Faculdade de Engenharia da Universidade do Porto

Windows Presentation Foundation no Processo Clínico

João Pedro Correia Gomes

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

Orientador: Jorge Alves da Silva (Professor Auxiliar)

Julho de 2008

Windows Presentation Foundation no Processo Clínico

João Pedro Correia Gomes

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

Aprovado em provas públicas pelo Júri:

Presidente: 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

O presente relatório tem como finalidade descrever o projecto ***Windows Presentation Foundation* no Processo Clínico**, realizado no âmbito do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto. Através deste projecto procurou-se avaliar a possibilidade de utilizar *Windows Presentation Foundation* para maximizar do ponto de vista da experiência de utilização soluções clínicas desenvolvidas na plataforma *.NET*. As soluções clínicas analisadas utilizavam, a nível de interfaces, *Windows Forms*. Este projecto tinha como principal objectivo determinar o custo da migração destas interfaces para *Windows Presentation Foundation*.

As soluções clínicas, por se tratar de sistemas críticos, foram sempre implementadas tendo como principais prioridades a disponibilidade, a fiabilidade, a eficiência e a segurança. Actualmente, estes atributos, por si só não são suficientes e toma particular importância a usabilidade, ou seja, a satisfação resultante da experiência de utilização destas aplicações por parte de todos os utilizadores que a elas recorrem para efectuarem o seu trabalho. No mercado, cada vez mais competitivo das soluções informáticas para a área da saúde, torna-se imperativo obter uma clara diferenciação pela qualidade da experiência de utilização, proporcionada pelas soluções desenvolvidas. A principal dificuldade que se coloca no cumprimento do objectivo principal deste projecto é, claramente, migrar as soluções, aumentando o nível da qualidade da experiência de utilização, mantendo ao máximo a qualidade nos pontos relativos à disponibilidade, fiabilidade, eficiência e segurança.

Para atingir o objectivo deste projecto, foi desenvolvido um plano que teve como primeira fase, analisar as soluções existentes implementadas com recurso a *Windows Forms*, estudar a interoperabilidade entre *Windows Forms* e *Windows Presentation*

Foundation e aferir o custo da migração de *Windows Forms* para *Windows Presentation Foundation*. Numa segunda fase, a abordagem ao problema consistiu em comprovar as conclusões retiradas na primeira fase, através do desenvolvimento de um protótipo. Devido ao sucesso da migração, foi decidido que este protótipo iria servir de base ao desenvolvimento de um módulo aplicável às soluções clínicas, para gerir informação relativa a exames clínicos. O desenvolvimento deste módulo irá decorrer para além do período de tempo em que decorreu o projecto.

As conclusões retiradas apontam para uma aposta cada vez mais vincada na *Windows Presentation Foundation* para melhorar a usabilidade das soluções clínicas.

Abstract

The objective of this report is to describe the project "***Windows Presentation Foundation* in the Clinical Process",** done under the scope of the Master in Informatics and Computing Engineering of the *Faculdade de Engenharia da Universidade do Porto*. The focus of this project was to evaluate the possibility to maximize the user experience in the clinical solutions developed in the *.NET* platforms with the usage of *Windows Presentation Foundation*. The analyzed clinical solutions used, at the interface level, *Windows Forms*. This project had as primary objective, the evaluation of the costs related to the migration of those interfaces to *Windows Presentation Foundation*.

Being critical systems, the clinical solutions, were always implemented having as primary priorities their availability, reliability, efficiency and safety. Nowadays, these attributes are not enough by themselves and the usability takes some importance. By usability is meant the satisfaction to the users resulting from the usage experience of the applications. In a health computer solutions market that is becoming each time more competitive, it becomes essential to have a clear differentiation by the user experience quality provided by the developed solutions. The main difficulty that has to be faced in the accomplishment of the main objective of this project is, clearly, the migration of the solutions, increasing the level of the usage experience's quality, keeping the maximum quality, in terms of availability, reliability, efficiency and security.

To achieve the project's objective, a working plan was developed that had as first phase the analysis of the current existing solutions implemented with *Windows Forms*, the study of the interoperability between *Windows Forms* and *Windows Presentation Foundation* and the determination of the migration costs from *Windows Forms* to *Windows Presentation Foundation*. In a second phase, the approach to the problem

consisted in the confirmation of the conclusions resulting from the first phase, through the development of a prototype. Due to the success of the migration, it was decided that the prototype will be used as base for the development of a module to manage the information of clinical exams, applicable to clinical solutions. The development of this module will occur beyond the time period of this project.

The conclusions taken from the project show that *Windows Presentation Foundation* will become more and more used to enhance the usability of clinical solutions.

Definições, Acrónimos e Abreviaturas

CAB (*Composite UI Application Block*) – *Framework* que foi desenhada para suportar o desenvolvimento de aplicações *Smart Client*.

Code-behind – Ficheiro escrito numa linguagem da *framework .NET* onde são realizadas operações de resposta a eventos e manipulação de objectos declarados no ficheiro *XAML*.

Control – Elemento da *UI* ou o conjunto de componentes da *UI* definidos e editados pelo programador.

CPC | HS – Companhia Portuguesa de Computadores- Healthcare Solutions, S.A.

CSS (*Cascading Style Sheets*) – Linguagem de estilo utilizada para definir a apresentação de documentos escritos numa linguagem de marcação, como *HTML* ou *XML*.

Experiência de Utilização – Termo utilizado para descrever a experiência e a satisfação sentidas, pelo utilizador, quando interage com uma aplicação.

Factory – ferramenta de geração automática de código que é comum às várias classes.

Infragistics – Empresa que disponibiliza, para venda, componentes de interface de utilizador (*UI*) para *Windows Forms* e *Windows Presentation Foundation*.

Interface de Utilizador (*UI*) – Interface através da qual os utilizadores interagem com a aplicação.

Layout – Espaço que contém e apresenta a distribuição física, tamanhos e cores de elementos como texto, gráficos, figuras, entre outros.

SCC – Smart Client Contrib.

SCSF – Smart Client Software Factory.

Smart Client – Uma aplicação *Smart Client* permite desenvolver *software* com o aspecto gráfico semelhante aos das aplicações *desktop* comuns. Para além disso é distribuída pela Internet e as actualizações são efectuados automaticamente sem a intervenção do utilizador.

Windows Forms – Tecnologia que disponibiliza acesso aos elementos nativos de interfaces do *Windows*.

WPF (*Windows Presentation Foundation*) – Tecnologia lançada pela *Microsoft* e que tem como objectivo melhorar significativamente a experiência de utilização das soluções.

XAML (*eXtensible Application Markup Language*) – Linguagem declarativa baseada em *XML* utilizada pelos programadores em *WPF*.

XHTML (*eXtensible Hypertext Markup Language*) – *XHTML* é uma reformulação da linguagem de marcação *HTML* baseada em *XML*.

Xmlns – Atributo para definição de *namespaces* na *XAML*.

Agradecimentos

Deixo os meus sinceros e profundos agradecimentos a todas as pessoas que muito me ajudaram ao longo da vida e que contribuíram em muito para a minha inspiração, de forma a realizar este projecto: pais, família, amigos e à minha musa, Catarina Pão Trigo. Estendo também estes agradecimentos às pessoas que contribuíram para a minha formação académica, desde professores a colegas. Quero também agradecer à Companhia Portuguesa de Computadores - Healthcare Solutions, S.A. por me ter proporcionado a oportunidade de realizar este projecto e a todas as pessoas que lá trabalham e que contribuíram para que este projecto tivesse sucesso. Por fim gostaria de deixar também um especial agradecimento aos meus orientadores, ao Professor Jorge Alves da Silva da Faculdade de Engenharia da Universidade do Porto e ao Engenheiro Nuno Filipe Reis Ribeiro da Companhia Portuguesa de Computadores - Healthcare Solutions, S.A., pelo apoio permanente ao longo do projecto.

Índice

ÍNDICE

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

Capítulo 1

Introdução

Este documento tem como objectivo descrever o projecto realizado no âmbito do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto. Este projecto, ***Windows Presentation Foundation* no Processo Clínico**, teve como foco principal a integração da tecnologia *WPF* (*Windows Presentation Foundation*) nas soluções informáticas na área dos Sistemas de Informação aplicados a tecnologias da saúde e gestão hospitalar.

Este capítulo tem como propósito dar a conhecer o enquadramento do projecto, a motivação e os objectivos que levaram à sua realização e os problemas que foram abordados. Por fim, será dada uma perspectiva da organização deste relatório e do conteúdo dos capítulos seguintes.

Pretende-se que o documento possua o menor número possível de termos técnicos e/ou referentes ao contexto em que se insere o projecto. Para os termos técnicos que o autor não conseguiu traduzir para uma linguagem mais acessível, é feita uma descrição, ou na secção Definições, Acrónimos e Abreviaturas, ou através de notas de rodapé.

1.1 Contexto/Enquadramento

O maior objectivo para quem desenvolve *software* foi, desde sempre, o de conseguir desenvolver produtos facilmente utilizáveis e que produzam uma experiência

de utilização[1] tão agradável quanto possível. Com o intuito de optimizar a interacção do utilizador com o produto, para que esta interacção vá de encontro às necessidades e actividades do utilizador, devem ser planificadas e implementadas as soluções tendo em vista o tipo de utilizador, o tipo de actividade e o contexto de interacção. As soluções devem ser desenvolvidas tendo em vista a sua usabilidade, ou seja, o grau com que um produto pode ser utilizado por utilizadores específicos, para alcançarem determinados objectivos com eficácia, eficiência e satisfação num determinado contexto de uso [IEEE98].

No presente, a usabilidade toma proporções decisivas na demarcação de qualidade de um produto, relativamente a produtos seus concorrentes. No mercado das soluções de *software*, o grau de usabilidade é crucial para a aceitação e sucesso de um produto junto dos utilizadores. A grande referência, ao nível de princípios de usabilidade, incide, actualmente, sobre a avaliação dos produtos tendo como principal incidência as dez heurísticas de Nielsen [NIE]:

1. Visibilidade do estado do sistema – O sistema deve manter sempre os utilizadores informados sobre o que se passa, através de *feedback* apropriado, num tempo aceitável.
2. Coerência entre o sistema e o mundo real – O sistema deve utilizar uma linguagem compreensível para o utilizador, que recorra a palavras, frases e conceitos que lhe sejam familiares, em vez de termos técnicos. Seguir as convenções do mundo real, fazendo com que a informação apareça de uma forma lógica e natural.
3. Controlo e liberdade do utilizador – Os utilizadores frequentemente escolhem funções do sistema erradamente e necessitam de uma "saída de emergência" claramente definida, para saírem de estados indesejados, sem serem confrontados com um diálogo extenso. Suportar os processos de avançar e retroceder.
4. Consistência e padrões – Os utilizadores não devem ter que pensar se diferentes palavras, situações ou acções significam a mesma coisa. Seguir padrões da plataforma.

[1] Experiência de utilização é o termo utilizado para descrever a experiência e a satisfação sentidas, pelo utilizador, quando interage com uma aplicação.

5. Prevenção de erros – Melhor do que mensagens de erro adequadas, é um *design* implementado de forma cuidadosa que previne um problema de acontecer em primeiro lugar. Eliminar condições que podem levar a erros ou procurá-las e apresentar aos utilizadores uma opção de confirmação antes de procederem à acção.
6. Reconhecimento preferível a recordação – Diminuir a utilização da memória por parte do utilizador, ao tornar os objectos, as acções e as opções visíveis. O utilizador não deve ter que relembrar informação de um diálogo para o outro. As instruções de uso do sistema devem ser visíveis e facilmente alcançáveis quando apropriado.
7. Flexibilidade e eficiência de uso – Aceleradores, geralmente imperceptíveis para um utilizador inexperiente, podem, várias vezes, aumentar a rapidez da interacção de um utilizador experiente para que o sistema se possa adequar tanto a um como a outro. Permitir aos utilizadores adaptarem-se a acções frequentes.
8. Design minimalista – Os diálogos não devem conter informação irrelevante ou raramente necessária. Cada unidade extra de informação num diálogo compete com uma unidade relevante de informação e diminui a sua visibilidade relativa.
9. Ajudar os utilizadores a reconhecer, diagnosticar e recuperar de erros – As mensagens de erro devem ser expressas em linguagem normal (sem conter código), indicar precisamente o problema e sugerir uma solução de forma construtiva.
10. Ajuda e documentação – Apesar de ser preferível que o sistema possa ser utilizado sem documentação, pode ser necessário disponibilizar ajuda e documentação. Esta informação deve ser fácil de procurar, focada na tarefa do utilizador, deve listar passos concretos a serem seguidos e não deve ser muito extensa.

É este o contexto em que se insere este projecto, de uma necessidade crescente de se aumentar a qualidade da experiência de utilização que uma aplicação pode proporcionar aos seus utilizadores. A criatividade e capacidade de inovação são factores indispensáveis para o sucesso, no desenvolvimento de *software*. Por maior grau de disponibilidade, fiabilidade, eficiência e segurança que uma aplicação disponha a sua qualidade irá ser sempre discutível, se a interacção com o utilizador não proporcionar um elevado nível de satisfação, se não atingir um grau elevado de usabilidade. O

sucesso de um produto é sempre definido pela qualidade da experiência geral de utilização por parte de um cliente [Res08].

1.2 Projecto

Este projecto foi realizado na empresa CPC | HS (Companhia Portuguesa de Computadores - Healthcare Solutions, S.A), empresa que tem como mercado alvo as instituições de saúde em funcionamento em Portugal e cujas actividades se distribuem por:

- Prestação de serviços, desenvolvimento, manutenção e suporte de aplicações informáticas na área dos Sistemas de Informação, com especial ênfase no domínio das Tecnologias da Saúde e Gestão Hospitalar.
- Licenciamento, implementação, parametrização, formação e consultoria, seja de produtos próprios ou representados.
- Consultoria e gestão de projectos.
- Venda de soluções integradas de Sistemas de Informação.

Este projecto surgiu da necessidade da empresa CPC | HS melhorar as suas aplicações destinadas à área clínica, no sentido do ponto de vista da experiência de utilização. Para atingir este objectivo era necessário avaliar as potencialidades da tecnologia *WPF*, como o caminho a seguir para atingir esse fim. Esta é, actualmente, a tecnologia de eleição da *Microsoft* para o desenvolvimento de interfaces de utilizador. As solução actuais da empresa estão desenvolvidas na plataforma *.NET*, sob a configuração de um *Smart Client*, utilizando uma arquitectura orientada a serviços (SOA), implementada à custa de *Web Services*.

1.3 Motivação e Objectivos

Actualmente as soluções informáticas na área dos Sistemas de Informação que servem de suporte às actividades relacionadas com a saúde utilizam, a nível de interfaces, *Windows Forms*. Esta tecnologia está actualmente desactualizada relativamente à sua tecnologia sucessora, a *WPF*. Na procura constante de melhorar a qualidade dos seus produtos, a CPC | HS sentiu necessidade de fazer um

aperfeiçoamento das suas soluções, ao nível da usabilidade e das próprias funcionalidades. Surgiu, desta forma, a necessidade de estudar a possibilidade de se utilizar a *WPF* como a tecnologia que daria resposta a estas necessidades.

Foram objectivos deste projecto avaliar claramente os pontos fortes e fracos da *WPF*, o custo da migração, ou seja, o tempo que seria necessário despender e a dificuldade associada a esta migração. Por fim, pretendia-se também avaliar experimentalmente os custos da migração, através do desenvolvimento de um protótipo. Como resultado do sucesso da migração, este protótipo resultou posteriormente num módulo aplicável às soluções clínicas, para gerir informação relativa a exames clínicos.

Para serem atingidos estes objectivos, foi estruturado um plano que teve como primeira fase estudar a interoperabilidade entre *Windows Forms* e *WPF* e o custo da migração de *Windows Forms* para *WPF*. Numa segunda fase, a abordagem ao problema consistiu em comprovar as conclusões retiradas através do desenvolvimento de um protótipo, com recurso a *WPF* e aos conhecimentos retirados da primeira fase do plano.

A descrição deste plano é feita com maior detalhe no capítulo seguinte, Análise do Problema.

1.4 Estrutura do Relatório do Projecto

Este relatório está estruturado em cinco capítulos. Neste primeiro capítulo fez-se o enquadramento do problema. No segundo capítulo, denominado Análise do Problema, é feita uma descrição e uma análise profunda do estudo e dos problemas abordados neste projecto. Seguidamente, no capítulo Revisão Tecnológica, é feito um estudo das tecnologias utilizadas ao longo do projecto. No capítulo seguinte, Descrição dos Principais Detalhes do Processo de Migração, são analisadas as soluções encontradas para os problemas identificados neste projecto. Por fim todo o trabalho envolvente ao projecto será revisto e analisado sob a forma de conclusões e será ainda feita a referência ao impacto dos resultados deste projecto no capítulo Conclusões e Trabalho Futuro.

Capítulo 2

Análise do Problema

Este projecto tinha como objectivos averiguar a viabilidade da migração de soluções em *Windows Forms* para *WPF*, os seus custos, quais as suas vantagens e qual a melhor forma de ser realizada esta migração, bem como estudar a interoperabilidade entre *Windows Forms* e *WPF*.

Em primeiro lugar foi necessário estudar as soluções clínicas da CPC | HS, desenvolvidas em *Windows Forms*, de forma a averiguar em que pontos as interfaces poderiam ser melhoradas. Estas soluções estão implementadas sobre uma *framework* que permite desenvolver aplicações do tipo *Smart Client*, a *CAB* (*Composite UI Application Block*), e utilizando componentes da *Infragistics* para *Windows Forms*. Foi necessário analisar as implicações da utilização dos componentes da *Infragistics* e qual o seu impacto na integração com *WPF*. No seguimento desta análise, colocou-se outra questão que se prendia com a possibilidade de utilização dos componentes da *Infragistics* para *Windows Forms* simultaneamente com *WPF*.

Com o decorrer do projecto e, nomeadamente, desta fase, outras questões se tornaram particularmente relevantes como o estudo da *SCSF* (*Smart Client Software Factory*), que fornece auxílio para programadores e arquitectos de *software*, uma vez que engloba já a *CAB* e gera o código comum a todos os componentes dos *Smart Clients*, diminuindo assim substancialmente o tempo de implementação das soluções. Tendo em conta as suas propagadas vantagens, tornou-se fulcral para o projecto analisar

se seria vantajosa a sua utilização e a utilização das suas extensões desenvolvidas no projecto *SCC* (*Smart Client Contrib*).

Foram analisadas outras possíveis vantagens para o projecto: a utilização da ferramenta *Microsoft Expression Blend*, para maximizar a qualidade das interfaces e a possibilidade de se expandir globalmente o *software* através de "localização" na *WPF*, ou seja, a capacidade de traduzir os recursos da aplicação conforme o idioma pretendido.

O estudo das soluções (para os problemas) pôde ser complementado com protótipos que cimentaram conhecimentos e serviram, ao mesmo tempo, de casos de teste. O estudo centrou-se nas tecnologias *Windows Forms* e *WPF*, a qual engloba a linguagem *XAML*. Este estudo cobriu, também, ferramentas como a *CAB*, a *SCSF*, a *Microsoft Expression Blend* e os componentes *UI* da *Infragistics* utilizados pelas soluções. No decorrer deste estudo, foram analisados os códigos fonte quer da *CAB*, quer da biblioteca da *Infragistics*, que permite a integração dos componentes da *Infragistics* na *CAB*, de maneira a perceber se existiriam problemas na integração com a *WPF* e, em caso afirmativo, como poderiam estes ser ultrapassados. O estudo contemplava também testes para aferição do grau de sucesso da migração nas soluções actuais e o desenvolvimento das infra-estruturas de suporte necessárias à migração das soluções em *Windows Forms* para *WPF*.

Por fim, foi objectivo desenvolver um protótipo, baseado em *WPF* e seguindo as directrizes retiradas ao longo do projecto, com a finalidade de comprovar as mesmas. Com o decorrer do projecto, este protótipo passou a ser desenvolvido com o intuito de integrar as soluções actuais, tornando-se assim num módulo a ser desenvolvido para além do período de tempo em que decorreu este projecto.

Capítulo 3

Revisão Tecnológica

Neste capítulo descrevem-se as tecnologias utilizadas, ao longo deste projecto, e que foram alvo de estudo. O estudo, aqui descrito, foi absolutamente necessário para se poderem cumprir os objectivos traçados. As secções seguintes são o resultado de uma investigação profunda do autor deste projecto. Este estudo foi a base para compreender o contexto em que se inserem os problemas, que este projecto abordou, e para se conseguir ultrapassar estes problemas. Permitiu ainda retirar conclusões relativamente aos pontos fortes e fracos da *WPF* e produzir um manual de utilização interno, da *WPF*, para a empresa.

A tecnologia à qual será dada maior ênfase é a *WPF*, que vai ser analisada de uma forma mais pormenorizada do que as restantes tecnologias, devido ao seu papel chave neste projecto. Grande parte do resultado deste estudo é descrito na secção 3.1. Todos os exemplos apresentados foram concebidos pelo autor deste projecto, sobre a forma de protótipos.

Outra tecnologia que será analisada com pormenor é a *CAB*, pois todas as soluções a migrar estão desenvolvidas sobre esta *framework*, tornando-se assim fundamental ter um conhecimento profundo sobre a mesma, de forma a ser atingido o objectivo principal deste projecto: determinar o custo associado à migração das soluções actuais para *WPF*.

3.1 *Windows Presentation Foundation*

A *Windows Presentation Foundation* (*WPF*) apareceu em 2001 sob o nome "Avalon", como a nova tecnologia de apresentação do *Windows Vista*. As suas principais características são:

- Flexibilidade da interface, que pode ser independente do código.
- Incorporação de todas as funções do *.NET 2.0*, acrescentando às interfaces novos recursos como 3D, animações, gráficos vectoriais, reconhecimento de voz, *layouts* avançados, entre outros.
- Implementação do conceito já existente de separação entre o *design* e o código, permitindo separar os processos de implementação de interfaces (*designers*) e de escrita de código (programadores).
- Utilização dos recursos do sistema em que se encontra a operar, de forma a optimizar a performance da interface tendo em conta o hardware da máquina em que está a ser executado.
- Independência da plataforma em que é executado.

Um programa/aplicação que usa *WPF* é normalmente dividido em duas partes:

- Ficheiros *XML* com características especiais chamados *XAML* (*eXtended Aplication Markup Language*).
- Ficheiros de código para *.NET* (escrito em qualquer linguagem compatível, *VB.net*, *C#*, etc).

O arquivo *XAML* contém as directrizes de interface, podendo ser comparado ao *XHTML* em relação ao *ASP.net*.

3.1.1 *XAML*

XAML (*e**X**tensible **A**pplication **M**arkup **L**anguage*) é uma linguagem declarativa baseada em *XML*, utilizada pelos programadores em *WPF* para implementar o *layout* de uma interface de utilizador (*UI*) e os recursos utilizados nessa interface de utilizador.

Quando uma aplicação *WPF* é compilada no *Visual Studio* os ficheiros *XAML* são compilados sob a forma de ficheiros *.baml* (*Binary Application Markup Language*) e

são guardados como recursos no *assembly* resultante. Em *run-time* são extraídos dos recursos do *assembly*, são efectuadas funções de *parse* sobre eles e são geradas as resultantes árvores visuais ou *workflows WPF*.

Tudo o que é implementado em *XAML* pode ser igualmente desenvolvido noutras linguagens *.NET*, como o *C#* ou o *Visual Basic .NET*. A utilização de *XAML* facilita o processo de desenvolvimento e torna-o mais rápido.

Sintaxe

XAML é uma linguagem baseada em *XML* e são dois os diferentes tipos de utilização dos elementos *XML* em *XAML*: elementos que representam objectos e elementos que definem as propriedades de objectos. O código em *XAML*, apresentado ma Figura 1, ilustra a afirmação anterior.

```
<Button Content="Open" Width="70" Height="70" Click="OnButtonClick">
      <Button.Background>
            <LinearGradientBrush>
                  <GradientStop Color="Aqua" Offset="0" />
                  <GradientStop Color="Yellow" Offset="1" />
            </LinearGradientBrush>
      </Button.Background>
</Button>
```

Figura 1: Código exemplo XAML

Figura 2: Resultado do código referente às Figuras 1 e 34

O elemento `<Button>` define uma instância da classe `Button`. Este objecto `Button` vai ter a sua propriedade `Content` definida pela *string* "Open"; o valor das propriedades `Width`, `Height` e `Click` também foi definido. O elemento seguinte

`<Button.Background>` representa a propriedade `Background` do `Button` que estamos a implementar. O elemento filho de um elemento propriedade é o valor para o qual a propriedade será definida, ou seja, neste caso a propriedade `Background` será definida pelo objecto `LinearGradientBrush`. Por último, convém referir que o evento `Click` irá invocar o código que reside no ficheiro (*C#* ou *VB.NET*), geralmente, associado ao ficheiro *XAML*. O ficheiro *XAML* é compilado numa classe. O nome desta classe é definido por omissão pelo compilador ou definido pelo programador através do atributo `x:Class` no elemento *XML* da raíz do ficheiro *XAML*, podendo ser criada posteriormente uma "partial class", nas linguagens anteriormente referidas, que permite associar acções à interface de utilizador definida no ficheiro em *XAML*.

Supondo que a classe referida se chama `MyWindow,` o código da classe em *C#* que permitiria activar a acção associada ao `Button` seria o que se apresenta na Figura 3.

```
public partial class MyWindow : Window
        {
            public MyWindow()
            {
                InitializeComponent();
            }
            void OnButtonClick(object sender, RoutedEventArgs e)
            {
                MessageBox.Show("It's open!");
            }
        }
```

Figura 3: Código da classe em C# que permite activar a acção associada ao `Button`

É possível constatar agora com facilidade uma das características da *WPF*: a vantagem de ser possível separar por completo os processos de *design* e de desenvolvimento de funcionalidades.

3.1.2 *Layout*

O *layout* é uma das partes mais importantes de qualquer projecto em *WPF*. Ao utilizar os *layout controls* disponibilizados pelo *WPF* os programadores/*designers* podem desenvolver complexas interacções entre *controls*/páginas.

Os *layout controls* mais importantes são:

- `Grid`.
- `Canvas`.
- `StackPanel`.
- `WrapPanel`.
- `DockPanel`

A propriedade `Margin` quando aplicada a um *control* define a margem que este tem à sua volta. Por exemplo, em `Margin="3,4,5,6"` o *control* terá margens de 3 pixeis à sua esquerda (“Left”), 4 para cima (“Top”), 5 à sua direita (“Right”) e 6 para baixo (“Bottom”).

3.1.2.1 *Grid*

O *control* `Grid` pode ser equiparado a um *table control* em *HTML*. Podem ser especificadas linhas (“rows”) e colunas (“columns”), bem como células que ocupam mais de uma linha ou coluna. Nas definições de largura (“width”) e altura (“height”) pode ser utilizada a notação “*” que em termos práticos significa o “restante disponível”. Esta situação pode ser melhor compreendida através do exemplo da Figura 4, onde a primeira coluna tem uma largura fixa de 50 pixeis, a segunda tem uma largura fixa de 20 pixeis e a terceira ocupa o restante espaço disponível (*).

```
<Grid.ColumnDefinitions>
      <ColumnDefinition Width="50" />
      <ColumnDefinition Width="20" />
      <ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
```

Figura 4: Exemplo de código de definição de largura numa Grid

Para definir a que células pertencem os elementos filhos de um `Grid` *control* são utilizadas as seguintes propriedades:

- `Grid.Column`.
- `Grid.Row`.

E para definir quantas colunas e linhas deve uma célula ocupar usam-se as propriedades:

- `Grid.ColumnSpan`.
- `Grid.RowSpan`.

No código definido na Figura 5, que exemplifica a utilização deste *control*, o objecto `Rectangle` com `Fill` definido para "Green" ocupa as duas colunas da direita (`Grid.ColumnSpan="2"`).

```
<Grid>
      <Grid.ColumnDefinitions>
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="20" />
            <ColumnDefinition Width="*"/>
      </Grid.ColumnDefinitions>
      <Rectangle Fill="Yellow" Grid.Column="0"/>
      <Rectangle Fill="Green" Grid.Column="1" Grid.ColumnSpan="2"/>
</Grid>
```

Figura 5: Exemplo de utilização de Grid

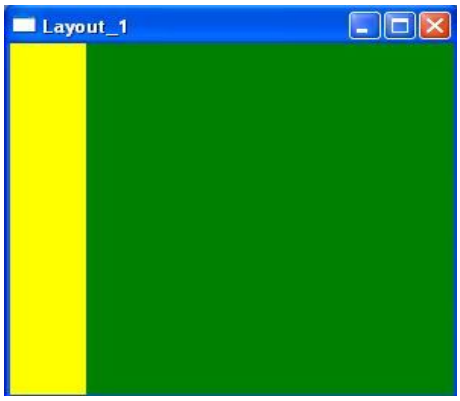

Figura 6: Resultado do código referente à Figura 5

3.1.2.2 *Canvas*

A `Canvas` é um *layout* simples onde os objectos são posicionados através das coordenadas x/y. Os *controls* filhos devem especificar pelo menos uma das quatro propriedades para serem devidamente posicionados na `Canvas`:

- `Canvas.Left`.
- `Canvas.Right`.
- `Canvas.Top`.
- `Canvas.Bottom`.

No exemplo ilustrado na Figura 7 o `Rectangle` encontra-se a 70 pixeis de distância do lado esquerdo da `Canvas` (`Canvas.Left`) e a 50 pixeis de distância do topo da `Canvas` (`Canvas.Top`), a `Ellipse` é definida de forma semelhante.

```
<Canvas Margin="0,0,0,0" Background="White">
    <Rectangle Fill="LightBlue"
        Stroke="LightBlue"
        Width="145"
        Height="126"
        Canvas.Left="70" Canvas.Top="50"/>

    <Ellipse Fill="LightGreen"
        Stroke="LightGreen"
        Width="121" Height="100"
        Panel.ZIndex="1"
        Canvas.Left="150" Canvas.Top="161"/>
</Canvas>
```

Figura 7: Exemplo de utilização da Canvas

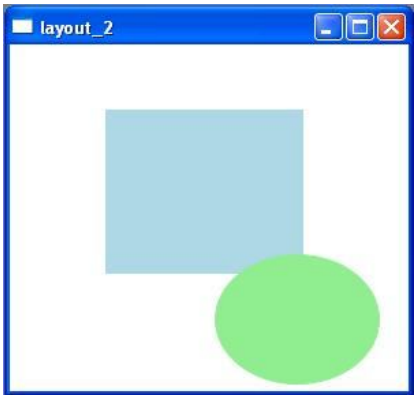

Figura 8: Resultado do código referente à figura 7

Como os objectos se sobrepõem, podemos decidir qual destes dois objectos se encontrará no topo, através da propriedade `Panel.ZIndex`, sendo os objectos dispostos de acordo com o valor desta propriedade: quanto maior o seu valor na coordenada z. Se não for declarada esta propriedade em nenhum dos objectos, o `ZIndex` será incrementado automaticamente nos objectos pela ordem em que eles são declarados, ou seja, se removermos a propriedade `Panel.ZIndex="1"` na `Ellipse` o resultado seria o mesmo.

3.1.2.3 *StackPanel*

O `StackPanel` funciona, tal como o nome indica, como uma *stack* (vertical ou horizontal) de ordenamento dos seus filhos, utilizando para isso a propriedade `Orientation`. O código da Figura 9 ilustra um exemplo de utilização deste *layout control*.

```
<StackPanel Margin="0,0,0,0" Background="White"
Orientation="Vertical">
    <Button Content="Top of the Stack"/>
    <Button Content="Bottom of the Stack"/>
</StackPanel>
```

Figura 9: Exemplo de utilização de StackPanel

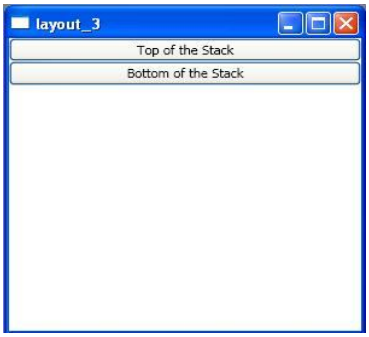

Figura 10: Resultado para o código referente à Figura 9

3.1.2.4 *WrapPanel*

Num `WrapPanel` os componentes são dispostos de forma a conseguirem ocupar o espaço disponível. Ilustrando através de um exemplo de utilização, as Figuras 11 e 12 apresentam, respectivamente, o código e o resultado da execução deste código, enquanto que a Figura 13 ilustra o resultado do aumento da `Width` deste *layout control*.

```
<WrapPanel Margin="0,0,0,0" Background="White">
    <Rectangle Margin="10,10,10,10" Fill ="Blue" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="Aqua" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="BurlyWood" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="Cornsilk" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="Firebrick" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="Goldenrod" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="LightGreen" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="MediumVioletRed" Width="60"
Height="60"/>
    <Rectangle Margin="10,10,10,10" Fill ="Orange" Width="60"
Height="60"/>
</WrapPanel>
```

Figura 11: Exemplo de utilização de WrapPanel

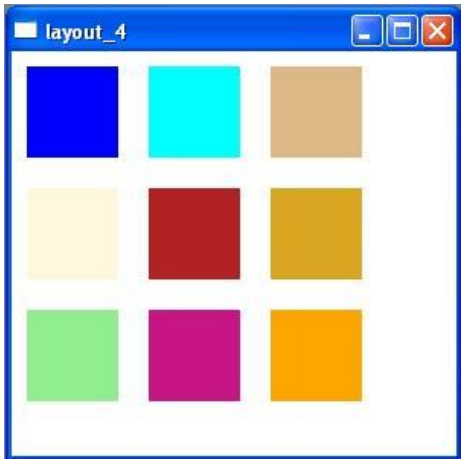

Figura 12: Resultado do código da Figura 11

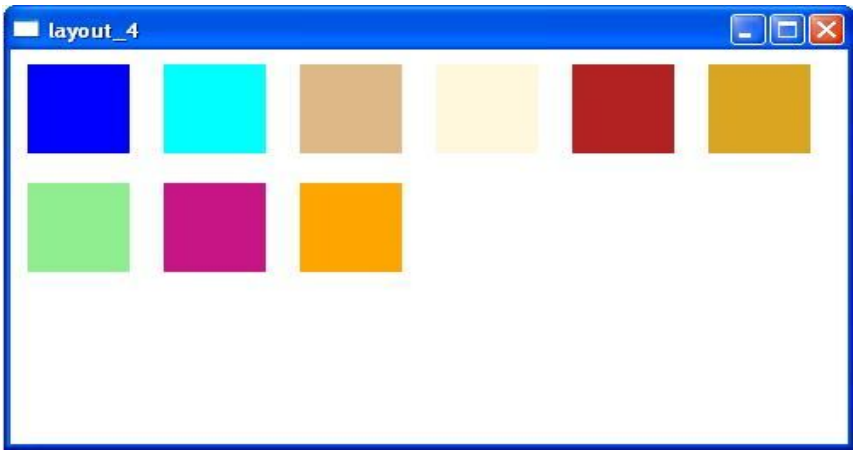

Figura 13: Exemplo de aumento de largura num WrapPanel

3.1.2.5 *DockPanel*

Por fim, o `DockPanel` é um dos *layouts* mais úteis. Permite-nos distribuir os seus componentes de forma muito simples através da propriedade `DockPanel.Dock` que pode ser definida como:

- `DockPanel.Dock="Top"`.
- `DockPanel.Dock="Bottom"`.
- `DockPanel.Dock="Right"`.
- `DockPanel.Dock="Left"`.

Como os nomes desta propriedade indicam, os componentes são distribuídos conforme a posição definida. Outra propriedade importante é a `LastChildFill` que quando está definida com o valor "true", no `DockPanel`, faz com que o último filho ocupe o restante espaço disponível. O código da Figura 14 e o respectivo resultado ilustrado na Figura 15 documentam um exemplo de utilização de um `DockPanel` onde são dispostos dois objectos `Rectangle`.

```
<DockPanel Width="Auto" Height="Auto" LastChildFill="True">
    <Rectangle Fill="LightBlue" Stroke="LightBlue"
        Height="50" DockPanel.Dock="Top"/>
    <Rectangle Fill="Yellow" Stroke="Yellow"/>
</DockPanel>
```

Figura 14: Exemplo de utilização de DockPanel

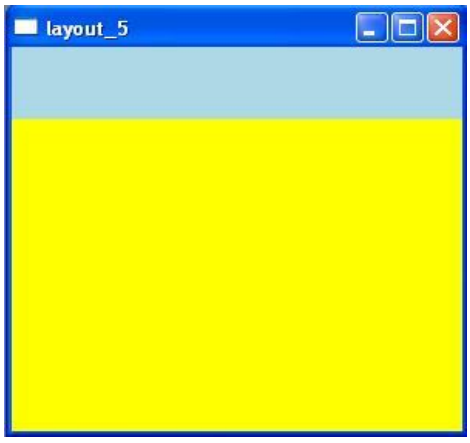

Figura 15: Resultado da execução do código da Figura 14

3.1.3 Recursos

3.1.3.1 Como Referenciar *Controls*/Classes em *XAML*

Existem quatro situações em que pode ser necessário referenciar *controls*/classes em *XAML*:

- Referenciar um *control* local (mesmo *namespace*).
- Referenciar uma classe local (mesmo *namespace*).
- Referenciar um *control* externo (*namespaces*/projectos diferentes).

- Referenciar uma classe externa (*namespaces*/projectos diferentes).

3.1.3.1.1 Referenciar um *control* local (mesmo *namespace*)

Para referenciar localmente um *control* basta utilizar uma declaração de *xmlns* na *tag* do *control* pai. Ilustrando através de um exemplo, considere-se o seguinte cenário representado na Figura 16, no qual uma `Window`, "Window1", pretende referenciar um *user control* local ("UserControl1"). É necessário adicionar a declaração de *xmlns* da Figura 17 na classe "Window1", definida na Figura 18. Neste cenário "UserControl1" está definido conforme indicado na Figura 19 e o resultado é o exposto na Figura 20.

Figura 16: Cenário de exemplo de como referenciar localmente um Control

```
xmlns:local="clr-namespace:resources_1;assembly="
```

Figura 17: Declaração do atributo xmlns para o exemplo em que é referenciado localmente um control

```
<Window x:Class="resources_1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:resources_1;assembly="
    Title="resources_1" Height="300" Width="300"
WindowStartupLocation="CenterScreen">
  <Grid>
    <local:UserControl1 />
  </Grid>
</Window>
```

Figura 18: Definição da "Window1" que referencia localmente um control

```
<UserControl x:Class="resources_1.UserControl1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <Grid>
      <Ellipse Width="200" Height="150">
        <Ellipse.Fill>
          <LinearGradientBrush>
            <GradientStop Offset="0" Color="Teal"/>
            <GradientStop Offset="1" Color="Aqua"/>
          </LinearGradientBrush>
        </Ellipse.Fill>
      </Ellipse>
</Grid>
</UserControl>
```

Figura 19: Definição do "UserControl1" que é acedido localmente

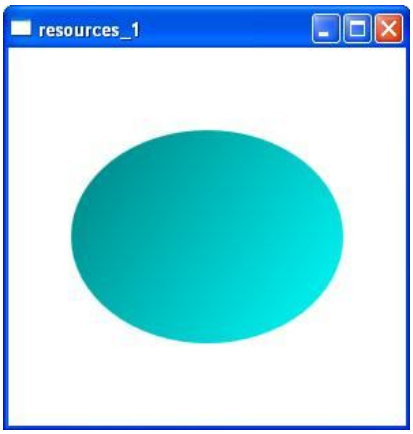

Figura 20: Resultado para o exemplo em que é referenciado um control local (mesmo namespace)

3.1.3.1.2 Referenciar uma Classe Local (Mesmo *Namespace*)

Da mesma forma, para referenciar localmente uma classe basta utilizar uma declaração de *xmlns* na *tag* do *control* pai. Ilustrando novamente, através de um exemplo, considere-se o cenário da Figura 21 onde uma `Window`, "Window1", pertencente à solução, pretende referenciar uma classe local ("Class1"). Para aceder à "Class1" definida na Figura 22 é necessário adicionar a declaração de *xmlns* da Figura 23 na classe "Window1", definida na Figura 24. É apresentado, na Figura 25, um

exemplo de como aceder a este objecto no *code-behind* e na Figura 26 é ilustrado o resultado da função `MessageBox.Show` da Figura 25.

Figura 21: Cenário do exemplo de como referenciar uma classe local

```
public class Class1
    {
        private int example;

        public int Example
        {
            get { return example; }
            set { example = value; }
        }

        public Class1()
        {
        }
    }
```

Figura 22: Definição da "Class1" referenciada localmente

```
xmlns:local="clr-namespace:resources_1;assembly="
```

Figura 23: Declaração do atributo xmlns para o exemplo em que é referenciada uma classe localmente

```
<Window x:Class="resources_2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:resources_2;assembly="
    Title="resources_2" Height="300" Width="300">
    <Grid x:Name="grid">
      <Grid.Resources>
        <local:Class1 x:Key="class1" Example="3"/>
      </Grid.Resources>
    </Grid>
</Window>
```

Figura 24: Definição da "Window1" que referencia uma classe local

```
public partial class Window1 : System.Windows.Window
    {

        public Window1()
        {
            InitializeComponent();
            Class1 class1 = grid.FindResource("class1") as Class1;
            int value = class1.Example;
            MessageBox.Show(value.ToString());
        }

    }
```

Figura 25: Exemplo de code-behind que vai ser aceder a "Class1" (Figura 22)

Figura 26: Resultado da função `MessageBox.Show` da Figura 25

3.1.3.1.3 Referenciar um *Control* Externo (*Namespaces*/Projectos Diferentes)

Neste caso o `UserControl` e a `Window` residem em projectos diferentes, logo é necessário referenciar o projecto onde reside o `UserControl` na `Window`. Recorrendo, mais uma vez, a um exemplo, considere-se a seguinte situação ilustrada na Figura 27, na qual "Window1", definida na Figura 28, pretende referenciar o "UserControl1" que reside no projecto "example_project". Para isso, é necessário adicionar a definição de *xmlns* presente na Figura 29. Neste cenário, o "UserControl1" é especificado na Figura 30 e o resultado de execução para "Window1" ilustrado na Figura 31.

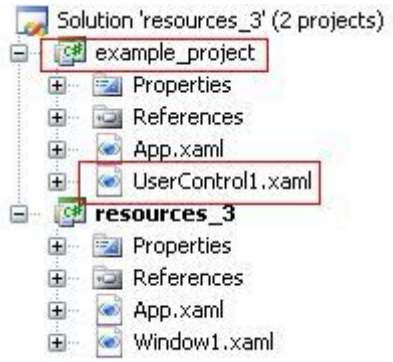

Figura 27: Cenário do exemplo de como referenciar um control externo

```
<Window x:Class="resources_3.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:example_project="clr-
namespace:example_project;assembly=example_project"
    Title="resources_3"            Height="300"           Width="300"
WindowStartupLocation="CenterScreen"
    >
    <Grid>
      <example_project:UserControl1/>
    </Grid>
</Window>
```

Figura 28: Definição da "Window1" que referencia um control externo

```
xmlns:example_project="clr-
namespace:example_project;assembly=example_project"
```

Figura 29: Declaração do atributo xmlns para o exemplo em que é referenciado um control externo

```
<UserControl x:Class="example_project.UserControl1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Height="300" Width="300"
    >
    <Grid>
      <Rectangle Fill="Blue" Height="100" Width="100"/>
    </Grid>
</UserControl>
```

Figura 30: Definição do "UserControl1" que vai ser acedido externamente

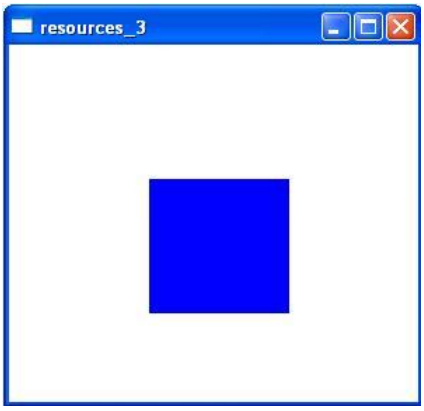

Figura 31: Resultado de execução para "Window1" ilustrada na Figura 28

3.1.3.1.4 Referenciar uma Classe Externa (*Namespaces*/Projectos Diferentes)

O modo de referenciar uma classe externa é semelhante ao de referenciar uma classe local. Supondo que se pretende referenciar as classes "System.Collections.Hashtable" e "System.Int32" da biblioteca "mscorlib.dll", é preciso declarar num *control* os atributos *xmlns*, tal como definido na Figura 32. Neste cenário temos por exemplo uma `Window`, "Window1", especificada na Figura 33.

```
xmlns:collections="clr-namespace:System.Collections;assembly=mscorlib"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

Figura 32: Declaração dos atributos xmlns para o exemplo em que é referenciada uma classe externa

```
<Window x:Class="resoures.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:collections="clr-
namespace:System.Collections;assembly=mscorlib"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    Title="resoures_4" Height="300" Width="300"
    >
    <Grid Name="grid">
      <Grid.Resources>

        <collections:Hashtable x:Key="ht1">
          <sys:Int32 x:Key="key1">1</sys:Int32>
          <sys:Int32 x:Key="key2">2</sys:Int32>
        </collections:Hashtable>

      </Grid.Resources>

    </Grid>
</Window>
```

Figura 33: Definição da "Window1" que referencia uma classe externa

3.1.3.2 *Markup Extensions*

O *parser* de *XAML* reconhece *Markup Extensions*. *Markup Extensions* permitem-nos configurar objectos e aceder a objectos que estão definidos noutros *controls*. São usadas para definir propriedades em objectos. São usadas também para definir um valor `null`, para declarar um *array*, para referenciar um objecto contido num `ResourceDictionary` e para *data binding*, entre outras coisas. No exemplo ilustrado na Figura 34, a propriedade `Background` do `Button` é definida através de um atributo *XML* `{StaticResource Brush}`. A classe `StaticResourceExtension` é utilizada para

referenciar o `LinearGradientBrush` contido no `ResourceDictionary` da `<Grid>`. Um valor entre chavetas "{}", quando aplicado a um atributo, é sempre uma *Markup Extension*. O resultado, para este exemplo, encontra-se ilustrado na Figura 2 da página 10.

```
<Grid>
      <Grid.Resources>
            <LinearGradientBrush x:Key="Brush">
                  <GradientStop Color="Aqua" Offset="0" />
                  <GradientStop Color="Yellow" Offset="1" />
            </LinearGradientBrush>
     </Grid.Resources>
     <Button Background="{StaticResource Brush}" Width="70"
Height="70" Click="OnButtonClick">Open</Button>
</Grid>
```

Figura 34: Exemplo de utilização de Markup Extensions

3.1.3.3 Declarar e Aceder a Recursos

Recursos (`Resources`) são objectos reutilizáveis que podem ser declarados em vários locais:

- Na aplicação, ou seja, são declarados no "App.xaml" ou equivalente. Esta declaração é global.
- Nos `Resources` da `Window` (ou equivalente) local.
- Na propriedade `Resources` de qualquer `FrameworkElement` ou `FrameworkContentElement`.
- Num ficheiro *XAML* à parte.

Os `Resources` podem ser estáticos ou dinâmicos, ou seja, podem ser alterados em *run-time* no caso de serem declarados como `DynamicResource`; inversamente, se forem declarados como `StaticResource` não podem ser alterados em *run-time*.

3.1.3.3.1 *Application Level Resource*

Para declarar um `Resource` global basta adicioná-lo ao ficheiro da aplicação, mais concretamente aos `Resources` da aplicação. A Figura 35 descreve um exemplo de um `Resource` declarado ao nível da aplicação. Para referenciar esse `Resource` basta utilizar uma *Markup Extension*, tal como no exemplo da Figura 36.

```
<Application.Resources>
      <SolidColorBrush x:Key="applicationLevelResource" Color="Blue"/>
</Application.Resources>
```

Figura 35: Recurso declarado ao nível da aplicação

```
<Button Background="{StaticResource applicationLevelResource}"
Content="Button in a Application Resource"></Button>
```

Figura 36: Exemplo de botão que acede a um recurso que se encontra ao nível da aplicação

3.1.3.3.2 *Window Level Resource*

Para declarar um `Resource` numa `Window` (ou equivalente) local basta adicioná-lo aos `Resources` da `Window`. A figura 37 descreve um exemplo de um `Resource` declarado ao nível de uma `Window`. Para referenciar esse `Resource` basta utilizar uma *Markup Extension* tal como no exemplo da figura 38.

```
<Window.Resources>
    <SolidColorBrush x:Key="windowLevelResource" Color="Red"/>
</Window.Resources>
```

Figura 37: Recurso declarado ao nível de um control pai

```
<Button Background="{StaticResource windowLevelResource}"
Content="Button in a Window Resource"></Button>
```

Figura 38: Exemplo de botão que acede a um recurso que se encontra ao nível de um control pai

3.1.3.3.3 *Framework Level Element Resource*

Para declarar um `Resource` num `FrameworkElement` ou `FrameworkElementHost` basta adicioná-lo aos `Resources` dos próprios. O `Resource` apenas pode ser acedido pelos seus filhos. O exemplo descrito na Figura 39 onde `Button` acede a um `Resource` de um `StackPanel` ilustra esta temática.

```
<StackPanel>
    <StackPanel.Resources>
      <LinearGradientBrush x:Key="frameworkElementLevelResource">
        <GradientStop Color="Aqua" Offset="0" />
        <GradientStop Color="Yellow" Offset="1" />
      </LinearGradientBrush>
    </StackPanel.Resources>
    <Button Background="{StaticResource
frameworkElementLevelResource}" Content="Button in a Framework Element
Resource"></Button>
</StackPanel>
```

Figura 39: Exemplo de botão que acede a um recurso da StackPanel em que está inserido

3.1.3.3.4 *Separate Loose XAML Resource*

Para declarar um `Resource` num ficheiro *XAML* independente temos de criar um ficheiro XAML do tipo *ResourceDictionary*. Ilustrando através de um exemplo, considere-se o seguinte cenário, em que num *ResourceDictionary*, "Dictionary1", é declarado um `Resource`, tal como descrito na Figura 40. Este `Resource` é acedido tal como no exemplo especificado na Figura 41.

```
<SolidColorBrush x:Key="seperateXAMLResource" Color="Yellow"/>
```

Figura 40: Exemplo de um Recurso declarado num ResourceDictionary

```
<Button Width="auto" Content="Button in a Seperated XAML Resource">
      <Button.Resources>
        <ResourceDictionary>
          <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="
/resources_4;component/Dictionary1.xaml"/>
          </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
      </Button.Resources>
      <Button.Background>
        <StaticResourceExtension ResourceKey="seperateXAMLResource"/>
      </Button.Background>
</Button>
```

Figura 41: Exemplo de utilização de um recurso contido num ResourceDictionary

Como `Button` declara o `MergedDictionaries`, todos os `Resources` do "Dictionary1" poderiam ser acedidos (caso existissem mais), bem como poderiam ser declarados mais `ResourceDictionarys`. Em relação ao *path* da `Source`, convém referir que a sua definição permite-nos também aceder a `ResourceDictionarys` noutros projectos, bastando, para isso:

- Inserir no projecto onde pretendemos usar esse `ResourceDictionary` uma referência para o projecto que o contém (por exemplo `example_project`).
- Declarar, na `Source`, esse projecto `/Example_Project`, seguido do caminho onde se encontra (supondo neste caso que se encontra numa pasta da raiz `Dictionarys`) o `ResourceDictionary` (neste exemplo `Example_Dictionary.xaml`).

É apenas necessário, neste exemplo, declarar da seguinte forma: `Source="/Example_Project;component/Dictionarys/Example_Dictionary.xaml"`.

3.1.4 *Data Binding*

3.1.4.1 *Dependency Properties*

Em *WPF* só podemos fazer *bind* a uma propriedade se ela for uma *dependency property*. *Dependency properties* são uma versão mais evoluída das propriedades *.NET*. São características destas propriedades:

- O seu valor pode ser determinado recebendo-o de um objecto `Binding`.
- Existe herança de valores entre propriedades.
- Podem ser declaradas em *XAML* tal como as propriedades normais.

3.1.4.2 *DataContext*

Os elementos *UI* no *WPF* possuem uma *dependency property*, `DataContext`. Esta propriedade permite a já referida herança de propriedades, ou seja, permite aos elementos sem valores de `DataContext` definidos herdarem os valores de `DataContext` do elemento superior mais próximo com `DataContext` definido.

3.1.4.3 A Classe *Binding*

Em *WPF*, o *data binding* só é possível graças à classe `Binding`. Alguns dos elementos desta classe são:

`Source` - Referencia uma `Binding` *data source*. Por omissão referencia o valor `DataContext` do elemento.

`Path` - É usado para indicar qual a propriedade no objecto que queremos ligar (*bind*) e por conseguinte o valor a definir.

`ElementName` - Pode ser utilizado como alternativa à propriedade `Source`. Permite especificar o nome de um elemento a utilizar como *data source*.

`Mode` - Indica a direcção de *data flow* (unidireccional, bidireccional, …).

`UpdateSourceTrigger` - Utilizado para definir quando é que os valores sofrem actualização.

3.1.4.3.1 *Data Binding* Simples entre Elementos

A Figura 42 exemplifica um cenário de *data binding* simples onde um `Button` usa o `Background` de outro `Button`. A Figura 43 ilustra o resultado para o exemplo descrito pela Figura 42.

```
<StackPanel>
    <Label Content="Simple Element Binding" Margin="5,0,0,0"
FontSize="14" />
    <StackPanel Orientation="Horizontal" Margin="10,10,10,10"
Background="Yellow">
      <Label Content="Simple Element Binding"/>
      <Button x:Name="buttonSource" Margin="10,0,0,0"
Background="Green" Content="Im buttonSource"/>
      <Button Margin="10,0,0,0" Background="{Binding
ElementName=buttonSource,
        Path=Background}" Content="Im bound to buttonSource"/>
    </StackPanel>
</StackPanel>
```

Figura 42: Exemplo de data binding simples

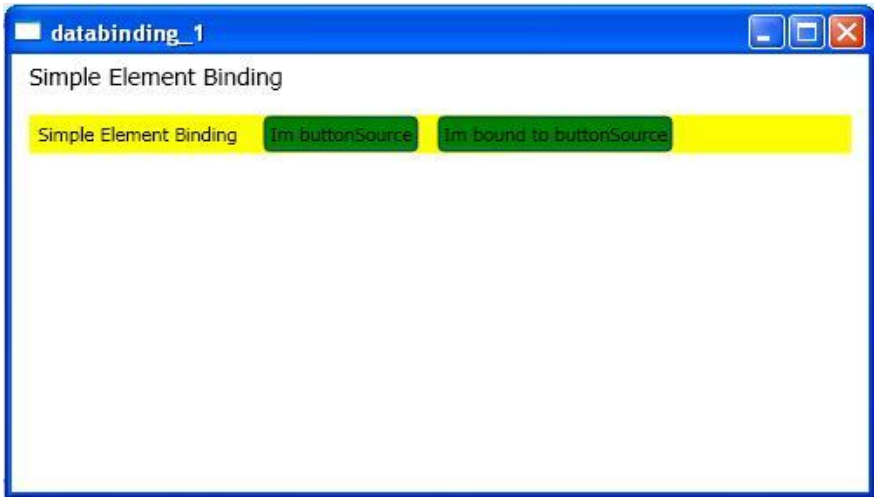

Figura 43: Resultado para o exemplo da Figura 35

3.1.4.3.2 *Data Binding* utilizando *UpdateSourceTrigger* e *Mode*

Um exemplo de *data binding*, usando `UpdateSourceTrigger` e `Mode`, é o ilustrado na Figura 44 o qual exemplifica a acção de editar uma `TextBox` e ver o resultado em tempo-real noutra `TextBox`. Este exemplo tem como resultado o descrito na Figura 45.

```
<StackPanel>
      <Label Content="Using UpdateSourceTrigger/Mode" Margin="5,0,0,0"
FontSize="14" FontWeight="Bold" />
      <StackPanel Orientation="Horizontal" Margin="10,10,10,10">
        <TextBox x:Name="textSource" Width="50" Height="25"
Margin="5,0,0,0"/>
        <TextBox Width="50" Height="25" Margin="5,0,0,0"
                 Text="{Binding ElementName=textSource,
            Path=Text, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged }"/>
      </StackPanel>
</StackPanel>
```

Figura 44: Exemplo de data binding usando `UpdateSourceTrigger` e `Mode`

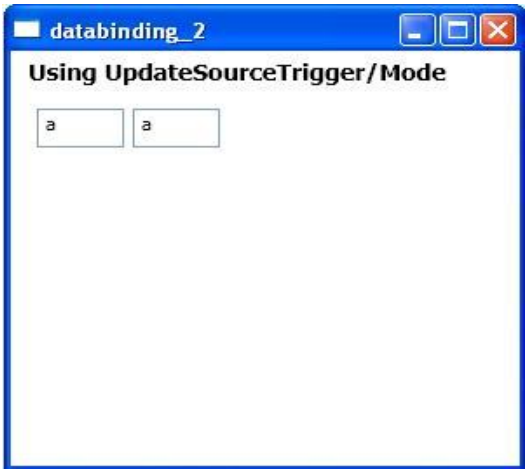

Figura 45: Resultado da execução do código exemplo da Figura 44

3.1.4.3.3 *Data Binding* utilizando *XML*

Existem duas formas de se fazer *bind* utilizando *XML*: tendo o *XML* alojado na `Window` (ou outro *host control*) ou num ficheiro *XML* à parte. Em ambos os casos temos que declarar nos `Resources` da `Window` o `XmlDataProvider`, como é visível através do exemplo da Figura 46. Traçando o cenário em que a `Source`, `XMLFile1.xml` é descrita como na Figura 47 e o código que usa o `XmlDataProvider` é especificado na Figura 48. O resultado para este cenário é ilustrado na Figura 49.

```
  <Window.Resources>
    <XmlDataProvider x:Key="xmlData" XPath="Elements">
      <x:XData>
        <Elements xmlns="">
          <Element>Element 1</Element>
          <Element>Element 2</Element>
        </Elements>
      </x:XData>
    </XmlDataProvider>

    <XmlDataProvider x:Key="xmlDataSeperateFile" XPath="Elements"
Source="XMLFile1.xml">
    </XmlDataProvider>
  </Window.Resources>
```

Figura 46: Exemplo de declarações de `XmlDataProvider`

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="">
  <Element>Element 3</Element>
  <Element>Element 4</Element>
</Elements>
```

Figura 47: "XMLFile1.xml"

```
<StackPanel>
      <Label Content="XML in the Window"></Label>
      <StackPanel Orientation="Horizontal" Margin="10,10,10,10">
        <ListBox ItemsSource="{Binding Source={StaticResource
xmlData}, XPath=Element}"/>
      </StackPanel>
      <Label Content="XML in the seperate file"></Label>
      <StackPanel Orientation="Horizontal" Margin="10,10,10,10">
        <ListBox ItemsSource="{Binding Source={StaticResource
xmlDataSeperateFile}, XPath=Element}"/>
      </StackPanel>
</StackPanel>
```

Figura 48: Exemplo de data binding usando XML

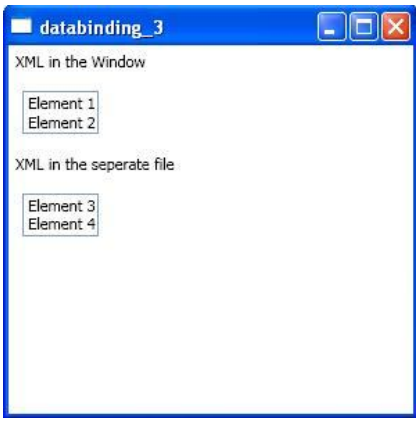

Figura 49: Resultado do exemplo de data binding utilizando XML

3.1.5 *DataTemplate* e *Triggers*

3.1.5.1 *DataTemplate*

Um `DataTemplate` é utilizado para especificar a visualização dos objectos a que se refere. Os objectos `DataTemplate` são bastante úteis quando se pretende fazer *bind* de um `ItemsControl` (tal como uma `ListBox`) a uma *collection*.

Concebendo um cenário simples, o de construir um `DataTemplate` para construir uma frase, a Figura 50 exemplifica a declaração do `DataTemplate` nos `Resources` de um *layout control*. Este `DataTemplate` referencia a "Class1" definida na Figura 51 que contêm informação sobre o nome e a idade de uma pessoa. Declara-se uma pessoa nos

`Resources` tal como no exemplo da Figura 52 e para se utilizar este `DataTemplate` basta declarar no *layout control* o código especificado na Figura 53. O resultado é o ilustrado na Figura 54.

```
<StackPanel>
    <StackPanel.Resources>
      <DataTemplate DataType="{x:Type local:Class1}">
          <Grid Margin="4">
            <TextBlock>
              <TextBlock Text="{Binding Name}"/>
              <Run Text="is"/>
              <TextBlock Text="{Binding Age}" />
              <Run Text="years old." />
            </TextBlock>
          </Grid>
</DataTemplate>
```

Figura 50: Exemplo de definição de um DataTemplate

```
public class Class1
    {
        string name;
        int age;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        public int Age
        {
            get { return age; }
            set { age = value; }
        }
    }
```

Figura 51: Exemplo de classe que é referenciada por um DataTemplate

```
<local:Class1 x:Key="person" Name="Steve" Age="50" />
```

Figura 52: Declaração de uma pessoa no exemplo sobre DataTemplate

```
<ContentControl Content="{StaticResource person}" />
```

Figura 53: Exemplo de declaração de código onde é aplicado um DataTemplate

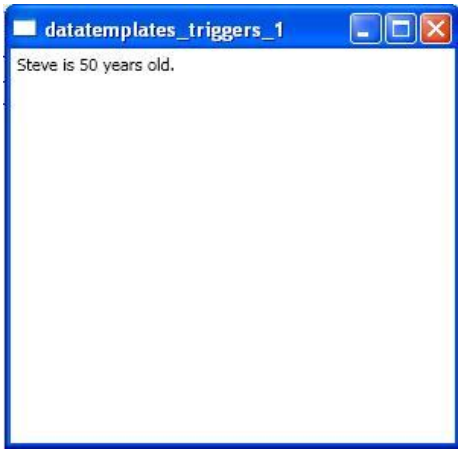

Figura 54: Resultado para o cenário exemplo sobre DataTemplate

3.1.5.2 *Triggers*

`Triggers` em *WPF* são, geralmente, associados a `DataTemplates`. Disparam um evento ou acção quando o resultado da avaliação de uma condição que lhe está associada toma um determinado valor.

No cenário da secção 3.1.5.1, ao implementar um `Trigger` que modifica o `Background` do `TextBlock` para "Green", quando o rato passa por cima do texto (*IsMouseOver*), exemplo descrito na Figura 55, o resultado é o ilustrado na Figura 56.

```
  <StackPanel>
    <StackPanel.Resources>
      <DataTemplate DataType="{x:Type local:Class1}">
          <Grid Margin="4">
            <TextBlock x:Name="text">
              <TextBlock Text="{Binding Name}"/>
              <Run Text="is"/>
              <TextBlock Text="{Binding Age}" />
              <Run Text="years old." />
            </TextBlock>
          </Grid>

        <DataTemplate.Triggers>
          <Trigger SourceName="text" Property="IsMouseOver"
Value="True">
            <Setter TargetName="text" Property="Background"
Value="Green"/>
          </Trigger>
        </DataTemplate.Triggers>
      </DataTemplate>

      <local:Class1 x:Key="person" Name="Steve" Age="50" />

    </StackPanel.Resources>
    <ContentControl Content="{StaticResource person}" />

  </StackPanel>
```

Figura 55: Exemplo de especificação de um `Trigger`

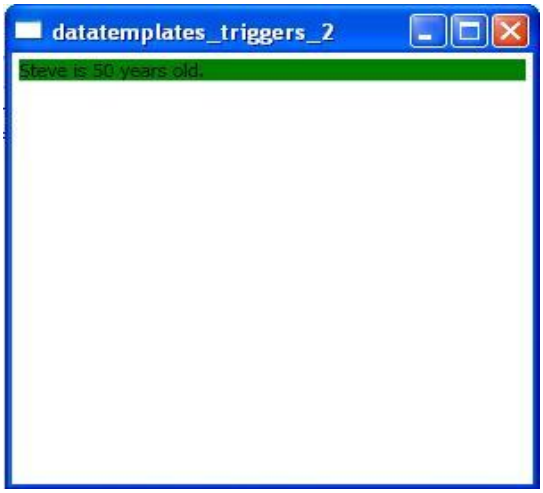

Figura 56: Resultado da execução do código exemplo da Figura 55

3.1.5.3 *Setters*

Os `Setters` definem um evento ou atribuem um valor a uma propriedade. Um exemplo de um `EventSetter` que dispara um evento poderia ser o descrito na Figura 57. Os `Setters` mais utilizados em *WPF* são os que definem o valor de uma propriedade, tal como visto no exemplo da secção 3.1.5.2, ilustrado com maior pormenor na Figura 58.

```
<Style TargetType="{x:Type Button}">
      <EventSetter Event="Click" Handler="button1SetColor"/>
</Style>
```

Figura 57: Exemplo de utilização de `EventSetter`

```
<Setter TargetName="text" Property="Background" Value="Green"/>
```

Figura 58: Exemplo de utilização de `Setter`

3.1.6 *Styles*

Aplicar Styles em *WPF* é um processo similar ao *CSS* (*Cascading Style Sheets*) no desenvolvimento *Web*. A *WPF*, como permite a separação entre o *design* e a lógica de negócio, torna-se, neste aspecto, uma ferramenta muito poderosa. A *WPF*, para além de incorporar toda uma nova gama de efeitos visuais (animações, objectos 3D, …), permite, a quem está a desenvolver o *design*, estabelecer padrões de *design* dentro da sua aplicação.

Interagindo com as outras características de *WPF* descritas neste documento, *styling* em *WPF* oferece aos *designers* uma liberdade criativa assombrosa e uma facilidade de interacção e reutilização de código impressionantes. Como já foi visto, é possível criar um `ResourceDictionary` com todos os `Styles` definidos para a aplicação, ficando este acessível a todos os ficheiros *XAML* da aplicação, aos quais esses `Styles` serão aplicados.

Todo o processo de aplicar `Styles` em *WPF* é baseado na classe `Style`. Algumas das suas propriedades são:

`Resources` - Trata-se de um `ResourceDictionary` onde podem ser armazenados objectos apenas utilizados pelo `Style`.

`Setters` - `Setters` definem um evento ou atribuem um valor a uma propriedade.

`TargetType` - Indica em qual tipo de elemento o `Style` vai ser aplicado.

`BasedOn` - Implementa a herança entre estilos, permite que um `Style` derive de outro `Style`.

`Triggers` - `Triggers` podem ser utilizados para aplicar valores a propriedades após avaliarem uma condição.

O `Style` pode ser definido num objecto, como no exemplo da Figura 59, sendo referenciado por esse objecto ou no *host control*, ficando definido para todos os `TargetTypes` desse *host control*.

Pode, também, ser definido globalmente quando especificado num `ResourceDictionary` utilizado por toda a aplicação ou nos `Resources` da `Application`, como no exemplo da Figura 60. Um exemplo de código que ilustra a utilização de `Styles`, definidos localmente e globalmente, é o descrito na Figura 61 e cujo resultado é ilustrado na Figura 62.

```
<Grid>
      <Grid.Resources>
        <Style TargetType="{x:Type TextBlock}">
          <Setter Property="Background" Value="LightGray" />
          <Setter Property="Margin" Value="2,4" />
          <Setter Property="FontSize" Value="15"/>
          <Setter Property="FontFamily" Value="Comic Sans MS" />
        </Style>
      </Grid.Resources>
      <TextBlock>Here we apply local style.</TextBlock>
</Grid>
```

Figura 59: Exemplo de `Style` definido para um objecto

```
<Application.Resources>

        <Style TargetType="{x:Type TextBlock}">
          <Setter Property="Background" Value="Green" />
          <Setter Property="Margin" Value="2,4" />
          <Setter Property="FontSize" Value="24" />
          <Setter Property="FontFamily" Value="Times New Roman" />
        </Style>

</Application.Resources>
```

Figura 60: Exemplo de `Style` definido globalmente

```
<StackPanel>
    <Grid>
      <Grid.Resources>
        <Style TargetType="{x:Type TextBlock}">
          <Setter Property="Background" Value="LightGray" />
          <Setter Property="Margin" Value="2,4" />
          <Setter Property="FontSize" Value="15"/>
          <Setter Property="FontFamily" Value="Comic Sans MS" />
        </Style>
      </Grid.Resources>
      <TextBlock>Here we apply local style.</TextBlock>
    </Grid>
    <TextBlock>Here we apply global style.</TextBlock>
</StackPanel>
```

Figura 61: Exemplo de código que ilustra a utilização de `Styles` definidos localmente e globalmente

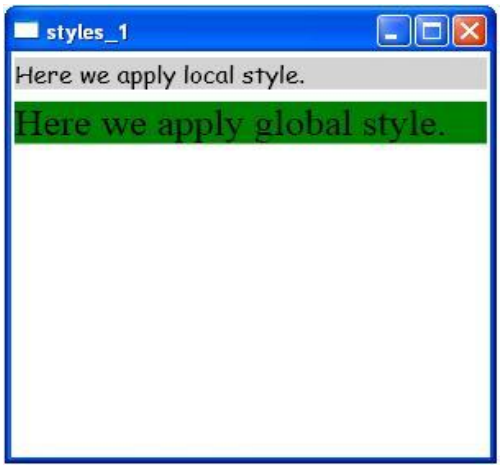

Figura 62: Resultado para o código referente à Figura 61

3.2 *Composite UI Application Block*

A *CAB* é uma *framework* que foi desenhada para suportar o desenvolvimento de aplicações *Smart Client*, facilitando a integração de vários componentes, de forma a cumprir as especificações de uma determinada aplicação. É uma extensão da *framework .NET* da *Microsoft*, funcionando como um bloco (“Block”) de código que pode ser compilado e incorporado directamente numa aplicação. Normalmente, é utilizada uma

arquitectura *Model-View-Presenter*, descrita na secção 3.2.2, para permitir que os *User Controls* (denominados *Smart Parts* na *CAB*) possam ser reutilizados. A *CAB* facilita o desenho e implementação das aplicações a três níveis:

- Permite que as aplicações sejam implementadas com base em módulos ou *plug-ins*.
- Permite aos programadores implementar componentes que separam o desenvolvimento das interfaces do desenvolvimento da lógica de negócio.
- Facilita o desenvolvimento pelo uso de padrões para uma interacção *loose coupling* entre módulos, ou seja, estes módulos são desenvolvidos sem que haja um conhecimento profundo entre as partes que recebem/enviam dados entre si.

3.2.1 Princípios

O *design* da *CAB* tem como base três áreas principais:

- Encontrar e carregar módulos na inicialização da aplicação para construir a solução dinamicamente.
- Separar o desenvolvimento de interfaces do desenvolvimento da lógica de negócio.
- Reutilizar e modularizar o código através de padrões utilizados para interacções *loose coupling.*

Este *design* é baseado nos conceitos de serviços (componentes que implementam funcionalidades) e de *containers* (componentes que possuem referências para os objectos que implementam a lógica de negócio, para *controls* e para serviços), entre os quais se encontra o *container* principal, a *shell*. Inicialmente, a *CAB* foi desenvolvida para utilizar *Windows Forms*, como ilustra a Figura 63.

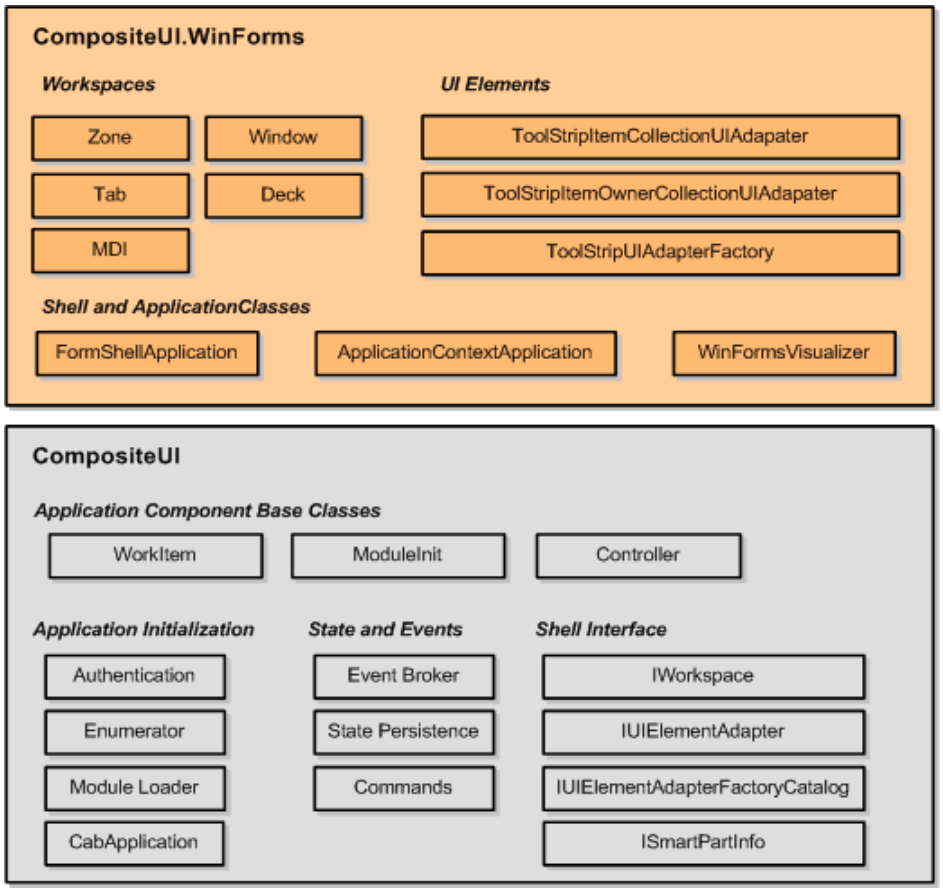

Figura 63: Subsistemas da CAB originais

3.2.1.1 Encontrar e Carregar Módulos na Inicialização da Aplicação Para Construir a Solução Dinamicamente

Uma das características principais da *CAB* é permitir o desenvolvimento de aplicações que englobam módulos independentes que colaboram entre si. Isto é conseguido através da implementação de uma lista (“*ProfileCatalog”*) que determina quais os módulos que devem ser carregados e através de um serviço, “*Module Loader*”, que carrega e inicializa todos os módulos que a aplicação engloba. As soluções a desenvolver podem ser completamente adaptáveis dependendo das necessidades de quem as desenvolve uma vez que todos os subsistemas da *CAB* funcionam como *plug-ins*.

3.2.1.2 Separar o Desenvolvimento de Interfaces do Desenvolvimento da Lógica de Negócio

Outra das características principais da *CAB* é o facto de permitir separar claramente os processos de desenvolvimento de interfaces e de desenvolvimento da lógica de negócio associada. Esta separação é conseguida de várias formas: através do uso de *Workspaces*[2], para tornar visíveis *controls* e esconder outros *controls* do utilizador, conseguindo o programador, desta forma, utilizar estes *controls* e as acções que os mesmos realizam, sem interferir com a lógica de negócio; através da padronização da adição e utilização dos elementos da *UI* na *shell*, o que permite aos programadores cuja actividade é centrada na lógica de negócio não se preocuparem em como e onde vão ser mostrados estes elementos da *UI*; por fim, através de uma arquitectura baseada em comandos (*Command*) que permite, aos programadores que se concentram na lógica de negócio, definir separadamente eventos e acções que um utilizador pode realizar e a forma como são dispostos na *shell*.

3.2.1.3 Reutilizar e Modularizar o Código através de Padrões Utilizados para Interacções *Loose Coupling*

A *CAB* também se caracteriza por uma interacção *loose coupling* entre os módulos. Esta interacção é conseguida através da utilização de *Workitems*[3], que permitem visualizar de forma eficaz quais os componentes que colaboram entre si, para definirem um caso de utilização, para partilharem estados, acções, eventos e serviços. A *CAB*, através do seu sistema "*Event Broker*", permite implementar um mecanismo de sistema de eventos entre objectos que não se encontram directamente ligados.

3.2.2 Padrão de *Software Model-View-Presenter*

Este padrão de *software* implementa a separação da parte gráfica da aplicação da parte da lógica de negócio, tornando a aplicação menos complexa e mais fácil de testar.

[2] Um *Workspace* é um *control* que é responsável por conter e tornar visíveis elementos *UI* criados pelos *WorkItems*.
[3] Um *WorkItem* é uma classe que encapsula a lógica para um único caso de uso.

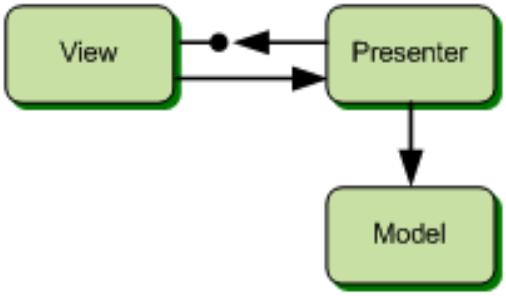

Figura 64: Vista lógica do padrão Model-View-Presenter

O *Model* assume a responsabilidade de guardar todos os dados do negócio e auxilia o *Presenter* a saber como deve responder aos eventos. Por sua vez, a *View* guarda uma referência para o *Presenter* de forma a poder delegar-lhe o processamento associado a todos os eventos despoletados pelo utilizador. O *Presenter* referencia a interface implementada pela *View*, de modo a permitir que seja utilizado por várias *Views*. Esta implementação permite a reutilização da lógica dos eventos e permite que sejam efectuados testes em *Views* que não possuem interface gráfica.

A principal limitação do *Model-View-Presenter* consiste no facto de *Model* não comunicar as suas alterações ao *Presenter*. Este facto obriga a que, sempre que o *Model* é modificado por outra entidade, o *Presenter* deva ser notificado, sendo esta notificação geralmente implementada através de eventos.

3.2.3 Padrão de *Software View Navigation*

Este padrão, *View Navigation*, implementa a comunicação entre *Views*, ou seja, permite que uma *View* actualize a sua informação em função do estado de outra *View*. Um exemplo desta utilização é quando existem duas *Views*, em que uma é uma lista de itens e a outra exibe os detalhes do item seleccionado na primeira *View*.

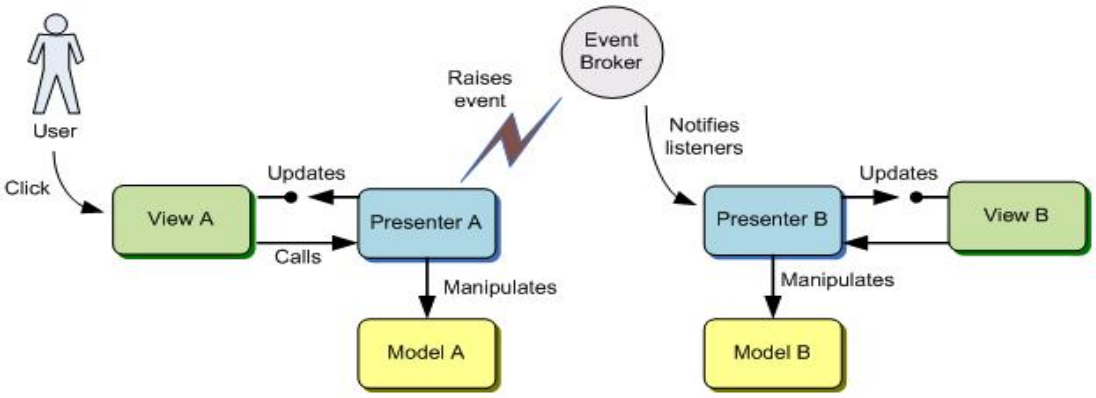

Figura 65: Vista lógica do padrão View Navigation

Este padrão de *software* preconiza que os *Presenters* associados às *Views* estabeleçam comunicação entre si através de eventos. Ilustrando através do exemplo descrito anteriormente e da vista lógica deste padrão apresentada na Figura 65, considere-se o seguinte cenário em que o utilizador seleccionou um item na "View A" e em que a "View B" deve responder a este evento, devolvendo, neste caso, os detalhes do item. A interacção para que este comportamento seja implementado segue os seguintes passos:

1. A "View A" comunica ao "Presenter A" o evento ocorrido.
2. O "Presenter A" publica o evento.
3. Todos os *Presenters* que subscrevam esse evento, entre os quais o "Presenter B", são notificados que o evento ocorreu e recebem a informação relativa a esse evento. O "Presenter B" recolhe os dados do "Model B" e comunica à "View B" qual a informação que deve devolver.

Para que o "Presenter B" possa subscrever eventos lançados pelo "Presenter A", o tópico referente ao evento tem que ser registado como global, ou as duas *Views* têm de pertencer ao mesmo *WorkItem, ou o WorkItem* a que pertence a "View A" tem de ser pai do *WorkItem* a que pertence a "View B".

3.3 *Smart Client Software Factory*

A *Smart Client Software Factory* (*SCSF*) é uma *factory* que disponibiliza maior orientação e ferramentas para a utilização da *CAB*. Implementa uma camada adicional de abstracção uma vez que automatiza o processo de desenvolvimento pela criação automática de classes e métodos usados frequentemente pelos programadores de soluções baseadas na *CAB*.

3.4 *Microsoft Expression Blend*

Microsoft Expression Blend é uma ferramenta profissional de *design* para a criação de interfaces para aplicações *Windows* que usam a tecnologia *.NET Framework 3.0* ou *3.5*. O principal intuito, da utilização desta ferramenta, prende-se com o facto de maximizar o processo de desenvolvimento de interfaces. A *.NET Framework 3.0* tem como uma das suas maiores vantagens, a separação entre o *design* e o código, permitindo separar os processos de implementação de interfaces (*designers*) e de escrita de código (programadores). Esta ferramenta ganha especial importância, uma vez que permite aos *designers* e programadores trabalhar em colaboração para produzir aplicações *Windows* de alta qualidade, oferecendo uma experiência de utilização com maior qualidade, maior produtividade e satisfação. A *XAML*, presente na tecnologia *WPF*, funciona como um formato comum entre *designers* e programadores, para trabalharem em conjunto no *Expression Studio* e no *Visual Studio*.

3.5 *Smart Client Contrib*

O projecto *Smart Client Contrib* (*SCC)* incorpora uma série de extensões para a *SCSF* e resulta do desenvolvimento de bibliotecas pela comunidade de programadores que utilizam a *SCSF*. Este projecto veio facilitar o processo de integração do *WPF* em aplicações *CAB*.

Capítulo 4

Descrição dos Principais Detalhes do Processo de Migração

Este capítulo descreve o processo de implementação das soluções para os problemas identificados no capítulo 2 (Análise do Problema). Primeiro, é descrita a estratégia de desenvolvimento adoptada, ao longo do projecto, para encontrar as respostas para os problemas e implementar as soluções pretendidas. Seguidamente, é feita a descrição pormenorizada das soluções, quer para o problema principal que este relatório aborda, a migração das soluções para *WPF*, quer para outras questões que se tornaram particularmente relevantes cuja solução poderia trazer vantagens para o projecto. Por fim, é feita uma pequena referência ao módulo, cujo desenvolvimento irá decorrer para além final do projecto.

4.1 Estratégia de Desenvolvimento

A abordagem a este projecto, devido ao seu carácter de investigação, passou sempre por complementar fases de estudo com fases de desenvolvimento e teste de pequenos protótipos que permitissem fazer uma avaliação experimental das soluções preconizadas. Na segunda parte deste projecto (desenvolvimento do módulo), que tem uma vertente inovativa muito forte, foi adoptado um modelo de desenvolvimento em espiral [Boe88], adequado a projectos de grande dimensão e complexidade. O modelo em espiral consiste num desenvolvimento iterativo e incremental. A abordagem iterativa

permite que a compreensão do problema seja feita de forma crescente, através de refinamentos sucessivos ao módulo e que sejam desenvolvidas soluções efectivas em várias iterações. Este modelo oferece maior flexibilidade na incorporação de novos requisitos ou na modificação dos requisitos previamente estabelecidos. Permite também que no desenvolvimento do módulo sejam detectados e resolvidos potenciais riscos antecipadamente.

4.2 Migração das Soluções Actuais para *Windows Presentation Foundation*

O principal objectivo do projecto foi avaliar a viabilidade da migração das soluções actuais, cujas interfaces estão implementadas em *Windows Forms*, para a tecnologia *WPF*, os custos desta migração, as suas vantagens e desvantagens e qual a melhor forma de ser realizada, ou seja, se a migração pode ser realizada passo a passo, ou se deve ser total. As seguintes subsecções detalham os pontos focados no capítulo 2, Análise do Problema. São descritos: o estudo da interoperabilidade entre *WPF* e *Windows Forms*, o processo de migração e o estudo acerca da utilização das bibliotecas *SCSFContrib* do projecto SCC. São, ainda, realizadas: uma análise sobre a utilização da *Microsoft Expression Blend*, uma descrição sobre o processo de "localização" em *WPF* e, por fim, uma breve descrição do módulo.

4.2.1 Estudo da Interoperabilidade entre *Windows Presentation Foundation* e *Windows Forms*

A biblioteca "CompositeUI.WPF" surgiu como uma extensão à *CAB* e veio permitir a utilização da *WPF* em aplicações desenvolvidas na *CAB*. A Figura 66 ilustra a arquitectura desta biblioteca.

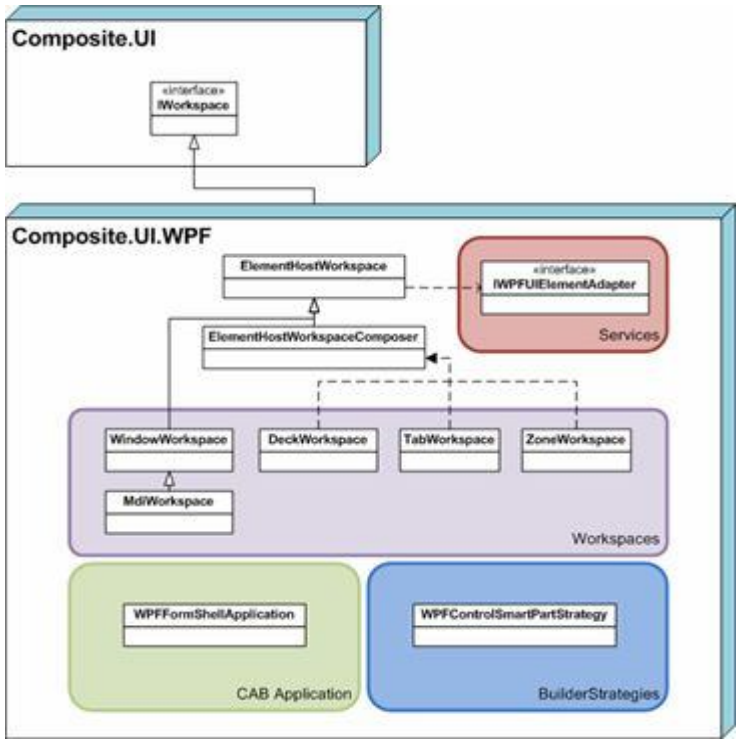

Figura 66: Arquitectura da "Composite.UI.WPF"

Esta biblioteca disponibiliza os mesmos *Workspaces* para *WPF* que já existiam originalmente na *CAB* para *Windows Forms*. Estes novos *Workspaces* permitem incluir *Smart Parts* em *WPF*, apesar de internamente funcionarem com *controls* em *Windows Forms*. Isto é conseguido através de alterações feitas à classe "Workspace", que na "CompositeUI.WPF" se passou a chamar "ElementHostWorkspace". Esta mudança de nome deve-se ao facto do "ElementHostWorkspace", quando uma Smart Part vai ser tornada visível, encapsular o *control WPF* num *control Windows Forms* utilizado para alojar elementos *WPF* em *Windows Forms*, o *ElementHost* [Cora]. O "ElementHostWorkspace" delega no serviço "WPFUIElementAdapter" a responsabilidade de encapsular e desencapsular os *controls WPF*. Este serviço é registado pela classe "WPFFormShellApplication" no *WorkItem* principal ("RootWorkItem") e é injectado automaticamente nos *Workspaces WPF*.

Uma vez que "ElementHostWorkspace" resulta de alterações feitas à classe "Workspace", é possível alojar *Smart Parts* em *Windows Forms* nos *Workspaces WPF*; o contrário, alojar *Smart Parts* em *WPF* nos *Workspaces Windows Forms*, já não é possível. Esta informação é ilustrada na Tabela 1.

	Workspaces Windows Forms	*Workspaces WPF*
Smart Parts Windows Forms		
Smart Parts WPF		

Tabela 1: Interoperabilidade entre Workspaces e Smart Parts em WPF e Windows Forms

4.2.2 Descrição do Processo de Migração

Após o estudo do funcionamento da *CAB* e da interoperabilidade entre *WPF* e *Windows Forms* apenas se colocava a questão da integração com a *shell* actual, que deriva da classe `IGFormShellApplication`, pertencente à biblioteca "Infragistics.Practices.CompositeUI" da *Infragistics*, para, assim, se poderem utilizar os componentes da *Infragistics* também nas soluções migradas. Para atingir este objectivo, foi necessário alterar o código fonte da biblioteca "Infragistics.Practices.CompositeUI", de modo a que a migração fosse realizada sem perder quaisquer funcionalidades e manter a estrutura associada à *CAB*.

Para permitir que a biblioteca "Infragistics.Practices.CompositeUI" alojasse componentes *WPF*, foi necessário adicionar às estratégias de compilação da *CAB*, isto é, às estratégias que gerem os processos que afectam os objectos durante a sua inicialização e remoção da aplicação, uma nova estratégia que permitisse a utilização destes componentes. Foram então adicionadas as estratégias utilizadas pela "Composite.UI.WPF". A Figura 67 descreve as declarações efectuadas.

```
using Microsoft.Practices.CompositeUI.WPF.BuilderStrategies;
using Microsoft.Practices.CompositeUI.WPF;
```

Figura 67: Declaração das principais bibliotecas necessárias à migração

A estratégia utilizada, "WPFControlSmartPartStrategy", percorre a cadeia de *controls WPF* contidos na aplicação e procura *controls* filhos que sejam *Smart Parts* ou *Workspaces* e adiciona-os ao *WorkItem* que contém os *controls WPF* pais, ou seja, sempre que um *control WPF* é adicionado a um *WorkItem*, esta estratégia vai procurar *SmartParts* e *IWorkspaces*, adicionando-os ao *WorkItem*, se necessário. A Figura 68 apresenta o código que adiciona a estratégia no arranque da aplicação.

```
protected override void AddBuilderStrategies(Builder builder)
        {
            base.AddBuilderStrategies(builder);

            // add strategies...

            // add a strategy that will register our workspaces, etc.

      builder.Strategies.AddNew<IGWorkItemStrategy>(BuilderStage.Initi
alization);


builder.Strategies.AddNew<WPFControlSmartPartStrategy>(BuilderStage.In
itialization);

      // if ink should be supported by the application then we
      // should add the builder that will create the ink providers
      // for the application.
                  if (this.ProvideInkSupportResolved)
                  {

      builder.Strategies.AddNew<InkProviderStrategy>(BuilderStage.Init
ialization);

                  }


        }
```

Figura 68: Adição das estratégias de compilação

Por fim, foi necessário adicionar o serviço "WPFUIElementAdapter", que encapsula e desencapsula os *controls WPF* em *controls Windows Forms*. Este serviço contém um dicionário, <UIElement, ElementHost>, que é consultado pelo serviço quando uma *Smart Part WPF* necessita de ser encapsulada num *control Windows Forms*:

- Se a *Smart Part* ainda não tiver sido encapsulada, é criado um novo objecto *ElementHost*, que encapsula a *Smart Part* e adiciona-a ao dicionário.

- Se a *Smart Part* já tiver sido encapsulada, a mesma instância do *ElementHost* criado anteriormente é retornada.

Isto permite aos *Workspaces* tratar todas as *Smart Parts* como *Smart Parts Windows Forms*. Por outro lado, a utilização deste dicionário não impede que os elementos da *UI* sejam removidos pelo gestor de memória. O método ilustrado na Figura 69 descreve a adição deste serviço.

```
protected override void AddServices()
        {
            base.AddServices();
            RootWorkItem.Services.AddNew<WPFUIElementAdapter,
IWPFUIElementAdapter>();

        }
```

Figura 69: Adição do serviço "WPFUIElementAdapter"

Ilustrando através de um exemplo, considere-se o seguinte cenário, onde é utilizada a *SCSF* para criar uma aplicação *Smart Client* (Figura 70) e onde se permite que esta utilize *Smart Parts WPF*, como é visível na Figura 71. A arquitectura resultante é ilustrada na Figura 72.

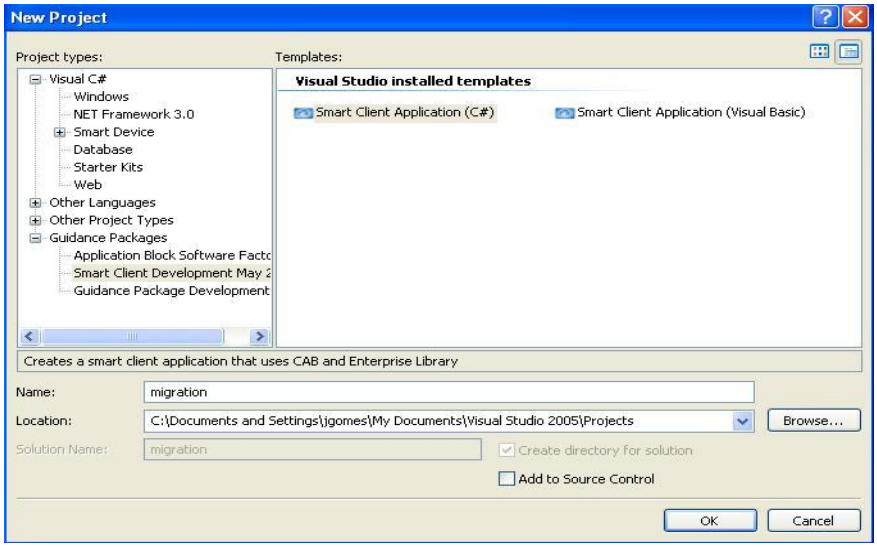

Figura 70: Criação de uma aplicação Smart Client através da SCSF

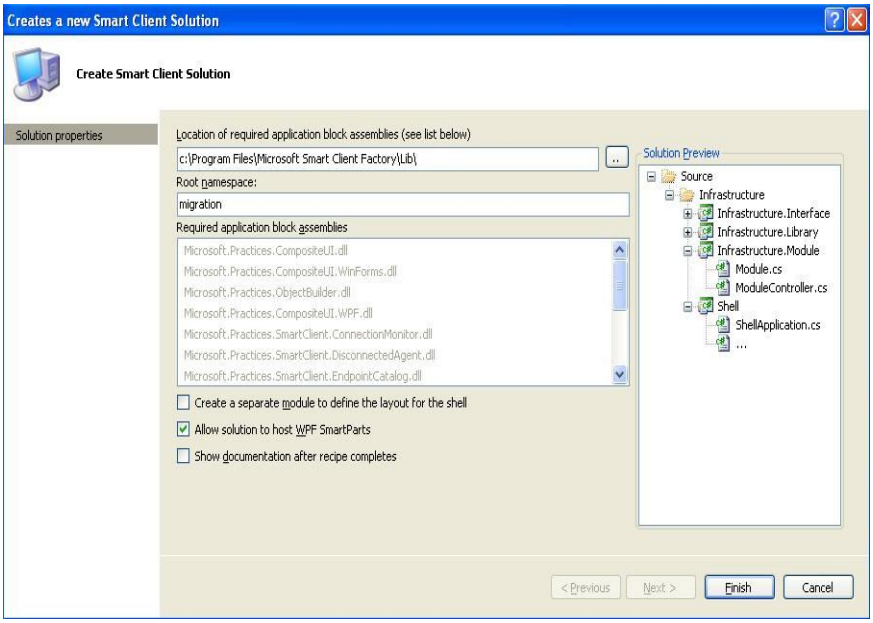

Figura 71: Permissão que a aplicação utilize Smart Parts WPF

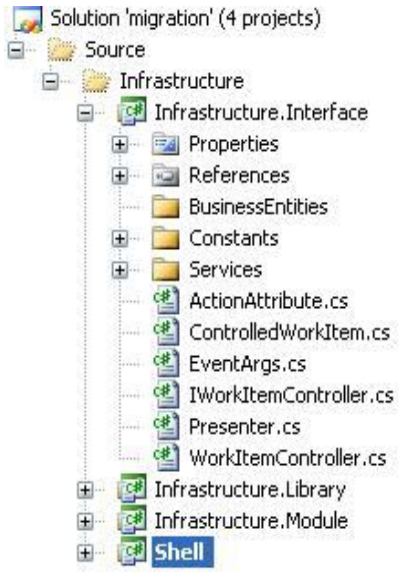

Figura 72: Arquitectura resultante da criação de uma aplicação Smart Client na SCSF

Para a aplicação suportar simultaneamente componentes da *Infragistics* e da *WPF*, é necessário, apenas, substituir, nos projectos que as referenciam, as bibliotecas "Microsoft.Practices.CompositeUI", "Microsoft.Practices.CompositeUI.Winforms" e "Microsoft.Practices.CompositeUI.WPF" pelas suas bibliotecas alteradas equivalentes e adicionar nos projectos "Infrastructure.Library" e "Shell" uma referência para a biblioteca "Infragistics.Practices.CompositeUI.WinForms".

Por fim, na classe "SmartClientApplication" do projecto "Infrastructure.Library" basta alterar o código especificado pela Figura 73, pelo código especificado pela Figura 74, para que a classe "IGFormApplication" seja herdada pela classe "SmartClientApplication"

```
public abstract class SmartClientApplication<TWorkItem, TShell> :
WPFFormShellApplication<TWorkItem, TShell>
```

Figura 73: Definição da classe "SmartClientApplication" que não suporta a utilização de WPF

```
public abstract class SmartClientApplication<TWorkItem, TShell> :
IGFormShellApplication<TWorkItem, TShell>
```

Figura 74: Definição da classe "SmartClientApplication" que suporta a utilização de WPF

Desta forma simples, a *shell* do exemplo já suporta *WPF*, *Windows Forms* e os componentes da *Infragistics*.

4.2.3 Utilização das Bibliotecas *SCSFContrib*

O maior ponto de interesse, para a utilização das bibliotecas *SCSFContrib* pertencentes ao projecto *SCC*, é o facto de permitirem uma integração da *WPF* com a *CAB* muito mais fácil e eficaz.

A biblioteca "SCSFContrib.CompositeUI.WPF" toma particular destaque uma vez que nos permite incluir *Views WPF* dentro de outras *Views WPF* sem se desviar das directrizes da *CAB* uma vez que permite a criação de *Workspaces*.

Ilustrando através de um exemplo, considere-se o seguinte cenário em que se pretende incluir numa *View WPF*, "ExampleWPFView", um `TabWorkspace` que contém dois *tabs*, cada um com uma *View WPF*, "NewWPFView" e "NewWPFView2" respectivamente, como é descrito na Figura 75. É necessário adicionar as bibliotecas às referências do projecto e declarar a sua utilização (neste caso da

"SCSFContrib.CompositeUI.WPF"), através de um atributo *xmlns* tal como exemplificado na Figura 76. Especifica-se na "ExampleWPFView" que a primeira *tab*, do `TabWorkspace`, contém a "NewWPFView " e a segunda a "NewWPFView2" tal como definido no código da Figura 77. O código da "NewWPFView" é definido na Figura 78 e o código da "NewWPFView2" é especificado na Figura 80. O resultado para este cenário é visível na Figura 79.

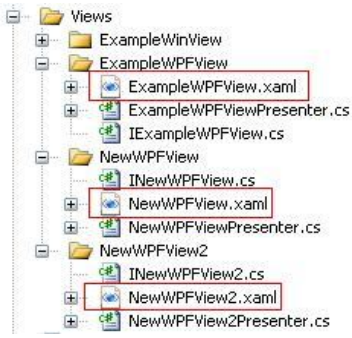

Figura 75: Cenário exemplo de utilização das bibliotecas SCSFContrib

```
xmlns:workspaces="clr-
namespace:SCSFContrib.CompositeUI.WPF.Workspaces;assembly=SCSFContrib.
CompositeUI.WPF"
```

Figura 76: Declaração do atributo xmlns para utilização da biblioteca "SCSFContrib.CompositeUI.WPF"

```
<UserControl x:Class="InfraProto.ExampleModule.ExampleWPFView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:workspaces="clr-
namespace:SCSFContrib.CompositeUI.WPF.Workspaces;assembly=SCSFContrib.
CompositeUI.WPF"
    xmlns:views="clr-namespace:InfraProto.ExampleModule">
  <Grid>
    <workspaces:TabWorkspace x:Name="tabWork">
    <TabItem Header="WPFForm">
      <TabItem.Content>
        <views:NewWPFView/>
      </TabItem.Content>
    </TabItem>
      <TabItem Header="WPFForm2">
        <TabItem.Content>
          <views:NewWPFView2/>
        </TabItem.Content>
      </TabItem>
    </workspaces:TabWorkspace>
  </Grid>
</UserControl>
```

Figura 77: Especificação da "ExampleWPFView" que utiliza o `TabWorkspace`

```
<UserControl x:Class="InfraProto.ExampleModule.NewWPFView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid>
      <Label Content="I am WPF" Background="Green"/>
      <Ellipse Width="200" Height="150">
        <Ellipse.Fill>
          <LinearGradientBrush>
            <GradientStop Offset="0" Color="Teal"/>
            <GradientStop Offset="1" Color="Aqua"/>
          </LinearGradientBrush>
        </Ellipse.Fill>
      </Ellipse>
    </Grid>
</UserControl>
```

Figura 78: Especificação da "NewWPFView"

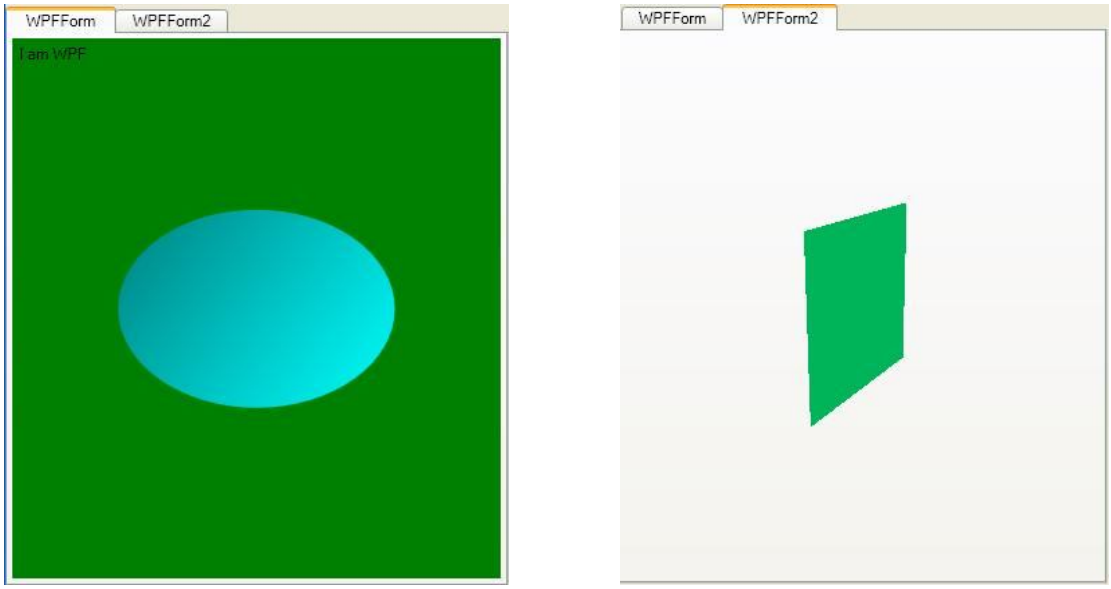

Figura 79: Resultado para o cenário de utilização das bibliotecas SCSFContrib: (a) Primeira tab seleccionada; (b) Segunda tab seleccionada;

```
<UserControl x:Class="InfraProto.ExampleModule.NewWPFView2"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid>

      <Viewport3D>
        <Viewport3D.Camera>
          <PerspectiveCamera FarPlaneDistance="20"
          LookDirection="5,-2,-3" UpDirection="0,1,0"
          NearPlaneDistance="1" Position="-5,2,3" FieldOfView="45" />
        </Viewport3D.Camera>
        <ModelVisual3D>
          <ModelVisual3D.Content>
            <Model3DGroup>
              <DirectionalLight Color="White" Direction="-3,-4,-5" />
              <GeometryModel3D>
                <GeometryModel3D.Geometry>
                  <MeshGeometry3D
                    Positions="-1 -1 0  1 -1 0  -1 1 0  1 1 0"
                    Normals="0 0 1  0 0 1  0 0 1  0 0 1"
                    TextureCoordinates="0 1  1 1  0 0  1 0"
                    TriangleIndices="0 1 2  1 3 2" />
                </GeometryModel3D.Geometry>
                <GeometryModel3D.Material>
                  <DiffuseMaterial>
                    <DiffuseMaterial.Brush>
                      <SolidColorBrush Color="SpringGreen"/>
                    </DiffuseMaterial.Brush>
                  </DiffuseMaterial>
                </GeometryModel3D.Material>
              </GeometryModel3D>
            </Model3DGroup>
          </ModelVisual3D.Content>
        </ModelVisual3D>
      </Viewport3D>

    </Grid>
</UserControl>
```

Figura 80: Especificação da "NewWPFView2"

4.2.4 Utilização da *Microsoft Expression Blend*

A ferramenta de *design Microsoft Expression Blend* é a utilizada para criar interfaces *WPF*. Um dos objectivos desta ferramenta é facilitar a comunicação entre *designers* e programadores no desenvolvimento de aplicações com interfaces *WPF*. Esta gera, automaticamente, a *XAML*, permitindo que o *designer* apenas se concentre no *desgin*, em si, da aplicação. Caso seja necessário, permite que o *design* seja, também, alterado ao nível da programação em *XAML*, o que concede um maior grau de liberdade a quem o está a implementar.

Os programadores e os *designers* podem trabalhar colaborativamente na mesma aplicação, uma vez que, pode ser partilhada simultaneamente pela *Microsoft Expression Blend* e pelo *Visual Studio*. A Figura 81 ilustra este cenário. Os *designers* podem desenvolver todo o *layout* da aplicação sem escrever código e os programadores podem interagir com os *controls* sem necessitarem de alterar a estrutura do *layout*.

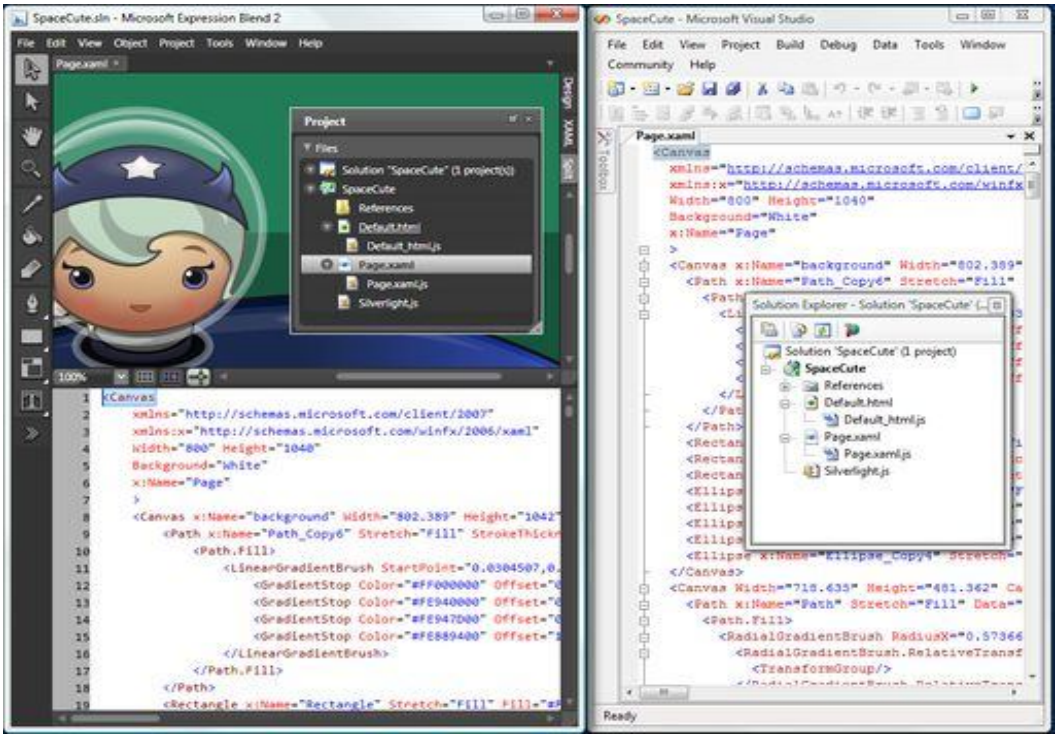

Figura 81: Interacção entre Visual Studio e Microsoft Expression Blend

Durante o projecto, surgiu a necessidade de investigar a utilização de *skins* (temas) em *WPF* para, por exemplo, alterar a *skin* da aplicação conforme a instituição de saúde a que a aplicação se destinasse. Esta necessidade veio proporcionar um caso de teste à utilidade e funcionalidades da *Microsoft Expression Blend* e à interacção com o *Visual Studio*. Foi concebido, então, um protótipo que permitisse testar a utilização de *skins* recorrendo à *Microsoft Expression Blend*.

O protótipo consistiu numa pequena aplicação que era constituída por um cabeçalho e três botões, em que o cabeçalho mudaria de *skin* em função do botão pressionado. O *control*, especificado na Figura 82, foi desenvolvido na *Microsoft Expression Blend* e dispõe o cabeçalho e os três botões horizontalmente. A cada botão é associado um estilo estático e ao cabeçalho é associado um estilo dinâmico, permitindo assim que o estilo deste seja alterado conforme o botão pressionado. A cada botão é associado um evento, que chama a função "ChangeSkin" do *code-behind* e um estilo distinto, que irá também ser aplicado ao cabeçalho, aquando da acção de pressionar um destes botões.

Os dicionários, onde se encontram os estilos associados a cada botão, são os definidos nas Figuras 83, 84 e 85. Todos estes dicionários referenciam o dicionário da Figura 86 que contém os estilos criados na *Microsoft Expression Blend*.

A Figura 87 especifica a classe em *C#* associada ao *control XAML*. A função "ChangeSkin" recebe a *tag* associada a cada botão. Esta *tag* indica o *path*, onde se encontra o dicionário, em que está definido o estilo de cada botão e que posteriormente será aplicado ao cabeçalho. A função "ApplySkin" é invocada pela função "ChangeSkin" e remove o dicionário anteriormente definido para o cabeçalho, carregando para a aplicação o dicionário que se encontra no *path*, recebido da *tag* na função "ChangeSkin". Por omissão, o dicionário utilizado é o referente ao primeiro botão.

```
<UserControl x:Class="Video.ExampleModule.ExampleWPFView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:cab="clr-
namespace:SCSFContrib.CompositeUI.WPF.Workspaces;assembly=SCSFContrib.
CompositeUI.WPF"
    xmlns:views="clr-namespace:Video.ExampleModule;assembly="
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d">


  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="0.161*"/>
      <RowDefinition Height="0.839*"/>
    </Grid.RowDefinitions>
    <Rectangle Margin="0.654,0.517,-0.379,-0.622"
Style="{DynamicResource headerStyle}"/>

    <Button d:LayoutOverrides="Width, Height"
Margin="112.581,89.14,139.336,0" VerticalAlignment="Top"
Content="Blue" Grid.Row="1" Background="{StaticResource myBlue}"
Foreground="#FFFBFBFB" FontFamily="Comic Sans MS" FontSize="9"
Click="ChangeSkin"
Tag="/ExampleModule;component\Views\Skins\BlueSkin.xaml"/>
    <Button d:LayoutOverrides="Width, Height"
Margin="112.581,55.589,139.336,0" VerticalAlignment="Top"
Content="Red" Grid.Row="1" Background="{StaticResource myRed}"
Foreground="#FFFBFBFB" FontFamily="Comic Sans MS" FontSize="9"
Click="ChangeSkin"
Tag="/ExampleModule;component\Views\Skins\RedSkin.xaml"/>
    <Button d:LayoutOverrides="Width, Height"
Margin="112.581,22.784,139.335,0" VerticalAlignment="Top"
Content="Default" Grid.Row="1" Background="{StaticResource myGray}"
Foreground="#FFFBFBFB" FontFamily="Comic Sans MS" FontSize="9"
Click="ChangeSkin"
Tag="/ExampleModule;component\Views\Skins\DefaultSkin.xaml"/>
    <Label HorizontalAlignment="Left" VerticalAlignment="Top"
Content="Video Tutorial" Width="Auto" Height="Auto"
Background="#00FFFFFF" Foreground="#FFFFFFFF" FontFamily="Comic Sans
MS" FontSize="14"/>

  </Grid>

</UserControl>
```

Figura 82: Control do protótipo sobre Skins que dispõe o cabeçalho e os botões

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary
Source="/LibraryApplication;component/ResourceDictionary1.xaml"/>
  </ResourceDictionary.MergedDictionaries>

  <Style x:Key="headerStyle" TargetType="{x:Type Rectangle}">
    <Setter Property="Fill" Value="{DynamicResource myGray}"/>
  </Style>


</ResourceDictionary>
```

Figura 83: Dicionário que define o estilo para o primeiro botão do protótipo sobre Skins em WPF

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary
Source="/LibraryApplication;component/ResourceDictionary1.xaml"/>
  </ResourceDictionary.MergedDictionaries>

  <Style x:Key="headerStyle" TargetType="{x:Type Rectangle}">
    <Setter Property="Fill" Value="{DynamicResource myRed}"/>
  </Style>


</ResourceDictionary>
```

Figura 84: Dicionário que define o estilo para o segundo botão do protótipo sobre Skins em WPF

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary
Source="/LibraryApplication;component/ResourceDictionary1.xaml"/>
  </ResourceDictionary.MergedDictionaries>

  <Style x:Key="headerStyle" TargetType="{x:Type Rectangle}">
    <Setter Property="Fill" Value="{DynamicResource myBlue}"/>
  </Style>


</ResourceDictionary>
```

Figura 85: Dicionário que define o estilo para o terceiro botão do protótipo sobre Skins em WPF

```
<ResourceDictionary
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentatio"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
      <LinearGradientBrush x:Key="myGray" EndPoint="0.5,1"
StartPoint="0.5,0">
            <GradientStop Color="#FF000000" Offset="0"/>
            <GradientStop Color="#FFFFFFFF" Offset="1"/>
      </LinearGradientBrush>

      <LinearGradientBrush x:Key="myRed" EndPoint="0.5,1"
StartPoint="0.5,0">
            <GradientStop Color="#FF000000" Offset="0"/>
            <GradientStop Color="#FFC22020" Offset="0.335"/>
      </LinearGradientBrush>

      <LinearGradientBrush x:Key="myBlue" EndPoint="0.5,1"
StartPoint="0.5,0">
            <GradientStop Color="#FF000000" Offset="0"/>
            <GradientStop Color="#FF3C67B3" Offset="0.554"/>
      </LinearGradientBrush>

</ResourceDictionary>
```

Figura 86: Dicionário que contém os estilos criados na Microsoft Expression Blend

```
public partial class ExampleWPFView :
System.Windows.Controls.UserControl, IExampleWPFView, IDisposable
    {
        public ExampleWPFView()
        {
            string skinDictPath =
"/ExampleModule;component\\Views\\Skins\\DefaultSkin.xaml";
            Uri skinDictUri = new Uri(skinDictPath, UriKind.Relative);
            this.ApplySkin(skinDictUri);
            InitializeComponent();
            this.Loaded += new RoutedEventHandler(OnLoad);
        }

        public void OnLoad(object sender, RoutedEventArgs e)
        {
            _presenter.OnViewReady();
        }

        public void ApplySkin(Uri skinDictionaryUri)
        {
            ResourceDictionary skinDict =
Application.LoadComponent(skinDictionaryUri) as ResourceDictionary;
            string resDictPath =
"/LibraryApplication;component\\ResourceDictionary1.xaml";
            Uri resDictUri = new Uri(resDictPath, UriKind.Relative);
            ResourceDictionary resDict =
Application.LoadComponent(resDictUri) as ResourceDictionary;

            Collection<ResourceDictionary> mergedDicts =
base.Resources.MergedDictionaries;

            if (mergedDicts.Count > 0) mergedDicts.Clear();

            mergedDicts.Add(skinDict);
            mergedDicts.Add(resDict);
        }

        public void ChangeSkin(object sender, RoutedEventArgs e)
        {
            Button b = e.OriginalSource as Button;
            string skinDictPath = b.Tag as string;
            Uri skinDictUri = new Uri(skinDictPath, UriKind.Relative);
            this.ApplySkin(skinDictUri);
        }
    }
```

Figura 87: Classe em C# *associada ao* control *XAML do protótipo sobre Skins em WPF*

Ao executar a aplicação, o estilo do cabeçalho é, inicialmente, o que está associado ao primeiro botão (Figura 88). As Figuras 89 e 90 ilustram o resultado da acção de pressionar o primeiro e o segundo botão.

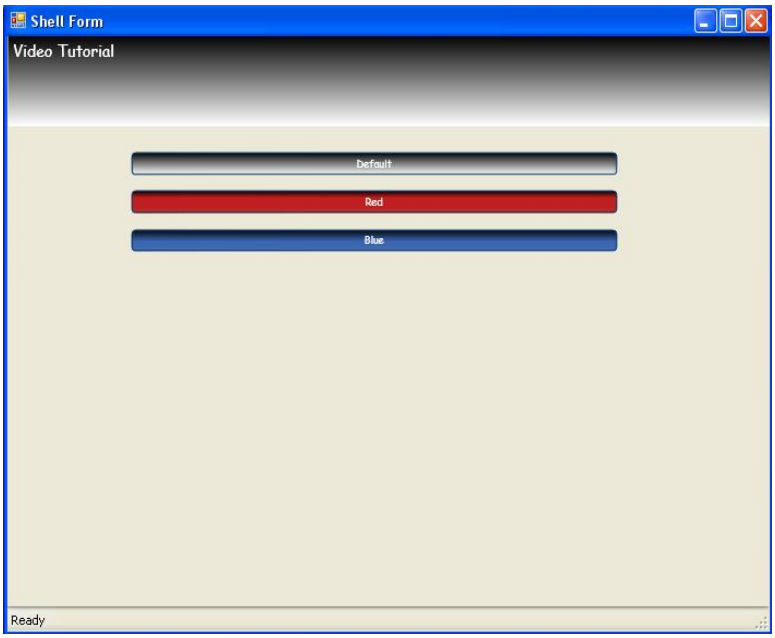

Figura 88: Estilo do cabeçalho ao iniciar ou quando o primeiro botão é pressionado

Figura 89: Estilo do cabeçalho quando o segundo botão é pressionado

Figura 90: Estilo do cabeçalho quando o terceiro botão é pressionado

4.2.5 "Localização" em *Windows Presentation Foundation*

Localizar a aplicação significa traduzir os recursos da aplicação (mensagens para o utilizador, títulos, entre outros) conforme o idioma pretendido. Este é um ponto bastante importante no desenvolvimento de soluções informáticas, uma vez que aumenta o número de potenciais utilizadores do *software* desenvolvido, pois, ao tornar as aplicações acessíveis a um maior número de utilizadores, o mercado alvo das aplicações aumenta substancialmente.

Este aspecto é fundamental quando se pretende alargar as aplicações desenvolvidas ao mercado internacional ou se quer tornar o *software* acessível a pessoas que apenas entendem o idioma nativo. Um exemplo muito comum, na área das soluções informáticas concebidas para a área da saúde, é, por exemplo, o de um médico estrangeiro obter uma experiência de utilização mais enriquecedora e produtiva pelo facto de a interacção com a aplicação ser feita no seu idioma nativo.

ResXFileCodeGenerator é uma ferramenta integrada no *Visual Studio*, essencial para a "localização", que é automaticamente associada aos ficheiros *.resx quando são adicionados à aplicação. Sempre que a aplicação é compilada, esta ferramenta gera uma

classe que corre sobre uma máquina virtual e que disponibiliza todos os recursos dos ficheiros *.resx (*strings*, imagens, entre outros), sobre a forma de propriedades do tipo *static*, permitindo um acesso fácil aos mesmos, por parte da aplicação. Uma das limitações desta ferramenta é que a classe gerada é do tipo *internal*, o que implica que outras *Dll*s que queiram aceder a esta classe não o consigam. Esta limitação faz com que a *Dll* utilizada pelo *WPF*, a *PresentationFramework*, não consiga aceder à classe gerada pela *ResXFileCodeGenerator*, impossibilitando assim o processo de "localização" através do uso de ficheiros *.resx.

Este problema foi detectado e resolvido através do desenvolvimento de uma nova ferramenta, a *ResXFileCodeGeneratorEX* [Kry], que funciona como uma extensão à *ResXFileCodeGenerator* e que tem como principal resultado tornar pública a classe que é gerada. Para utilizar a *ResXFileCodeGeneratorEX* basta alterarmos, nas propriedades dos ficheiros *.resx a ferramenta ("Custom Tool") que estamos a associar ao ficheiro *.resx como demonstra a Figura 91.

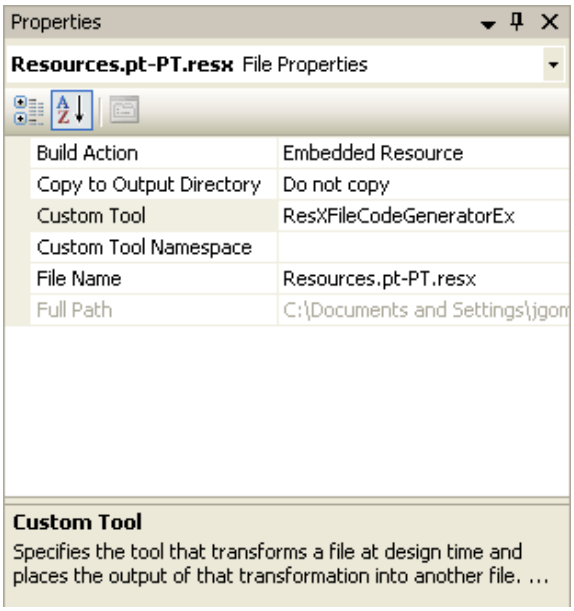

Figura 91: Utilização da ferramenta ResXFileCodeGeneratorEx

Continuando com o cenário da secção 4.2.4 e recorrendo a um exemplo, considere-se a situação em que a aplicação deveria indicar no canto superior direito o

país referente à *CultureInfo* [Corb] do sistema. Para se obter este resultado deve ser acrescentado o código especificado na Figura 92 ao código anteriormente exemplificado na Figura 82 da página 62. Deve ser acrescentada, também, a declaração do atributo *xmlns*, que permite aceder à pasta *Properties* do projecto, onde se encontram os recursos a utilizar, tal como no exemplo da Figura 93. O código resultante destas duas alterações é descrito na Figura 94.

A Figura 95 especifica ficheiros *.resx que vão retornar em "HeaderText"o país referente à *CultureInfo* do sistema ("England" por omissão e "Portugal" para a *CultureInfo "pt-PT"*), a Figura 96 ilustra o resultado para a *CultureInfo* por omissão e a Figura 97 ilustra o resultado para a *CultureInfo "pt-PT"*.

```
    <Label HorizontalAlignment="Right" Width="Auto" Height="Auto"
Content="{x:Static properties:Resources.HeaderText}"
Foreground="White"/>
```

Figura 92: Label que vai ser acrescentada ao exemplo da secção 4.2.5 para mostrar o país referente à CultureInfo do sistema

```
xmlns:properties="clr-
namespace:Video.ExampleModule.Properties;assembly="
```

Figura 93: Declaração do atributo xmlns para o exemplo da secção 4.2.5

```
<UserControl x:Class="Video.ExampleModule.ExampleWPFView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:cab="clr-
namespace:SCSFContrib.CompositeUI.WPF.Workspaces;assembly=SCSFContrib.
CompositeUI.WPF"
    xmlns:views="clr-namespace:Video.ExampleModule;assembly="
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
    xmlns:properties="clr-
namespace:Video.ExampleModule.Properties;assembly=">


  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="0.161*"/>
      <RowDefinition Height="0.839*"/>
    </Grid.RowDefinitions>
    <Rectangle Margin="0.654,0.517,-0.379,-0.622"
Style="{DynamicResource headerStyle}"/>

    <Button d:LayoutOverrides="Width, Height"
Margin="112.581,89.14,139.336,0" VerticalAlignment="Top"
Content="Blue" Grid.Row="1" Background="{StaticResource myBlue}"
Foreground="#FFFBFBFB" FontFamily="Comic Sans MS" FontSize="9"
Click="ChangeSkin"
Tag="/ExampleModule;component\Views\Skins\BlueSkin.xaml"/>
    <Button d:LayoutOverrides="Width, Height"
Margin="112.581,55.589,139.336,0" VerticalAlignment="Top"
Content="Red" Grid.Row="1" Background="{StaticResource myRed}"
Foreground="#FFFBFBFB" FontFamily="Comic Sans MS" FontSize="9"
Click="ChangeSkin"
Tag="/ExampleModule;component\Views\Skins\RedSkin.xaml"/>
    <Button d:LayoutOverrides="Width, Height"
Margin="112.581,22.784,139.335,0" VerticalAlignment="Top"
Content="Default" Grid.Row="1" Background="{StaticResource myGray}"
Foreground="#FFFBFBFB" FontFamily="Comic Sans MS" FontSize="9"
Click="ChangeSkin"
Tag="/ExampleModule;component\Views\Skins\DefaultSkin.xaml"/>
    <Label HorizontalAlignment="Left" VerticalAlignment="Top"
Content="Video Tutorial" Width="Auto" Height="Auto"
Background="#00FFFFFF" Foreground="#FFFFFFFF" FontFamily="Comic Sans
MS" FontSize="14"/>
    <Label HorizontalAlignment="Right" Width="Auto" Height="Auto"
Content="{x:Static properties:Resources.HeaderText}"
Foreground="White"/>

  </Grid>
</UserControl>
```

Figura 94: Código resultante da alteração ao código da Figura 82

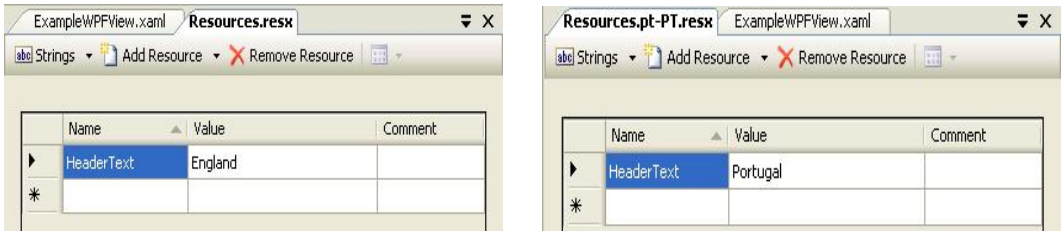

*Figura 95: Ficheiros *.resx: (a) Resources.resx; (b) Resources.pt-PT.resx*

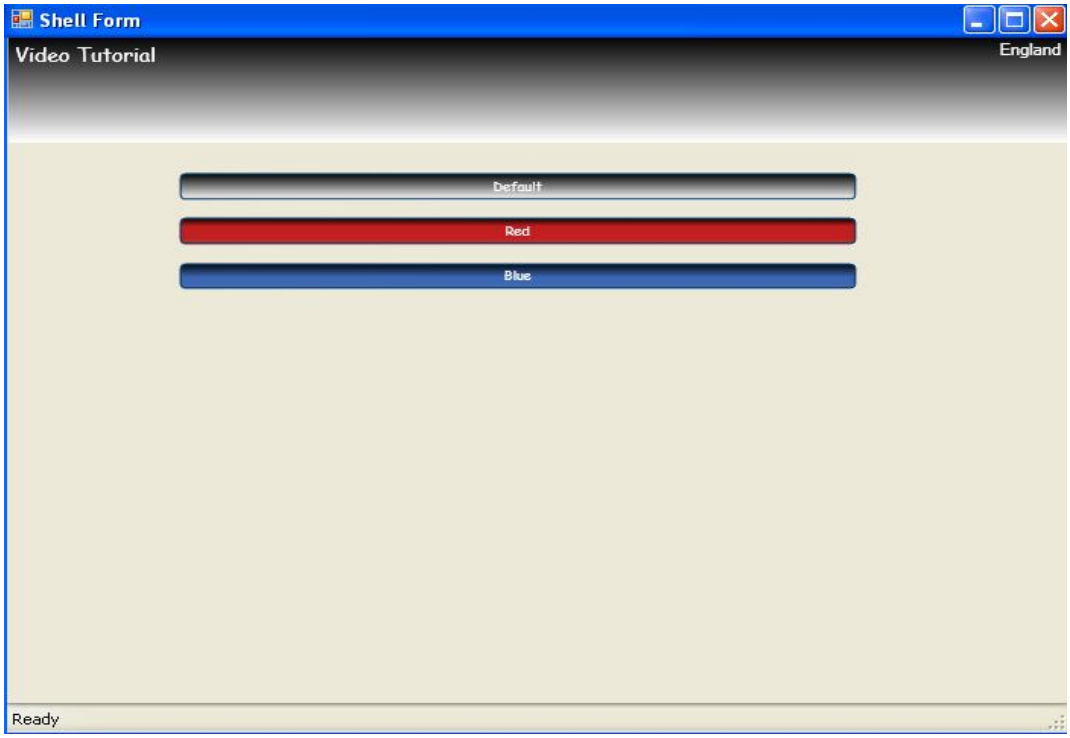

Figura 96: Resultado do exemplo da secção 4.2.5 para a CultureInfo por omissão

Figura 97: Resultado do exemplo da secção 4.2.5 para a CultureInfo "pt-PT"

4.3 Módulo de Visualização de Informação Referente aos Meios Complementares de Diagnóstico e Terapêutica

A segunda fase deste projecto consistiu no desenvolvimento de um módulo, baseado em *WPF*, seguindo as directrizes retiradas do problema anterior, com o intuito de integrar as soluções actuais. Este módulo pretendia implementar uma nova metodologia para a visualização, manipulação e circulação de informação relativa a meios complementares de diagnóstico e terapêutica[4] (exames clínicos), melhorando o processo de interacção entre a aplicação e os seus utilizadores, colmatando ainda algumas limitações funcionais e de visualização que actualmente podem levar a certas inferências dúbias, relativamente à informação associada a cada MCDT.

Esta aplicação pretende gerir a informação relativa a todo o circuito inerente a um exame clínico dentro de um centro hospitalar. Tem como objectivo apresentar uma visão consolidada do estado de um exame, ou seja, se está realizado, se tem relatório, se

[4] Irá ser utilizada a abreviatura MCDT a partir deste ponto.

está empacotado, entre outras informações. Em suma, pretende responder claramente à pergunta: "Qual o estado actual deste exame?".

O circuito de um exame é caracterizado e composto por vários *workflows*[5]. O utilizador tem a possibilidade de fazer "avançar" um dado *workflow*, tendo para tal de efectuar as diferentes operações que o constituem. Este circuito é iniciado pela chegada de um pedido de exame, no grupo Acolhimento, e finalizado, aquando da entrega do mesmo, no grupo Entrega. Outros grupos intermédios fazem, também, parte do circuito: Validação de Exames, Relatório Áudio, Relatório Texto, Assinatura, Correcções, Empacotamento, Movimentação. A aplicação possuirá vários actores: os auxiliares de acolhimento, os administrativos, os estafetas, os técnicos, os médicos e o utilizador monitor, cada um, naturalmente, com os seus papéis bem definidos.

Actualmente, a informação referente a este circuito encontra-se dividida por vários ecrãs de visualização, ou seja, não é possível obter uma visão consolidada sobre os vários estados de um exame, mais concretamente, não é possível, de uma forma fácil e eficaz, aferir os estados de cada um dos *workflows* relacionados com um exame.

A solução encontrada para este problema compreende a criação de um ecrã único para os vários actores deste processo, que será composto por uma grelha com toda a informação relativa aos vários exames e que se pretende adaptável de acordo com o actor. Este ecrã deve fornecer filtros que permitam aceder de forma mais eficaz à informação pretendida. É fundamental que o actor tenha sempre a clara noção do estado de um exame. Este tipo de visão, que resume a informação, permitirá visualizar e manipular de forma abrangente e global toda a informação relativa a um dado exame e, desta forma, maximiza a produtividade dos vários actores.

Após, na primeira fase, terem sido estudados os componentes *UI* da *Infragistics*, foi com naturalidade que se optou por utilizar a grelha para *WPF*, *XamDataGrid*[6], desenvolvida pela *Infragistics* como base do desenvolvimento desta aplicação. Numa primeira abordagem foram estudadas todas as questões relativas à grelha da *Infragistics*,

[5] Conjunto de possíveis acções ou operações sequenciais que se realizam dentro de um determinado grupo.
[6] Componente *UI* para *WPF* comercializado pela *Infragistics*

desde a sua *API*, passando pelas suas capacidades e, claro está, a sua performance. Seguidamente, começou a fase de desenvolvimento propriamente dita do módulo.

Nesta secção, descrever-se-á o estado actual de desenvolvimento do módulo. Este módulo surgiu após a conclusão do principal objectivo deste projecto, a migração das soluções actuais para *WPF*. Inicialmente, este seria apenas um protótipo para testar o sucesso da migração, tendo evoluído, posteriormente, para uma aplicação a adicionar às soluções actuais. Actualmente, foram já levantados todos os requisitos, desenvolvida a arquitectura física que sustentará todas as acções e definido o *layout* da aplicação. Nas secções seguintes, são apresentados os actores da aplicação e são definidos os respectivos papéis e é dado a conhecer o estado actual da aplicação ao nível gráfico.

4.3.1 Actores e Papéis

4.3.1.1 Actores

4.3.1.1.1 Auxiliares de Acolhimento

Os auxiliares de acolhimento podem realizar as seguintes acções no processo de um MCDT:

1. Acompanhar o doente até à sala de exame.
2. Dar início ao exame propriamente dito, ou seja, iniciar o processo de validação.

4.3.1.1.2 Administrativos

Os administrativos são actores com forte intervenção no workflow dos MCDTs. Estes actores podem intervir nas seguintes tarefas:

1. Transcrever um relatório áudio para um relatório em texto.
2. Corrigir um dado relatório de acordo com as indicações do médico responsável pelo mesmo.
3. Entregar pacotes de exames a um destinatário, seja este o próprio doente ou um estafeta.
4. Realizar/continuar/finalizar o processo de empacotamento.

4.3.1.1.3 Estafetas

Os estafetas têm como principal função a movimentação de pacotes de exames, quer movimentações relacionadas com a entrega, quer movimentações relacionadas com o processo interno de realização de um exame (ex: mudança de sala de arquivo).

4.3.1.1.4 Técnicos

Os técnicos são um dos actores mais importantes no processo de realização de MCDTs, uma vez que são estes que validam o exame. Estes podem, igualmente, iniciar o processo de relatório.

4.3.1.1.5 Médicos

Os médicos possuem um papel extremamente importante neste processo. Têm como principal função a validação e assinatura de relatórios. Para além disso, podem gerir o processo de empacotamento (agrupar para entrega) e gerir o processo de alteração de relatórios.

4.3.1.1.6 Utilizador Monitor

Este é um tipo de actor especial, isto é, deve ser visto com um administrador do negócio, ou seja, alguém que queira ter uma visão global de todo o processo, apenas numa óptica de consulta.

4.3.1.2 Papéis

Neste processo existem acções que podem ser realizadas por diferentes actores. A enumeração dos diversos papéis, bem como a associação a cada um dos actores é apresentada na Tabela 2.

Papel\Actor	Auxiliares de Acolhimento	Administrativos	Estafetas	Técnicos	Médicos	Utilizador Monitor
Acolhimento	☑	☑		☑	☑	
Valida Exames				☑	☑	
Cria Relatórios		☑		☑	☑	
Transcreve Relatórios		☑				
Assina Relatórios					☑	
Realiza Empacotamento		☑	☑	☑	☑	
Realiza Entregas		☑	☑	☑		
Realiza Movimentações		☑	☑	☑	☑	
Monitor						☑

Tabela 2: Papéis dos actores

4.3.2 Estado Actual da Aplicação

O ecrã desenvolvido é constituído por três *controls*: uma *WinRibbon*[7], uma *XamDataGrid* e um *control* que permite visualizar os detalhes relativos a cada exame, em função do grupo seleccionado. A figura 98 ilustra o *layout* actual da aplicação.

[7] Componente *UI* para *Windows Forms* comercializado pela *Infragistics*

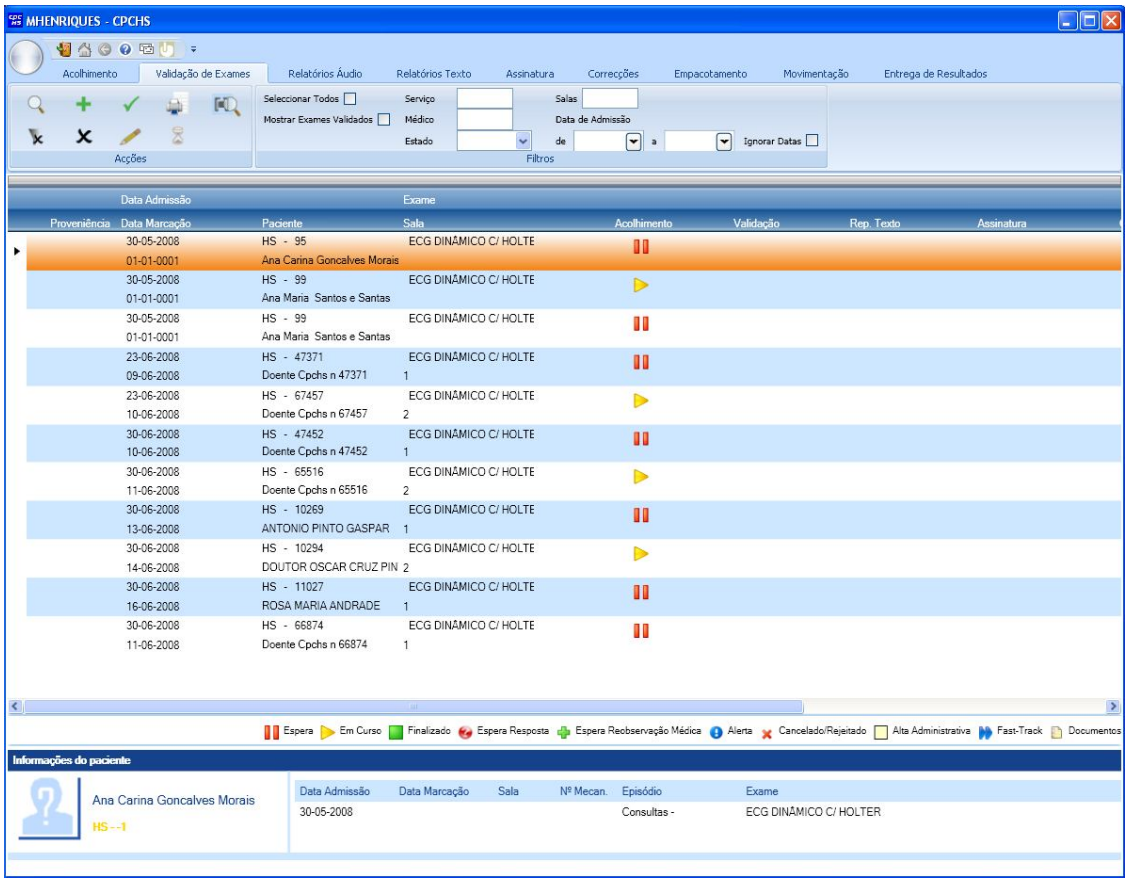

Figura 98: Layout actual da aplicação

A *WinRibbon* contém nove *tabs* (uma por cada grupo). Cada *tab* possui uma secção de filtros e uma secção de acções que permitem "avançar" o *workflow* dentro de cada grupo. Alguns filtros e acções para os diferentes grupos encontram-se já implementados. Ao ser efectuado um duplo *click* na célula referente a cada grupo (Acolhimento, Validação, …) é apresentada uma nova janela, com os exames relativos ao paciente seleccionado. A Figura 99 ilustra uma destas novas janelas, onde se podem, também, filtrar os exames visíveis e realizar acções sobre os mesmos. Por fim, a figura 100 ilustra a acção de filtrar os exames pelos diferentes campos de informação referentes a um exame.

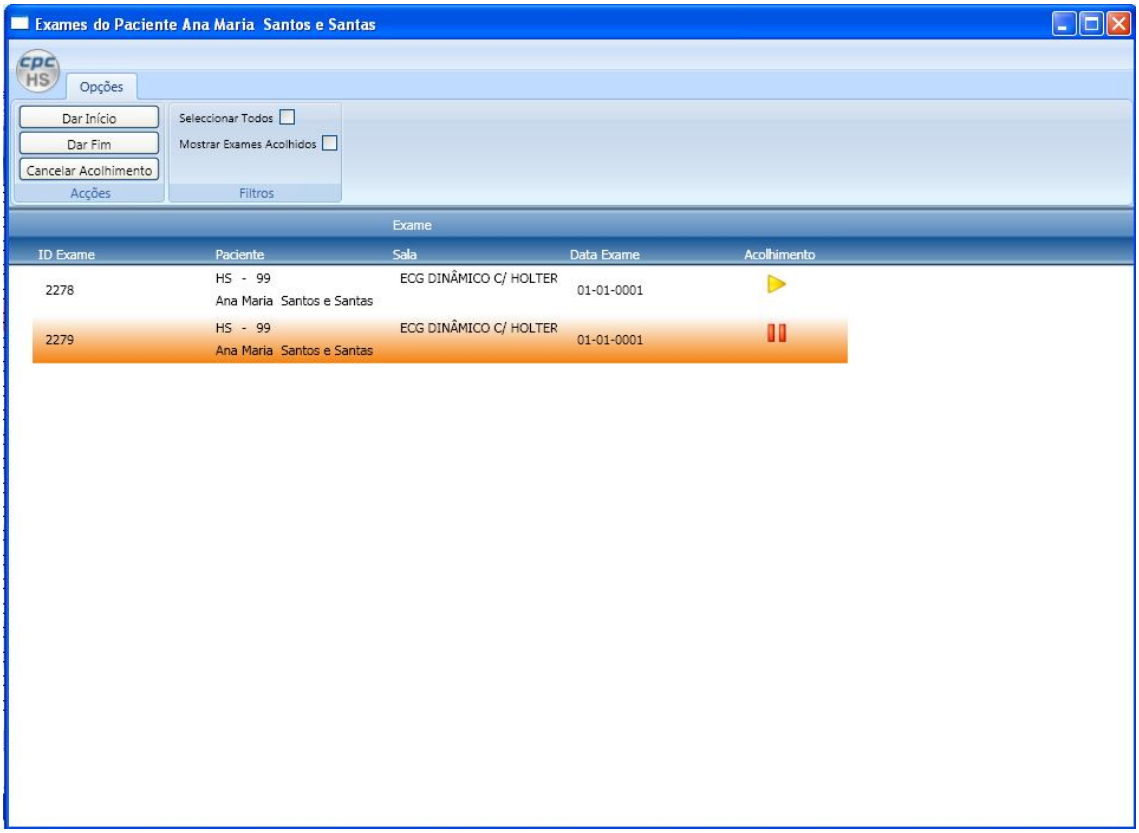

Figura 99: Janela com todos os exames relativos a um paciente

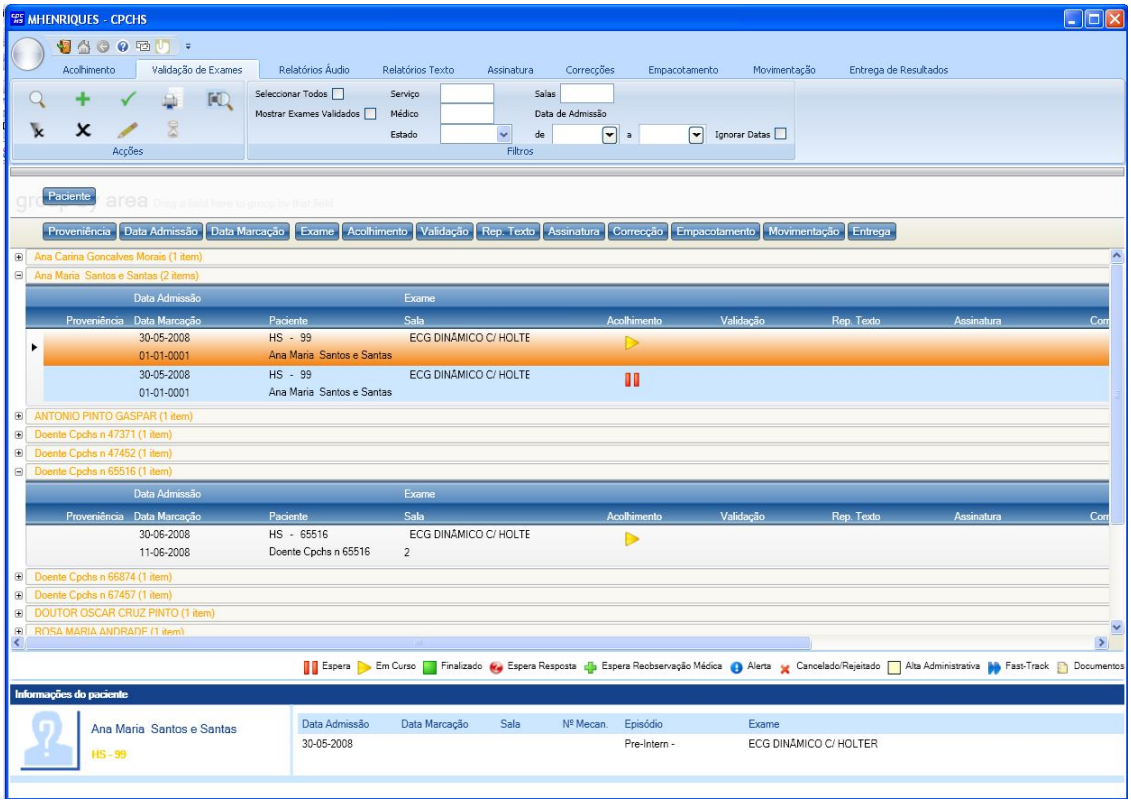

Figura 100: Acção de filtrar os exames pelos diferentes campos de informação referentes a um exame

Capítulo 5

Conclusões e Trabalho Futuro

Neste capítulo, é apresentado um resumo do trabalho realizado e apreciada a satisfação dos objectivos que se pretendiam alcançar com a realização deste projecto. É apresentada, também, uma lista de contribuições principais do trabalho e algumas linhas orientadoras de trabalho futuro.

5.1 Satisfação dos Objectivos

O principal objectivo do projecto foi avaliar a viabilidade da migração das soluções actuais, cujas interfaces estão implementadas em *Windows Forms*, para a tecnologia *WPF*, os custos desta migração, as suas vantagens e desvantagens e qual a melhor forma de ser realizada, ou seja, se a migração poderia ser realizada passo a passo, ou se deveria ser total.

Em primeiro lugar, foi necessário efectuar o estudo das soluções clínicas em *Windows Forms*, de forma a averiguar em que pontos as interfaces poderiam ser melhoradas. Estas soluções estão implementadas sobre a *framework CAB* e utilizam componentes da *Infragistics* para *Windows Forms*. Foi necessário, também, analisar as implicações da utilização dos componentes da *Infragistics* e qual o seu impacto na integração com *WPF*. No seguimento desta análise, colocou-se outra questão que se

prendia com a possibilidade de utilização dos componentes da *Infragistics* para *Windows Forms* simultaneamente com *WPF*.

Com o decorrer do projecto, outras questões se tornaram particularmente relevantes como o estudo da *SCSF*, que fornece auxílio para programadores e arquitectos de *software*, uma vez que, engloba já a *CAB* e gera o código comum a todos os componentes dos *Smart Clients*, diminuindo assim substancialmente o tempo de implementação das soluções. Tendo em conta as suas propagadas vantagens, tornou-se fulcral para o projecto analisar se seria vantajosa a sua utilização e a utilização das suas extensões desenvolvidas no projecto *SCC*.

Foram analisadas outras possíveis vantagens para o projecto: a utilização da ferramenta *Microsoft Expression Blend*, para melhorar a qualidade das interfaces e a possibilidade de se expandir globalmente o *software* através de "localização" na *WPF*, ou seja, a capacidade de traduzir os recursos da aplicação conforme o idioma pretendido.

O estudo das soluções (para os problemas) foi complementado com protótipos que cimentaram conhecimentos e serviram, ao mesmo tempo, de casos de teste. O estudo centrou-se nas tecnologias *Windows Forms* e *WPF*, a qual engloba a linguagem *XAML*. Este estudo contemplou, também, uma análise profunda da interoperabilidade entre *Windows Forms* e *WPF*. Esta análise foi uma das chaves para o sucesso deste projecto.

Este estudo cobriu, também, ferramentas como a *CAB*, a *SCSF*, a *Microsoft Expression Blend* e os componentes *UI* da *Infragistics*, utilizados pelas soluções. No decorrer deste estudo, foram analisados os códigos fonte quer da *CAB*, quer da biblioteca da *Infragistics*, que permite a integração dos componentes da *Infragistics* na *CAB*, de maneira a perceber se existiriam problemas na integração com a *WPF* e, em caso afirmativo, como poderiam estes ser ultrapassados. O estudo contemplava, também, testes para aferição do grau de sucesso da migração nas soluções actuais e o desenvolvimento das infra-estruturas de suporte necessárias à migração das soluções em *Windows Forms* para *WPF*.

Do estudo efectuado ao longo do projecto foram retiradas um conjunto de directrizes, que serviram de orientação, para ser efectuada a migração. Por fim, foi desenvolvido um protótipo, baseado em *WPF*, com a finalidade de comprovar a validade destas directrizes, e, consequentemente, da migração.

Os objectivos traçados para este projecto foram cumpridos na sua totalidade. O estudo da *WPF* permitiu desenvolver directrizes de utilização para esta tecnologia e averiguar os seus pontos fortes e fracos relativamente a *Windows Forms*. A *WPF* possui algumas vantagens relativamente a *Windows Forms*. Estas vantagens são:

- Incorporação de todas as funções do *.NET 2.0*, acrescentando às interfaces novos recursos como 3D, animações, gráficos vectoriais, reconhecimento de voz, *layouts* avançados, entre outros.
- Implementação do conceito de separação entre o *design* e o código, permitindo separar os processos de implementação de interfaces (*designers*) e de escrita de código (programadores).
- Utilização dos recursos do sistema em que se encontra a operar, de forma a optimizar a performance da interface tendo em conta o *hardware* da máquina em que está a ser executado.
- Independência da plataforma em que é executado.
- Suporte de um novo modelo de desenvolvimento, que permite que quase todos os *controls* possam alojar outro *control*.

Relativamente aos pontos fracos, a *WPF* não suporta, ainda, todos os *controls* disponibilizados por *Windows Forms* e a sua aprendizagem é bastante mais complicada que a necessária para se desenvolverem interfaces em *Windows Forms*.

A análise da interoperabilidade entre *WPF* e *Windows Forms* foi efectuada com sucesso. Esta análise, juntamente com o estudo da *CAB* e da biblioteca da *Infragistics*, que permite a integração dos componentes da *Infragistics* na *CAB*, permitiu que fossem compreendidas, na sua totalidade, todas as questões inerentes ao processo de migração das soluções.

Após o estudo da *SCSF* e das suas extensões desenvolvidas no projecto *SCC*, concluiu-se que a utilização das mesmas facilita e acelera o processo de

desenvolvimento de aplicações deste tipo, ou seja, a sua utilização, no contexto actual, é vantajosa.

O estudo da *Microsoft Expression Blend* permitiu concluir que a sua utilização, para o desenvolvimento de *designs*, resulta em várias vantagens. A *Microsoft Expression Blend* facilita a comunicação entre *designers* e programadores no desenvolvimento de aplicações com interfaces *WPF*, uma vez que, a aplicação pode ser partilhada simultaneamente pela *Microsoft Expression Blend* e pelo *Visual Studio*. Esta gera, automaticamente, a *XAML*, permitindo que o *designer* apenas se concentre no *desgin* em si, da aplicação. Caso seja necessário, permite que o *design* seja, também, alterado ao nível da programação em *XAML*, o que concede um maior grau de liberdade a quem o está a implementar. Os *designers* podem desenvolver todo o *layout* da aplicação, sem escrever código e os programadores podem interagir com os *controls*, sem necessitarem de alterar a estrutura do *layout*.

Foi também cumprido com sucesso, o objectivo de averiguar a possibilidade de se expandir globalmente o *software*, através de "localização" na *WPF*. Foi provado, de uma forma simples e eficaz, que era possível efectuar esta expansão.

O protótipo foi desenvolvido e testado com êxito. Tendo em conta o sucesso deste protótipo foi decidido integrá-lo nas soluções actuais, sob a forma de um módulo a desenvolver futuramente.

A migração das soluções foi efectuada com êxito e permitiu concluir, relativamente ao custo a ela associado, que não será necessário um esforço elevado para serem migradas as soluções clínicas da CPC | HS para *WPF*, uma vez que, apesar do processo de aprendizagem da *WPF* ser relativamente longo, as infra-estruturas de suporte à migração encontram-se completamente desenvolvidas e funcionais. Devido ao facto do processo de aprendizagem da *WPF* ser relativamente longo, a migração deve ser realizada gradualmente e, preferencialmente, deve congregar os pontos fortes das tecnologias *WPF* e *Windows Forms*, uma vez que a interoperabilidade entre estas duas tecnologias assim o permite, como comprova o módulo, actualmente a ser desenvolvido, que utiliza componentes *WPF* e *Windows Forms*.

5.2 Trabalho Futuro

A realização deste projecto foi, para o autor, uma experiência curricular e profissional bastante aliciante e que permitiu enriquecer os seus conhecimentos sobre várias áreas, sobretudo na área de engenharia de *software*. Foi também uma experiência bastante gratificante ao nível do relacionamento interpessoal.

A realização do projecto resultou, inequivocamente, numa mais-valia, uma vez que permitiu criar as bases necessárias para a migração, para *WPF*, das soluções clínicas, desenvolvidas na CPC | HS. No futuro será realizada gradualmente esta migração. Por sua vez, também continuará a ser desenvolvido o módulo descrito neste relatório.

Este projecto poderá significar uma mudança na forma como são implementadas actualmente as soluções clínicas. Através dos resultados obtidos, podem ser, agora, desenvolvidas soluções que possuam níveis elevados de usabilidade.

Bibliografia e Referências

[Boe88] Barry W. Biehm. A spiral model of software development and enhancement. IEEE 21, páginas 61-72, 1998.

[Cora] Microsoft Corporation. ElementHost Class. .NET Framework Developer Center, disponível em http://msdn.microsoft.com/en-us/library/system.windows.forms.integration.elementhost.aspx, acedido pela última vez em 4 de Julho de 2008.

[Corb] Microsoft Corporation. CultureInfo Class. .NET Framework Developer Center, disponível em http://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo.aspx, acedido pela última vez em 5 de Julho de 2008.

[Corc] Microsoft Corporation. Windows Presentation Foundation. .NET Framework Developer Center, disponível em http://msdn.microsoft.com/en-us/netframework/aa663326.aspx, acedido pela última vez em 3 de Julho de 2008.

[Cord] Microsoft Corporation. Smart Client - Composite UI Application Block. .NET Framework Developer Center, disponível em http://msdn.microsoft.com/en-us/library/aa480450.aspx, acedido pela última vez em 3 de Julho de 2008.

[Core] Microsoft Corporation. patterns & practices - Smart Client Guidance. CodePlex, disponível em http://www.codeplex.com/smartclient, acedido pela última vez em 3 de Julho de 2008.

[Corf] Microsoft Corporation. Smart Client Contrib. CodePlex, disponível em http://www.codeplex.com/%5Cscsfcontrib, acedido pela última vez em 3 de Julho de 2008.

[Corg] Microsoft Corporation. Microsoft Expression Blend 2. Microsoft Expression, disponível em http://www.microsoft.com/expression/products/overview.aspx?key=blend, acedido pela última vez em 3 de Julho de 2008.

[IEEE98] IEEE Committee. ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs). Technical report, IEEE, Março 1998.

[Inf] Infragistics, Inc. Infragistics. Infragistics, disponível em http://www.infragistics.com/Default.aspx, acedido pela última vez em 2 de Julho de 2008.

[Kry] Dmytro Kryvko. Extended Strongly Typed Resource Generator. CodeProject, disponível em http://www.codeproject.com/KB/dotnet/ResXFileCodeGeneratorEx.aspx, acedido pela última vez em 4 de Julho de 2008.

[NIE] Jakob Nielsen. Ten Usability Heuristics. Useit, disponível em http://www.useit.com/papers/heuristic/heuristic_list.html, acedido pela última vez em 3 de Julho de 2008.

[Pot96] Mike Potel. MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. Taligent, Inc., 1996.

[Res08] Forrester Research, Inc. Forrester, 2008. Forrester Research, disponível em http://www.forrester.com/rb/research, acedido pela última vez em 4 de Julho de 2008.