realização das pesquisas – maioritariamente causada pelo cliente – determinou que muitas horas de desenvolvimento fossem dedicadas para disponibilizar o acesso aos dados na página.

Contudo, agora que uma *Combo Box* já foi desenvolvida pelo autor deste projecto, espera-se que dentro em breve se integrem os campos de pesquisa no próprio módulo, eliminando os acessos à página através do *DOM* e consequentemente as dificuldades que lhe são inerentes.

6.3.3.2 Pesquisa de pacientes reutilizável

Uma necessidade que não foi inicialmente pensada no projecto dos módulos, foi que para manter o método da realização das pesquisas consistente em todo o *eResults*, o aspecto que os formulários de pesquisa apresentavam, bem como a forma com que os resultados duma pesquisa que retornava diversos doentes eram apresentados, teriam de ser semelhantes no módulo de apresentação de resultados, e num outro módulo fora do âmbito deste projecto, de visualização de resultados analíticos.

Esta necessidade implicou uma reformulação de certas partes do código a meio do desenvolvimento do projecto, para que a pesquisa de pacientes pudesse ser realizada de forma independente do módulo de apresentação de resultados.

Deste esforço resultou que os métodos para a pesquisa de pacientes foram concentrados numa única classe, que permite a criação duma pesquisa de doentes e obtenção de resultados de forma assíncrona.

```
PatientSelection patientSel = new PatientSelection(rootVisual);
patientSel.PatientSelected  += new
PatientSelection.PatientSearchEnded(PatientSelection_Selected);
patientSel.Search_Patients();
```

Figura 6-16 – Código para a criação da primeira fase da pesquisa de resultados

```
/*********************************************/
/* evento para sinalizar a escolha do paciente */
/*********************************************/
public delegate void PatientSearchEnded(object sender, string
patientId);
public event PatientSearchEnded PatientSelected;
```

Figura 6-17 – Especificação do evento difundido quando o paciente é seleccionado

Como se pode ver na Figura 6-16, à classe `PatientSelection` é passada a `rootVisual` na sua criação. A `rootVisual` é a raiz da aplicação que implementa a interface `IUIParent`. Esta referência é necessária para que se possa criar *popup's* com o resultado de uma pesquisa, para que o utilizador indique qual o doente do qual deseja ver os resultados.

Depois é registado o método a ser chamado aquando da obtenção do resultado da pesquisa.

Numa última nota sobre este assunto, refere-se que uma vez que a barra de pesquisa – e consequentemente os campos de pesquisa – são os mesmos nos dois módulos que usam esta pesquisa, não é necessário para já, passar ao método `Search_Patients` os parâmetros a pesquisar; contudo, a realização dessa alteração ao código, se necessária no futuro, é trivial.

6.3.3.3 Redimensionamento das Miniaturas

Permitir que as miniaturas dos resultados sejam redimensionáveis, mantendo sempre uma apresentação ajustada ao espaço disponível na aplicação, foi um processo moroso e de alguma complexidade.

O espaço onde as miniaturas estão inseridas é um *Canvas*[9], inserido dentro de um *ScroolViewer*[10].

A decisão da necessidade de redimensionamento cabe ao utilizador. Quando este deseja redimensionar as miniaturas, pode fazê-lo usando o *slider* que existe na aplicação para o efeito. Como tal, a classe responsável pela apresentação das miniaturas regista-se como receptora dos eventos de modificação do valor do *slider* e, quando este valor é alterado, procedem-se aos redimensionamentos das miniaturas e ajustes do *layout* necessário.

O processo de redimensionamento e ajuste das miniaturas compreende os seguintes passos, que foram mapeados para métodos na classe *SnippetContainer*:

1) Chamada do método `scale_ValueChanged` como resultado duma alteração da posição do *slider* e consequentemente do seu valor; neste método verifica-se se o espaço disponível no *Layout* é suficiente para as novas dimensões que as miniaturas vão ter após redimensionamento, ou se pelo contrário este é excessivo, permitindo a inserção de uma nova coluna para a apresentação de miniaturas. As miniaturas são então redimensionadas.
2) O método `reArrange` é chamado para reposicionar as miniaturas
3) O método `updateScroll` é chamado para que a barra de deslocamento vertical do *ScroolViewer* seja actualizada, para permitir a visualização correcta de todas as miniaturas existentes no *Canvas*.

[9] Um *Canvas* em *Silverlight*, é um componente que funciona como uma tela, na qual o programador pode "desenhar" o que desejar, e redimensionar da mesma forma. Neste caso em particular, possibilita a inserção e posicionamento das miniaturas, usando translações no eixo horizontal e vertical.
[10] Um *ScroolViewer* é um componente que contém uma área que pode ser navegável quer na horizontal, quer na vertical, mediante as necessidades do programador. Os conteúdos são navegáveis usando barras de deslocamento. Na área navegável podem ser inseridos os mais diversos objectos visuais, tais como uma imagem, ou um bloco de texto.[16]

6.3.3.4 Ajuste automático ao espaço disponível

Para aumentar as potencialidades dos módulos desenvolvidos, foi determinado que estes deveriam ajustar-se automaticamente ao espaço que lhes era disponibilizado pelo *plugin* do *Silverlight* no *Browser*.

Muito embora este processo seja suportado automaticamente por diversas tecnologias – incluindo a *WPF* por exemplo – para conseguir este objectivo na *Silverlight* é necessário combinar cuidadosamente o uso de *Grid*'s nas diversas classes que compõem os módulos desenvolvidos, não definindo as dimensões dos *user controls* e permitindo à *Grid* o seu ajuste completo ao espaço disponível.

A *Grid* da *Silverlight* permite a definição do *layout* de uma interface do utilizador, usando linhas e colunas [6]. Sem entrar em grandes pormenores sobre a sua utilização, a funcionalidade disponibilizada pela *Grid* que importa aqui referir é o dimensionamento das linhas e das colunas que a compõem, de modo a que estas se ajustem ao espaço disponível. Para tal, ao identificar as dimensões das colunas e das linhas da *Grid*, basta indicar que a sua dimensão é *star* (estrela) o que em *XAML* se faz usando um asterisco (*). O código na Figura 6-18 figura apresenta a definição da *Grid* da classe *SnippetContainer*, para ilustrar este processo.

```
<UserControl
    x:Class="Cpchs.Modules.DocumentsManagement.UI.SnippetContainer"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="10"    />
            <ColumnDefinition Width="*"     />
            <ColumnDefinition Width="Auto"  />
            <ColumnDefinition Width="10"    />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition MinHeight="10" Height="Auto"/>
        </Grid.RowDefinitions>
    </Grid>

     (...)

</UserControl>
```

Figura 6-18 – Definição de linhas e colunas numa Grid

Para não suscitar dúvidas, refira-se que a dimensão *Auto* define que o tamanho da linha/coluna onde é usada, assumirá altura/largura do componente com a maior dimensão que seja definido nessa linha/coluna.

Como se verifica, a segunda coluna e a segunda linha da *Grid* foram definidas usando a dimensão *star* para que estas ocupassem todo o espaço que resta depois da definição das outras linhas e das outras colunas no *plugin*. Replicando este processo pelos restantes componentes que definem o *layout* dos módulos, consegue-se que a aplicação se ajuste sempre perfeitamente a todo o espaço disponível. Na prática houve necessidade de aplicar este mecanismo à raiz da aplicação, a *LayoutThumbs*, e ao seu filho, o *SnippetContainer*.

Esta definição permite o ajuste do *layout* ao espaço disponibilizado; contudo, para maximizar o uso do espaço disponível, mais que redimensionar o espaço, há que redimensionar/ajustar os conteúdos ao espaço disponível quando a aplicação é redimensionada em tempo de execução. Isto implicou a necessidade da resolução de outros problemas tal como o ajuste do posicionamento e tamanho das miniaturas ao espaço disponível quando a página é redimensionada.

Dando uso aos métodos criados para o redimensionamento das miniaturas, cujo processo é descrito em pormenor na secção 6.3.3.3, a resolução do primeiro problema torna-se trivial, bastando recorrer ao evento que a *Silverlight* disponibiliza de detecção de alterações do tamanho página. Ao tratar este evento, o valor do *slider* é alterado internamente, fazendo disparar o evento de alteração desse valor, que, conforme já foi descrito, ao ser tratado culmina na actualização das dimensões e posições das miniaturas para se ajustarem ao espaço disponível.

6.3.4 Transmissão de ficheiros do Servidor

Os problemas que a transmissão de dados cria no servidor já foram descritos na secção 3.2.1.1 pelo que aqui se descreve apenas a implementação da resolução.

Antes de mais, refira-se mais uma vez que o pedido para o *download* de ficheiros é feito através de chamadas a páginas do módulo `Cpchs.Documents.Web.DataPresenter`. O código apresentado na figura 6-19 é

retirado da página `FileViewer`, que obtém os dados de um ficheiro armazenado na base de dados em pequenas partes e envia-os para o cliente. A tabela a que acede é a *ER_FICHEIRO* e o parâmetro que usa para determinar qual o ficheiro que deve devolver é o identificador do documento. O ficheiro está armazenado num campo do tipo `blob` que é convertido para o tipo `OracleLob` do C#.

```
public void GetFileStream(string docId)
{
    OracleConnection con = ConnectionData.Connect();

    OracleDataAdapter datadap = new OracleDataAdapter();

    datadap.SelectCommand = new OracleCommand(
      "SELECT nome_original_ficheiro, file_stream " +
      "FROM er_ficheiro, er_elemento " +
      "WHERE er_ficheiro.elemento_id = er_elemento.elemento_id " +
      "AND " +
      "documento_id = :docId ", con);

    datadap.SelectCommand.Parameters.Add("docId", OracleType.Number);
    datadap.SelectCommand.Parameters["docId"].Value = docId;

    OracleDataReader reader = datadap.SelectCommand.ExecuteReader();

    using (reader)
    {
        reader.Read();
        //Obtain the LOB
        OracleLob blob = reader.GetOracleLob(1);

        //set the headers
        setResponseHeaders(reader.GetOracleString(0).ToString(),
                           blob.Length.ToString());

        //send the data
        byte[] bytes = new byte[1024 * 128];
        int bytesRead;
        while ((bytesRead = blob.Read(bytes, 0, bytes.Length)) > 0)
        {
            Response.OutputStream.Write(bytes, 0, bytesRead);
            Response.Flush();
        }
        blob.Close();
    }
    con.Close();
    con.Dispose();
}
```

Figura 6-19 – Código para leitura de um ficheiro da base de dados

Para a compreensão deste código é ainda necessário referir a chamada da função `setResponseHeaders`. Esta função determina o tipo do ficheiro a partir do seu nome, e preenche os *Headers* da página com o tipo de ficheiro a ser enviado e o seu tamanho. A realização desta operação é fundamental para que, do lado do cliente, se possa

determinar a percentagem do progresso de *download* do ficheiro e se apresente esse valor ao utilizador. O código que preenche os dados do *Header* da página é apresentado na Figura 6-20:

```
Response.AddHeader("Content-Disposition",
                   "inline;filename=" + filename);
Response.AddHeader("Content-Length", contentLength);
Response.ContentType = sContentType;
```

Figura 6-20 – Código para preenchimento dos 'headers'

O valor que é inserido em `ContentType` varia consoante o tipo de ficheiro e é esquematizado na tabela 6-2.

Extensão	Header
.dwf	Application/x-dwf
.pdf	Application/pdf
.doc	Application/vnd.ms-word
.ppt .pps	Application/vnd.ms-powerpoint
.xls	Application/vnd.ms-excel
.jpg .bmp .tif .png .gif	image/jpeg
outra	Application/octet-stream

Tabela 6-2 – 'ContentType' de diversos tipos de ficheiros

Capítulo 7

Conclusões e Trabalho Futuro

Os objectivos propostos para este projecto foram cumpridos. Tanto o Módulo de Apresentação dos Resultados como o Módulo de Inserção de Conteúdos foram desenvolvidos, apresentam todas as funcionalidades desejadas e cumprem todos os requisitos que foram definidos.

Como tal, o Módulo de Apresentação de Resultados permite a realização de pesquisas de acordo com diversos campos de pesquisa, apresentando os respectivos resultados de forma clara e funcional, e o Módulo de Inserção de Conteúdos permite, tal como foi solicitado, a inserção de documentos no *eResults*, realizando a respectiva associação de meta-informação.

Por sua vez, a capacidade de visualização e interacção com resultados multimédia, internamente aos módulos, é também suportada em ambos os módulos desenvolvidos, pelo que os vídeos e as imagens são manipuláveis, pelo utilizador, de uma forma simples, funcional e facilmente perceptível. Esta capacidade de visualização numa forma simples decorre do objectivo de desenvolver módulos com um elevado grau de usabilidade e, muito embora a avaliação do cumprimento, ou não, dum requisito deste tipo seja sempre muito difícil, é do entender do autor deste projecto e dos outros membros da equipa de desenvolvimento que ele foi efectivamente cumprido. Contudo, caberá aos futuros utilizadores a derradeira avaliação.

A tecnologia *Silverlight* revelou-se um precioso auxiliar para a criação de uma *RIA* de âmbito empresarial, com um suporte extenso a elementos multimédia, facilitando, em grande medida, as tarefas do programador no tratamento desse tipo de dados. Apesar disso, a ausência de maturidade da *Silverlight* e a instabilidade de que ainda sofre influenciou as opções de desenvolvimento do projecto e criou diversas dificuldades. De facto, ao longo do projecto foram usadas três versões da *Silverlight 2* e houve mesmo a necessidade de desenvolver controlos básicos de interface que de momento não são suportados pela *Silverlight*. Mas aquela que foi a maior dificuldade sentida, devido à ausência de maturidade da tecnologia, foi a documentação praticamente inexistente da *Silverlight* no início do projecto, facto que dificultou imenso o processo de aprendizagem da mesma.

Apesar dos objectivos deste projecto terem sido cumpridos, ainda existem diversas acções que podem e devem ser tomadas tendo em vista o melhoramento dos módulos desenvolvidos. O melhoramento da modularização do código produzido no projecto, para permitir uma grande percentagem de reutilização do mesmo em *WPF* é, para já, um dos pontos cuja necessidade já se determinou, na empresa.

Outras pequenas funcionalidades poderiam ser acrescentadas para aumentar a funcionalidade dos módulos desenvolvidos. São exemplos disso, entre outros, a criação de atalhos para as funcionalidades mais usadas, a disponibilização de mecanismos de ordenação dos resultados no modo de visualização de miniaturas, a criação de uma barra de ferramentas para a manipulação da imagem e a possibilidade de manipular a imagem em ecrã inteiro e apresentar o número de resultados associados a cada doente, quando uma pesquisa realizada retorna diversos doentes.

Bibliografia e Referências

1. CPC HealthcareSolutions, S.A., “Documento de Especificação Para Parceiro - Fornecimento de Solução para distribuição de Resultados Electrónicos de MCDT’s”, 2007
2. Microsoft Corporation, “XAML Overview (Silverlight 2)”,*MSDN - Microsoft Developer Network,* [consultado em: 2008-06-26]
   http://msdn.microsoft.com/en-us/library/cc189036(VS.95).aspx
3. Microsoft Corporation, “Code-behind and Partial Classes (Silverlight 2)”, *MSDN - Microsoft Developer Network* [consultado em: 2008-06-26]
   http://msdn.microsoft.com/en-us/library/cc221357(VS.95).aspx
4. Microsoft Corporation, “Key-Frame Animations Overview”, *MSDN - Microsoft Developer Network* [consultado em: 2008-07-05]
   http://msdn.microsoft.com/en-us/library/bb979740(VS.95).aspx,
5. Adobe Systems, “Flash Player Penetration” [consultado em: 2008-06-28]
   http://www.adobe.com/products/player_census/flashplayer/
6. Microsoft Corporation, “Grid Class”, *MSDN - Microsoft Developer Network* [consultado em: 2008-07-06]
   http://msdn.microsoft.com/pt-pt/library/system.windows.controls.grid(VS.95).aspx,
7. Microsoft Corporation, “Silverlight 2 Beta 2 - Development with the .NET Framework”, *MSDN - Microsoft Developer Network* [consultado em: 2008-06-13]
   http://msdn.microsoft.com/en-us/library/bb404700%28VS.95%29.aspx
8. Microsoft Corporation, “Silverlight Programming Models (Silverlight 2)”, *MSDN - Microsoft Developer Network* [consultado em: 2008-06-17]
   http://msdn.microsoft.com/en-us/library/cc189092(VS.95).aspx
9. Microsoft Corporation, “XAML Overview (Silverlight 2)”, *MSDN - Microsoft Developer Network* [consultado em: 2008-06-19]
   http://msdn.microsoft.com/en-us/library/cc189036(VS.95).aspx
10. Microsoft Silverlight, “Microsoft Silverlight FAQ” [consultado em: 2008-06-05]
    http://www.microsoft.com/silverlight/overview/faq.aspx
11. Adobe Systems, “Adobe Flash Player Features” [consultado em: 2008-07-01]

http://www.adobe.com/products/flashplayer/productinfo/features/,

12. Adobe Systems, “Adobe Flash Player 9 Datasheet” [consultado em: 2008-07-02]
    http://www.adobe.com/products/flashplayer/pdfs/Datasheet_Flash_Player_9_ue.pdf
13. Mozilla Developer Center, ”About JavaScript” [consultado em: 2008-06-29]
    http://developer.mozilla.org/en/docs/About_JavaScript
14. Sun Microsystems, “Applets”, *The Java Tutorials* [consultado em: 2008-06-28]
    http://en.wikipedia.org/wiki/Java_applet
15. World Wide Web Consortium, “Rich Web Clients Activity Statement” [consultado em: 2008-06-28]
    http://www.w3.org/2006/rwc/Activity.html,
16. Microsoft Corporation, “ScrollViewer Class”, *MSDN - Microsoft Developer Network* [consultado em: 2008-07-02]
    http://msdn.microsoft.com/pt-pt/library/system.windows.controls.scrollviewer.aspx

Anexos

A) Estrutura do *XML* para Transmitir Informação de Indexação

A informação que pode ser enviada neste formato para o serviço é descrita em seguida.

- Dados
    - Documento
        - Id – identificador único do documento
        - Titulo – título do documento
        - DescricaoDoc – descrição do documento
        - Instituicao – código da instituição que produziu o documento
        - Local – código do local
        - Origem – aplicação que produziu o documento
        - Tipo – tipo do documento
        - IdExterno – id associado ao documento relacionado com uma aplicação externa
        - DataDoc – data em que o documento foi gerado
        - Publico – parametrização do âmbito do documento
        - Estado – estado do documento
        - Utente
            - Id – identificador único do utente
            - Tdoente – tipo de doente
            - Nome – nome do utente
            - NProc – número do processo
            - NBenef – número de beneficiário
            - NBI – número do BI
            - NContrib – número de contribuinte

- DtNasc – data de nascimento
- Sexo – sexo do utente
- EstadoCivil – estado civil do utente
- CentroSaude – Não aplicável
- Morada – morada do utente
- Localidade
- CodigoPostal
- Telefone1
- Telefone2
- Fax
- Estado – estado do utente

- Visita
  - Episodio – Obrigatório
  - TipoEpisodio – Obrigatório
  - DtInicio – data do episódio
  - DtFim – data fim do episódio
  - Efr – entidade financeira responsável associada ao episódio
  - NrBenef – número de beneficiário associado à entidade financeira do episódio
  - VisitaDet
    - ServicoReq – serviço requisitante
    - ServicoExec – serviço executante
    - MedicoReq – número mecanográfico do médico requisitante
    - MedicoExec – número mecanográfico do médico executante
- Elemento
  - DescricaoElem – descrição do elemento (igual ao documento)
  - Estado – estado do elemento
  - Data – data do elemento (igual ao documento, usada para a verificação da versão)
  - Versão – versão do documento
  - Ordem – ordem do elemento
  - Ficheiro
    - NomeOriginal – nome do ficheiro
    - Extensao – extensão do ficheiro

- InfoXML – informação extra associada ao ficheiro no formato XML

- InfoRelevante
  - Info – informação relevante associada ao documento no formato XML
  - Tipo – tipo de informação enviada

Contudo, do ponto de vista do Módulo de Inserção de Conteúdos, será enviada muito menos informação. Um exemplo do ficheiro *XML* construído pelo Módulo para submeter ao serviço *ER2Indexer* apresenta-se em seguida.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Dados xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema”>"
  <Documento>
    <Id>200723451</Id>
    <Titulo>Relatório de Radiologia</Titulo>
    <Instituicao>CHVNG</Instituicao>
    <Local>CHVNG</Local>
    <Origem>RADIO</Origem>
    <Tipo>REL</Tipo>
    <DataDoc>26-05-2008 19:29:17</DataDoc>
    <Utente>
      <Id>6941223</Id>
    </Utente>
    <Visita>
      <Episodio>3848345</Episodio>
    </Visita>
    <Elemento>
      <DescricaoElem>Relatório de Radiologia</DescricaoElem>
      <Data>26-05-2008 19:29:17</Data>
      <Ficheiro>
        <NomeOriginal>200723451.pdf</NomeOriginal>
        <Extensao>PDF</Extensao>
        <InfoXML />
      </Ficheiro>
    </Elemento>
  </Documento>
</Dados>
```