Criação de uma *framework* para desenvolvimento colaborativo

Luís Filipe Medeiro Costa

Dissertação submetida como requisito parcial para obtenção do grau de

Mestre em Engenharia de Telecomunicações e Informática

Orientador:
Prof. Doutor Carlos Manuel Jorge da Costa,

ISCTE-IUL

Setembro, 2010

RESUMO

Num mundo em que a evolução das aplicações se tornou uma prioridade é essencial para o seu desenvolvimento que as ferramentas correspondentes possam acompanhar este crescimento.

Este artigo baseia-se numa investigação feita como dissertação de mestrado em que o seu grande foco central é a evolução de uma *framework* para construção de *Rich Internet Applications* (RIAs), na sua utilização e resultados obtidos.

O desenvolvimento em equipa, por vezes pode tornar-se bastante complicado quando pensamos em aplicações de grandes dimensões. Neste tipo de aplicações, o mais importante a considerar antes do inicio do desenvolvimento da mesma, para além das tecnologias que irão ser utilizadas, é a sua arquitectura e estruturação, para que o desenvolvimento conjunto da mesma possa ser feito em sincronia e sem entrar em conflitos aquando o código é cometido para repositórios. Como tal a utilização de uma *framework* que normalize a estruturação do código por ficheiros torna-se essencial, só assim é possível desenvolver uma aplicação de uma forma mais eficiente e coesa.

Como desenvolvedor de RIAs, utilizando o Adobe Flex, decidi estudar mais aprofundadamente a *framework* Mate. Sendo que esta é uma estrutura baseada nos eventos e mapas de eventos, adaptou-se a mesma utilizando conceitos que eram usados em *frameworks* da primeira geração.

Para complementar esta investigação, foi criada também uma aplicação que apenas configurando os nomes de mapas, gestores, eventos e respectivas variáveis, cria logo toda a estrutura já com a maior parte do código incluído, gravando também um ficheiro de configurações, para quem em futuras versões da aplicação esta estrutura possa também ser utilizada para outras linguagens.

PALAVRAS CHAVE

Web Applications; Rich Internet Application (*RIA*); *Actionscript*; *;* Soluções *open-source* e *workflow* para desenvolvimento de RIAs; Cairngorm; Mate; *framework*; Padrões de desenho; Usabilidade; *SVN*.

ABSTRACT

In a world where the development of applications has become a priority is essential to its development as the corresponding tools to accompany this growth.

This article is based on a research done as a dissertation of a masters degree in which the big focus is the evolution and development of a framework, for building *Rich Internet Applications* (RIAs), in its utilization and the obtained results.

The development in team, sometimes, can be very complicated when we talk about applications of big dimensions. In this kind of applications, the most important to consider before the beginning of the development of the same, beyond the technologies that will be used, it's his architecture and structure, so that the collaborative development can be made in synchrony and without conflicts when the code is committed to repositories. So, the utilization of a framework that normalizes the structure of the code by logical files it's essential to make possible to develop an application in a more efficient and correct form.

As a RIA developer, using Adobe Flex, I decided to investigate more depth the Mate framework, as this is a very structure free, being only based on events and their maps, so I have tried to improve it. So, I decided to adapt it using concepts used on other Adobe Flex frameworks from the first generation.

To complete this investigation, I've created an application that only configuring the names of maps, managers, events and respective variables, creates all the structure of the framework including all the code in his respective files, also saves a configuration file, so that other users, in future version, could load them and create the structure in other developing languages.

KEYWORDS

Web Application; Rich Internet Application (RIA); Actionscript; ; open-source solutions and workflows for development of RIAs; Cairngorm; Mate; framework; Design Patterns; Usability; SVN.

ÌNDICE

Pág.

RESUMO I
PALAVRAS CHAVE II
***ABSTRACT* ..... III**
***KEYWORDS* ..... III**
ÍNDICE IV
ÍNDICE DE FIGURAS VI
ACRÓNIMOS E ESTRANGEIRISMOS VII
AGRADECIMENTOS ..... *XI*

1 – Introdução…… 1
1.1 – Introdução…….. 1
1.2 – Objectivos…….. 1
1.3 – Problema……..… 2
1.4 – Motivação……...…… 3
1.5 – Estrutura do Documento…… 4
2 – Revisão da Literatura…….. 5
2.1 - Conceitos Teóricos……... 5
2.1.1 - Padrões de desenho…….. 7
2.1.1.1 – Exemplo de padrões de desenho…….. 8
2.1.2 - Padrões de desenho em MVC……. 10
2.1.3 - Como utilizar um padrão de desenho…….. 12
2.2 – *Frameworks*…… 12
2.2.1 - A *framework* Cairngorm…… 14
2.2.2 - O estado corrente das *frameworks* para Flex……..…. 17
2.2.3- Mate……..……18
2.2.3.1 - Como funciona o Mate…… 19
2.3 – Enquadramento tecnológico…….. 21
2.3.1 – Aplicações *Desktop* vs Aplicações *Web*…….. 21
2.3.2 - *Rich Internet Applications*…… 24
2.4 – Conclusão..... 27

3 – Proposta Conceptual……... 28
3.1 - Visão geral das frameworks ……..……..…….. 28
3.2 – Proposta de *framework*……...…….. 29
4 – Implementação…..……..……... 36
4.1 - Desenvolvimento de *Rich Internet Applications*..... 36
4.2 - Framework Creator – Facilitando o trabalho de quem desenvolve…….... 38
5 –Utilização……..……... 43
5.1 - Solução *Client-side*: Adobe Flex……. 43
5.2 - Solução *Server-side*: .NET + SQL Server Express…….. 44
5.3 - Comunicação entre o cliente e o servidor…… 45
5.4 - Construção da aplicação – Utilização da *framework*……. 46
5.4.1 - Projecto Gestor de Associados……. 47
5.4.1.1 – Associados…… 48
5.4.1.2 – Actividades……... 49
5.4.1.3 - Gestão de Dados…… 50
5.4.1.4 – Estatísticas……. 52
6 – Conclusões finais…….. 54
6.1 – Passos futuros……. 55
7 - Referências Bibliográficas…… 56

ÌNDICE DE FIGURAS

Pág.

Fig.1 – Utilização do Model (Observer) para colocar os dados nas Views…… 10
Fig.2 – Modelo Model-View-Controller…… 11
Fig. 3 – Processo de comunicação de dados entre cliente e servidor……. 14
Fig.4 – Esquema das pastas e ficheiros utilizando a *framework* Cairngorm…. 17
Fig.5 – Aplicação *Desktop* vs Aplicação *Web*……..... 22
Fig.6 – Enquandramento das RIAs nas soluções actuais…… 25
Fig.7 – Estrutura dos padrões de desenho na Framework Cairngorm …… 28
Fig.8 – Estrutura dos padrões de desenho na Framework Mate ……. 29
Fig.9 – Divisão lógica dos gestores de resultados de eventos…… 33
Fig.10 – Estrutura dos padrões de desenho na Framework desenvolvida…… 33
Fig.11 – Escolha do caminho do projecto……... 39
Fig.12 – Criação dos mapas de eventos e dos gestores de resultados…… 40
Fig.13 – Criação dos eventos…….. 41
Fig.14 – Esquema das pastas e ficheiros utilizando a nova *framework*……. 41
Fig.15 – Ficheiro de configuração do projecto…… 42
Fig.16 – Ecrã de Login da aplicação……. 47
Fig.17 – Área de navegação da aplicação…… 48
Fig.18 – Área da gestão dos associados…….. 49
Fig.19 – Área de gestão de Actividades……. 50
Fig.20 – Plano de Pagamentos……..... 51
Fig.21– Inserção de Associados…… 51
Fig.22 – Inserção de Actividades…… 52
Fig.23 – Gestão de Administradores……... 52
Fig.24 – Zona de estatísticas…….. 53

ACRÓNIMOS E ESTRANGEIRISMOS

.NET - Plataforma para desenvolvimento e execução de sistemas e aplicações. Todo e qualquer código gerado para **.NET**, pode ser executado em qualquer dispositivo que possua uma *framework* de tal plataforma

AJAX - Asynchronous Javascript and XML, que representa um conjunto de tecnologias que quando usadas em conjunto se tornam extremamente poderosas permitindo o desenvolvimento de RIAs.

AMF - Action Message Format . É o formato das mensagens usadas pelo Flash Remoting

Apache - Servidor *Web* gratuito, compatível com o protocolo HTTP.

API - Application Programming Interface . É um conjunto de rotinas e padrões estabelecidos por um *software* para que este possa ser utilizado por programadores no desenvolvimento de outras aplicações.

Cache - Consiste em dotar um computador da capacidade de armazenar temporariamente parte ou a totalidade de um conjunto de informação armazenada num servidor, para diminuir o número de comunicações.

Callbacks - Código executável ou função que é passado por argumento a outro código.

Combobox - Componente gráfico usado em aplicações informáticas que consiste numa lista de items que o utilizador pode seleccionar.

CSS - Cascading Style Sheets , é uma linguagem usada para representar a apresentação gráfica de um documento.

ECMAScript - Linguagem de programação de scripting, sendo a base de Javascript , Jscript , e Actionscript.

Firewall - Aplicação ou dispositivo de *hardware* que se encarrega de filtrar todos os pacotes que são trocados entre dois computadores, de forma a evitar penetrações indesejadas nos sistemas.

framework **-** Estrutura de trabalho, sobre a qual um projecto de *software* pode ser desenvolvido.

GUI - Graphical User Interface . Nome dado ao interface gráfico com o utilizador.

HTML - Hyper-Text Markup Language . Linguagem de formatação utilizada para definir o aspecto e conteúdo de uma página *Web*.

Hosting - Serviço que gere servidores de Internet , e permite a individuos ou organizações alojar conteúdo *Web*

HTTP - Hypertext Transfer Protocol, é o protocolo usado na Internet para transportar informação.

IDE - Integrated Development Environment, é um ambiente de desenvolvimento de *software*.

Interface - Definição da forma de comunicação entre duas entidades, *software* ou *hardware*.

Internet - É uma rede de redes de computadores, que permite a transferência de todo o tipo de dados.

Javascript - Linguagem de scripting interpretada em *run-time* normalmente embebida em HTML .

LAN - Local Area Network. Nome dado a uma rede local de computadores.

Listbox - É um componente gráfico semelhante à *combo-box*, que permite a selecção de itens a partir de uma lista.

MVC - Model View Controller , padrão de desenho que separa uma aplicação em três componentes distintos, o modelo de dados, o interface gráfico e o controlo da lógica de negócio e sistema.

MySQL - Servidor de bases de dados popularmente utilizado na Internet. OCAML Objective Caml , é uma linguagem genérica de programação criada em 1996

Open-Source - *Software* fornecido aos utilizadores dando-lhe a liberdade de o executar, estudar, modificar e redistribuir, sem que seja necessária a autorização dos autores Osflash Comunidade *open-source* de Flash .

Padrões de desenho - É uma solução base para um problema já conhecido, em desenvolvimento de *software*. A solução encontra-se na forma de uma descrição ou de um template de resolução do problema.

PHP - Linguagem de scripting open-source. O PHP é usado essencialmente para aplicações do lado do servidor.

Plugin - Pequena aplicação informática que se pode acoplar numa outra dotando a de novas possibilidades e capacidades.

Protótipo - É a implementação de uma pequena parte de um sistema, com o objectivo de demonstrar o seu eventual potencial.

Scripting - Linguagem de programação mais simples e leve, que as tradicionais liguagens de programação. As linguagens de *scripting* são interpretadas e não compiladas.

SOA - Service Oriented Architecture. Arquitectura de *software* utilizada para permitir que dois sistemas fracamente acoplados possam comunicar, onde o servidor disponibiliza funcionalidades que satisfazem os requisitos funcionais dos clientes, independentemente das tecnologias utilizadas por ambos os lados.

SOAP - Simple Object Access Protocol , é um protocolo que troca mensagens XML numa rede de computadores através de HTTP.

Streaming - Tecnologia que permite que o computador cliente possa interpretar e exibir a informação à medida que esta é recebida durante uma comunicação com o servidor. Este termo é normalmente utilizado no contexto de vídeo ou áudio.

Textbox - Componente gráfico, que permite ao utilizador a introdução de dados através do GUI de uma aplicação.

Thin Client - É um terminal gráfico ou computador cliente que não processa nenhuma ou quase nenhuma lógica da aplicação, sendo essa tarefa deixada a cargo do servidor.

URL - Uniform Resource Locator . Nome dado a um endereço *Web*, normalmente constituído por um conjunto de caracteres que determinam o caminho para um determinado recurso.

Usabilidade - É o termo usado para definir a facilidade com que uma pessoa pode interagir com uma aplicação.

Web-browser - Aplicação informática que se encarrega de interpretar HTML para exibir páginas Web.

WIMP - Window, Icon, Menu, Pointing Device . É um estilo de interacção com uma aplicação usando os componentes referidos.

Workflow - Automatização de processos de negócio, onde existem um conjunto de regras pré-definidas, para passar de um processo para outro.

XML - Extensible Markup Language , metodologia usada para estruturar informação em árvores. É um *standard* que entre outras coisas, facilita a troca de informação entre sistemas.

AGRADECIMENTOS

Gostaria de agradecer a:

Prof. Doutor Carlos Costa pela orientação prestada à dissertação e todo o apoio e atenção prestados ao projecto.

Ao RIAPT pelo apoio nas dúvidas colocadas.

À comunidade Flexcoders pelo apoio nas dúvidas colocadas.

A todos os amigos e família que apoiaram nas inúmeras horas de trabalho e de estudo dedicados a esta dissertação ao longo destes vários meses.

1 - Introdução

1.1 - Introdução

A dissertação de um tema no Mestrado é uma das escolhas mais difíceis no percurso académico de um aluno. É na dissertação que um aluno tem de mostrar todo o conhecimento obtido durante anos de estudo, e para além disso, também as metodologias de trabalho que foi aprendendo assim como a sua vontade de melhorar e a capacidade para solucionar obstáculos que lhe surjam ao longo do seu caminho.

Neste caso, foi decidido optar pela investigação em novas tecnologias e em desenvolver uma nova *framework* para a criação de *Rich Internet Applications* (daqui em diante denominadas por RIAs). Neste documento será explicado a importância das RIAs, tecnologias existentes para a sua criação, uma abordagem mais profunda sobre *frameworks* entre outras investigações feitas ao longo deste ano no âmbito desta dissertação.

1.2 - Objectivos

Para que esta investigação e dissertação pudessem ser completas era necessário cumprir objectivos com diferentes graus de dificuldade, como tal inicialmente deparou-se com objectivos e tarefas mais básicas de modo a aumentar o *know-how* na área das RIAs e das *frameworks*. Para estes era necessário adquirir conhecimentos de várias tecnologias de desenvolvimento de RIAs e dominar em específico uma tecnologia de desenvolvimento, sendo Adobe Flex a escolhida por motivos mais à frente apresentados, e dominar as *frameworks* associadas a esta tecnologia. Para dominar o conceito de *framework* é importante conseguir compreender a arquitectura Cliente-Servidor para uma melhor compreensão do funcionamento das mesmas, assim como compreender e dominar os padrões de desenho e de arquitectura tal como o Model-View-Controller (MVC).

Com estes objectivos básicos compreendidos é possível passar aos desafios propostos para o desenvolvimento desta dissertação.

- Criar e desenvolver uma *framework* genérica de desenvolvimento de RIAs mais adaptada para o desenvolvimento colaborativo;
- Desenvolver pelo menos uma RIA utilizando a *framework* proposta.
- Criar uma aplicação que crie automaticamente a *framework* referida nos pontos anteriores.

1.3 – Problema

A construção de uma *Rich Internet Application* de grande dimensão envolve uma certa complexidade na conjugação das várias linguagens. Como tal para que seja possível fazer uma construção eficiente de uma aplicação escalável tanto em trabalho individual como em trabalho de equipa é necessário recorrer a boas práticas de programação.

É então necessário ter uma *framework* bem adaptada para que o desenvolvimento possa ser feito da forma mais eficiente e coesa. As *frameworks* têm se tornado comuns e importantes. Estas são a maneira como sistemas orientado a objectos alcançam uma maior reutilização.

O grande problema que se têm vindo a deparar com a evolução das aplicações, das tecnologias de desenvolvimento e com o surgimento de novas *frameworks* é o de não existir um equilíbrio nestas *frameworks* quanto à sua rigidez da estrutura, isto é, certas *frameworks* são demasiado rígidas, o que para um arquitecto de *software* quando tenta criar a estrutura base para o desenvolvimento de uma aplicação a maior parte das vezes encontra-se sem opções para que possa variar a estrutura consoante os problemas que lhe forem deparados. Por outro lado, outras *frameworks* são demasiado livres, fazendo com que as opções de um arquitecto de *software* por vezes não sejam demasiado limitadoras e criando conflitos de código quando os desenvolvedores dessa aplicação fazem junção de código dos projectos. A grande questão que se coloca é se será possível encontrar um meio termo entre estas *frameworks* adaptando as vantagens de cada uma numa *framework* única que melhore o trabalho colaborativo, e se será possível que esta possa ser utilizada para mais do que uma linguagem de programação de modo a poder uniformizar o desenvolvimento destas aplicações não interessando qual a linguagem utilizada.

1.4 – Motivação

Um dos grandes desafios é o de ser obrigatório recorrer a boas práticas da Engenharia de Programação, pois sem o recurso a estas o trabalho da equipa pode não ser o desejado. Como tal, ao longo destes últimos meses foi necessário adquirir conhecimentos sobre inúmeros *workflows* (RPC; Cairngorm, PureMVC, Mate) pois cada projecto é diferente e cada qual necessita de um *workflow* à sua medida.

Por isso o processo utilizado durante a investigação para esta dissertação consistiu em:

- Revisão da literatura para perceber os conceitos de padrões de desenho e *frameworks* bem como as *frameworks* em estudo.
- Criação de uma proposta de *framework* através da adaptação de outras *frameworks*
- Desenvolvimento de uma aplicação para a prática desta *framework*
- Criação de uma aplicação que consiga criar a *framework* atrás desenvolvida.

Outro dos factores que permitiu uma grande evolução na aprendizagem das novas tecnologias foi a utilização e investigação de comunidades online onde todo o apoio é sempre garantido, sendo as que mais contribuíram na reposta de dúvidas foram o RIAPT (www.riapt.org) e o Flexcoders (tech.groups.yahoo.com/group/flexcoders).

Ao longo destes meses de dissertação dei uma aula de apresentação de Flex no ISCTE assim como aulas de SIC (Sistemas de Informação e Comunicação) no Curso de Especialização Tecnológica em Desenvolvivmento de Produtos Multimédia e no Curso de Especialização Tecnológica em Fotografia no IADE (Instituto de Artes Visuais, Design e Marketing)

Entretanto também fui construindo outros projectos enquanto estive a trabalhar na empresa Apriva tais como:

- http://www.grupo-holon.pt
- http://www.apriva.pt
- http://www.mpsfarmaceutica.pt
- http://www.saborlisboa.com

1.5 – Estrutura do Documento

Este trabalho de investigação está organizado em sete capítulos.

No primeiro, a introdução, é revisto os objectivos e o problema que motivaram à escolha desta dissertação assim como uma breve abordagem à evolução dos conceitos que levaram ao problema em questão, falando assim no processo utilizado para a investigação.

O segundo capítulo trata-se da revisão de literatura onde é feito explicado o conceito de padrões de desenho, frameworks e são explicadas as frameworks que foram utilizadas para basear a nova Framework assim como o enquadramento tecnológico onde é revisto um estudo sobre as Rich Internet Applications de modo a dar um enquadramento sobre as tecnologias onde é pretendido melhorar

A partir do terceiro capítulo começa o trabalho de investigação, onde neste especificamente, é efectuado a proposta conceptual onde é explicada a nova Framework e as razões que levaram à construção da mesma segundo as configurações finais. No quarto capítulo está a demonstração de uma aplicação criada para implementar a mesma Framework e para terminar o quinto capítulo é onde está demonstrada a utilização desta nova Framework na construção de uma aplicação.

Por último temos o capítulo da conclusão onde se descreve também os passos futuros a serem dados.

2 – Revisão da Literatura

2.1 - Conceitos Teóricos

Para se compreender a explicação da *framework*, é necessário introduzir alguns conceitos teóricos de padrões de desenho e arquitectura. Para evitar que o relatório seja demasiado extenso estes conceitos serão explicados de uma forma resumida.

Para que houvesse uma evolução dos padrões de desenho e das *frameworks* foi necessários que várias pessoas investigassem e tentassem progressivamente melhorar os estudos anteriores efectuados.

"Cada padrão descreve um problema que acontece repetidamente no nosso meio, e o núcleo da solução para esse problema de tal forma que é possível utilizar essa solução milhões de vezes sem nunca o fazer da mesma maneira duas vezes." (Christopher Alexander, 1977)

Padrão de desenho é um conceito que surgiu na década de 70 por Christopher Alexander ao qual este adicionou todas as características e definições que os mesmos devem conter, mas nesta altura, padrões de desenho eram utilizadas para problemas na área da arquitectura. Foi necessário esperar até 1987 os programadores Kent Beck e Ward Cunningham adaptassem este conceito para o desenvolvimento de *software*.

O próximo quadro é um paralelismo feito sobre aprender a ser um mestre do xadrez com ser um mestre em desenhar *software* por D.C. Schmidt e J. O. Coplien, em "Pattern Languages of Program Design" para evidenciar a importância da utilização de padrões de desenho na construção de *software*.

Nível	**Objectivo**	**Tornar-se um mestre do xadrez**	**Tornar-se um desenhador de *software***
1	Aprender as regras	Nomes das peças, movimentos das peças, geometria do tabuleiro, etc…	Algoritmos, estruturas de dados, sintaxe da linguagem de programação, etc…
2	Aprender os princípios	Valor relativo de certas peças, valor estratégico dos quadrados centrais, etc…	Modularidade, orientação a objectos, etc…
3	Aprender os padrões	Para ser um mestre do xadrez é preciso estudar os jogos de outros mestres. Estes jogos envolvem padrões que têm de ser compreendidos, memorizados e aplicá-los repetidamente	Para ser um mestre em desenho de *software* é preciso estudar os padrões nos desenhos de outros mestres

Assim, naturalmente, começaram a ser desenvolvidos padrões de arquitectura baseados na união destes padrões de desenho e por consequente surgiram as *frameworks* através de conjugação destes padrões de arquitectura.

Para a investigação em questão interessam duas *frameworks* em pormenor. A Cairngorm, que pertence à primeira geração de *frameworks* para Adobe Flex, foi desenvolvida pela companhia Intertion::two, adquirida em 2005 pela Macromedia, e logo se tornou a mais utilizada devido às suas qualidades e a ser baseada num modelo de Model-View-Controller (MVC), um dos modelos mais utilizados na área do desenvolvimento de *software*. Mais recentemente surgiu a Mate, já pertencente à segunda geração de *frameworks* para Adobe Flex, foi desenvolvida e mantida pela ASFusion desde Julho de 2008 e surgiu com o intuito de facilitar a gestão de eventos, sendo esta uma das mais utilizadas das *frameworks* de segunda geração.

2.1.1 - Padrões de desenho

Desenhar *software* orientado a objectos é difícil e desenhar *software* orientado a objectos reutilizável é ainda mais difícil. O desenho deve ser específico ao problema em mãos mas também ser suficientemente generalista para poder resolver futuros problemas com diferentes requisitos, fazendo com que se evite ou minimize o redesenho do *software*. Já por inúmeras vezes houve a sensação de que o problema que está a ser resolvido, já tenha sido já resolvido mas sem ser fácil lembrar onde e como? Se fosse possível relembrar os detalhes do problema anterior e como fora resolvido, então seria possível reutilizar essa experiência em vez de a redescobrir. É assim que nasce os padrões de desenho.

O que é um padrão de desenho?

Padrões de desenho, na sua generalidade, são descrições de objectos e classes comunicantes que são customizados para resolver um desenho geral num contexto particular. O padrão de desenho identifica as classes e instâncias participantes, os seus objectivos e colaborações e a distribuição das responsabilidades. Cada padrão de desenho foca-se num problema de orientação a objectos. [17],[19]

Um padrão de desenho tem quatro elementos essenciais [1], [2], [5]:

- O nome, que tem por objectivo descrever o problema, a solução e as consequências em uma ou duas palavras aumentando a abstracção na nossa lógica mental.
- O problema que descreve onde aplicar o padrão, explicando o problema e o seu contexto. Pode também descrever o problema de desenho específico tal como a representação algorítmica como objectos. Por vezes o problema pode incluir uma lista de condições que têm de ser encontradas antes de fazer sentido aplicar o padrão.
- A solução descreve os elementos que constroem o desenho, a sua relação, responsabilidades e colaborações. A solução não descreve um desenho concreto nem a implementação, porque o padrão é como um *template* que pode ser aplicado em inúmeras situações. Ao contrário, os padrões fornecem uma

descrição abstracta do problema de desenho e como uma estruturação generalista dos elementos o resolve.

- As consequências são os resultados da aplicação do padrão. Como a reutilização é muito utilizada em desenho orientado a objectos, as consequências de um padrão inclui o seu impacto na flexibilidade do sistema, escalabilidade ou portabilidade. Perceber as consequências ajuda a entender e avaliar os padrões de desenho.

2.1.1.1 – Exemplo de padrões de desenho

Apesar do tamanho do domínio dos padrões ser provavelmente infinito, a maioria das principais generalizações de padrões para engenharia de *software* precisas de conhecer já estão identificadas. Os padrões de desenho dividem-se em três tipos principais [17],[18],[19]:

1 - Padrões de Criação

- **Abstract Factory** - Um método Factory é um método que fabrica objectos de um tipo particular; Um objecto Factory é um objecto que encapsula métodos Factory.
- **Builder** - Separa a construção de um objecto complexo da sua representação, de forma que o mesmo processo de construção possa criar diferentes representações.
- **Factory Method** - É uma interface para instanciação de objectos que mantém isoladas as classes concretas usadas da requisição da criação destes objectos.
- **Prototype** - O padrão Prototype fornece uma outra maneira de se construir objectos de tipos arbitrários.
- **Singleton**. - Garante que para uma classe específica só possa existir uma única instância, a qual é acessível de forma global e uniforme.

2- Padrões de Estrutura

- **Adapter** - Permite que dois objectos se comuniquem mesmo que tenham interfaces incompatíveis.
- **Bridge** - Desacopla a interface da implementação; Ocultação de detalhes de implementação dos clientes.
- **Composite** - Lida com uma estrutura de elementos agrupada hierarquicamente.

- **Decorator** - Atribui responsabilidade adicionais a um objecto dinamicamente. O Decorator fornece uma alternativa flexível a subclasses para a extensão da funcionalidade.
- **Facade** - Interface unificada para um subsistema; Torna o subsistema mais fácil de usar.
- **Flyweight** - Usa compartilhamento para dar suporte a vários objectos de forma eficiente.
- **Proxy** - Fornece um objecto representante ou procurador de outro objecto para controlar o acesso ao mesmo.

3- Padrões de Comportamento

- **Chain of Responsability** - Evita dependência do remetente (cliente) de uma requisição ao seu destinatário, dando a oportunidade de mais de objecto tratar a requisição.
- **Command** - Associa uma acção a diferentes objectos através de uma interface conhecida.
- **Interpreter** - Usado para ajudar uma aplicação a entender uma declaração de linguagem natural e executar a funcionalidade da declaração.
- **Iterator** - Provê uma forma de percorrermos os elementos de uma colecção sem violar o seu encapsulamento.
- **Mediator** - Cria um objecto que age como um mediador controlando a interacção entre um conjunto de objectos.
- **Memento** - Torna possível salvar o estado de um objecto de modo que o mesmo possa ser restaurado.
- **Observer** - Define uma relação de dependência 1:N de forma que quando um certo objecto (assunto) tem seu estado modificado os demais (observadores) são notificados; Possibilita baixo acoplamento entre os objectos observadores e o assunto.
- **State** – Permite ao objecto alterar seu comportamento quando o estado interno muda.
- **Strategy** - Permite que uma família de algoritmos seja utilizada de modo independente e selectivo.
- **Template Method** -Define o esqueleto de um algoritmo em uma operação adiando a definição de alguns passos para a subclasse.

- **Visitor** - Define operações independentes a serem realizadas sobre elementos de uma estrutura.

Da lista de padrões atrás um dos mais conhecidos é o Observer. Um exemplo actual da utilização deste padrão é quando recebemos no nosso computador actualizações automáticas para alguns dos nossos programas assim que nos ligamos à Internet. O padrão Observer tornou possíveis estas actualizações informando os computadores cada vez que alguma alteração interessante aconteça.

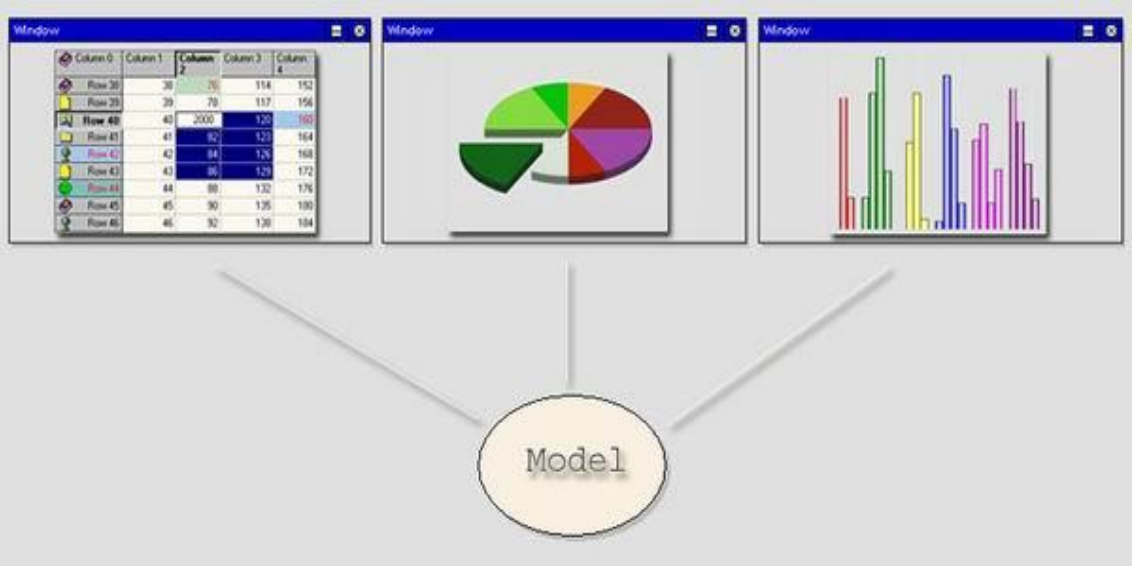

Fig.1 – Utilização do Model (Observer) para colocar os dados nas Views

2.1.2 - Padrões de desenho em MVC

Uma arquitectura é uma forma de representar o desenho de sistemas de *software* que melhora a estrutura das aplicações especialmente em projectos de grande dimensão [21]. O MVC ou Model-View-Controller [22] é uma arquitectura de *software* que possui três componentes distintos o **Model –** componente responsável por agrupar e lidar directamente com os dados – a **View -** consiste no interface com o utilizador, ou por outras palavras, na representação visual dos dados que estão no *model* – e o **Controller -** componente responsável por fazer alterações ao *model* ou às *views*, respondendo a eventos, chamando comandos, e definindo o comportamento da aplicação. Antes do MVC, o desenho do interface de utilizador tendia a juntar estes objectos num só. Com o MVC é possível dividi-los para aumentar a flexibilidade e a reutilização.

O MVC divide as *views* do *model* estabelecendo protocolos de notificação entre os dois componentes. Uma *view* tem de garantir que reflecte na sua apresentação o estado do *model*. Sempre que os dados do *model* alterarem, o *model* notifica as *views* que dependem dele. Esta abordagem permite um acoplar de múltiplas *views* ao *model* para fornecer diversas apresentações, sendo também possível criar novas *views* para um *model* sem ter de o rescrever.

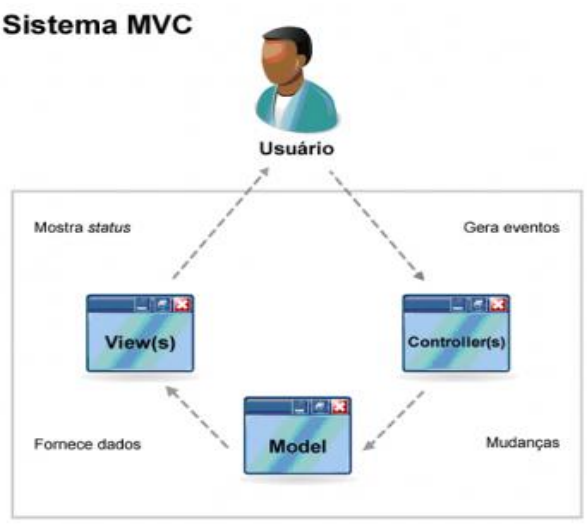

Fig.2 – Modelo Model-View-Controller

O MVC também permite alterar o modo como uma *view* responde ao *input* do utilizador sem ter de alterar a sua representação visual. Por isso, o MVC encapsula os mecanismos de resposta no objecto Controller. Esta é uma classe hierárquica de controladores, tornando mais fácil para criar um controlador como uma variação de um já existente.

Uma *view* utiliza uma instância da subclasse do Controller para implementar uma resposta particular, ao qual para alterar, basta simplesmente substituir a instância por um diferente tipo de *controller*.

A relação View-Controller é um exemplo de um padrão de desenho chamado Strategy. Strategy é um objecto que representa um algoritmo, e é útil quando se pretende substituir um algoritmo tanto estaticamente ou dinamicamente, quando se tem imensas variáveis no algoritmo, ou quando o algoritmo tem estruturas de dados complexas que queremos encapsular.

2.1.3 - Como utilizar um padrão de desenho

Seguidamente, é apresentada uma pequena descrição de como aplicar um padrão de desenho eficazmente [3],[4],[5]:

1. Ler o padrão de desenho e prestar atenção particular às suas secções da Aplicabilidade e das Consequências para assegurar que o padrão é o correcto para o problema.
2. Voltar a estudar às secções da Estrutura, Participantes e Colaborações para se ter a certeza como as classes e objectos no padrão se relacionam entre elas.
3. Ver a secção do Código Simples para se compreender um exemplo do padrão em código e aprender como o implementar.
4. Escolher nomes para os participantes do padrão que sejam importantes no contexto da aplicação. É importante a escolha dos nomes para tornar o padrão mais explícito na sua implementação.
5. Definir as classes. Declarar os seus interfaces, estabelecer as suas relações, e definir as variáveis que representam os dados e as referências dos objectos. Identificar as classes da aplicação que o padrão irá afectar e modificá-las de acordo com os objectivos.
6. Definir nomes para as operações do padrão com a mesma importância que no ponto 4.
7. Implementar as operações que irão realizar as responsabilidades e colaborações do padrão.

2.2 - *Frameworks*

Uma *framework* é um conjunto de classes cooperativas que utilizam um desenho de reutilização para uma classe específica de *software* [23]. Por exemplo, um *framework* pode ser utilizada para ajudar a construir compiladores para diferentes linguagens de programação ou diferentes máquinas, ou também pode ser criada para construir aplicações de gestão financeira. As *frameworks* podem ser customizadas para uma aplicação em particular criando subclasses específicas a partir das classes abstractas da *framework.*

A *framework* dita a arquitectura da aplicação, definindo toda a estrutura, a sua partição das classes e objectos, as responsabilidades ou a colaboração entre os objectos e as classes,

predefine também os parâmetros do desenho para que o programador/desenhador possa concentrar-se nas especificações da sua aplicação, capturando assim as decisões de desenho que são comuns ao domínio da aplicação.

Quando se utiliza uma *framework*, reutiliza-se o corpo principal e escreve-se o código que este chama, tendo de ser escritas as operações com nomes particulares e pedidos convencionais, mas reduzindo as decisões de desenho necessárias de fazer. Não só torna a construção da aplicação mais rápida mas faz com que todas as aplicações tenham uma estrutura semelhante tornando mais fácil a sua manutenção. Por outro lado, perde-se alguma liberdade criativa tendo em conta que a maior parte das decisões de desenho já terem sido tomadas.

Se uma aplicação é complicada de desenhar, uma *framework* é ainda mais complicada. Um desenhador de *frameworks* tem de assegurar que a arquitectura funcionará para qualquer domínio da aplicação. É por isso necessário que o seu desenho a faça o mais flexível e extensível possível. Como as aplicações se tornam tão dependentes de uma *framework* é necessário que quando um *framework* evolui que a aplicação evolua também.

Uma *framework* mais madura normalmente incorpora vários padrões de desenho. Os padrões ajudam a arquitectura da *framework* a adequar-se às diferentes aplicações sem haver redesenho. Padrões de desenho e *frameworks* são bastante semelhantes mas eles diferem em 3 maneiras [23]:

- Padrões de desenhos são mais abstractos que as *frameworks*. O positivo das *frameworks* é que estas podem ser escritas na programação e não apenas estudadas, mas executadas e reutilizadas directamente. Em contraste, os padrões de desenho têm de ser implementados cada vez que são reutilizados
- Padrões de desenho são elementos de arquitectura mais pequenos que as *frameworks*. Uma *framework* tipicamente contêm alguns padrões de desenho, mas o contrário nunca acontece
- Padrões de desenho são menos especializados que as *frameworks*. As *frameworks* têm sempre um domínio particular a serem aplicados. Em contraste, os padrões de desenho pode ser utilizados em quase todas as aplicações.

A utilização de uma *framework* para desenvolver uma RIA traz-nos a coesão de uma estrutura que melhora em bastante a organização do código, a escalabilidade e a manutenção de toda a RIA, podendo incluir componentes que ajudem ao desenvolvimento da mesma.

A *framework* Cairngorm [12],[14] foi inicialmente desenvolvida pela equipa da Adobe Consulting, no intuito de melhorar o desenvolvimento das RIAs na altura do ActionScript 2 e do Flash Remoting com Flash MX 2004. Com o melhoramento das tecnologias para o desenvolvimento das mesmas, também o Cairngorm teve de ser revisto e passado para a versão 2.0 mais adaptado à *framework* do Flex 2.0.

2.2.1 - A *framework* Cairngorm

A utilização deste tipo de *frameworks*, baseados em padrões de desenho, implica também alguns conhecimentos sobre estes e como tal mais à frente irá estar apresentado um resumo sobre os vários componentes utilizados pelo Cairngorm.

De seguida é apresentada uma figura que mostra a fase pela qual a comunicação dos dados entre o cliente e o servidor passam desde que é feito um pedido pela aplicação cliente até o retorno dos seus dados:

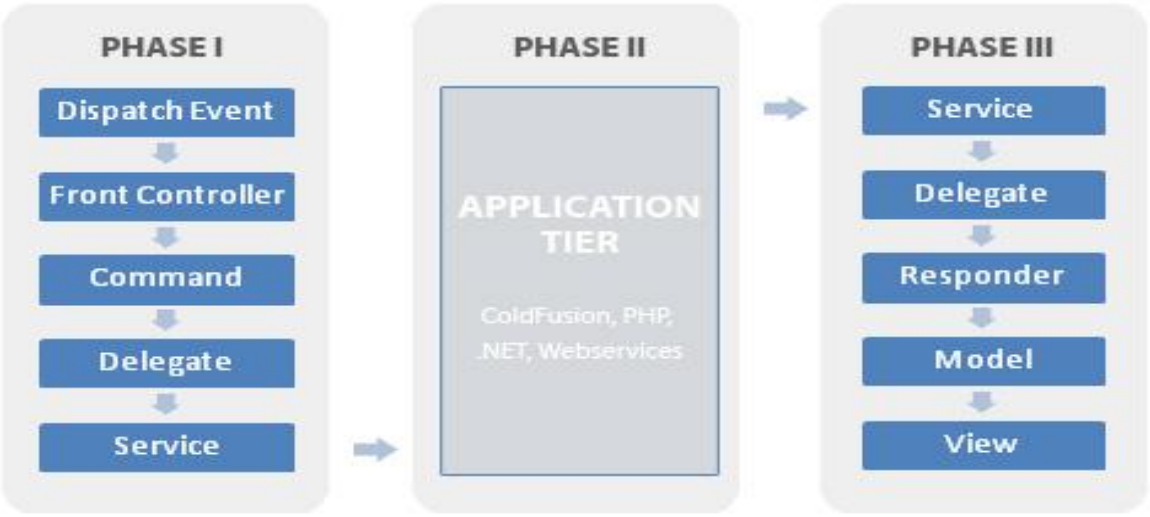

Fig. 3 – Processo de comunicação de dados entre cliente e servidor (fonte: http://www.flexbuilder.direciona.com/?p=54)

Como é possível observar a fase 1 e a fase 3 são aproximadamente iguais tendo apenas uma ordem inversa e enquanto na primeira fase há um envolvimento de eventos e do *controller* já na terceira fase os dados que foram requisitados são colocados no *ModelLocator* para a *view* os poder consumir.

Componentes

Antes de compreender como todos estes componentes funcionam em conjunto é necessário compreender o que é cada um em separado. Seguidamente, serão analisados cada um dos módulos, a saber, ModelLocator, View, Front Controller, Command, Delegate e Service [6]

- ***O ModelLocator*** guarda toda a informação dos *Value Objects da aplicaçã*o assim como das variáveis partilhadas num só lugar. É similar a um objecto de sessão do http excepto pelo facto de este estar alojado no lado do cliente.

- ***O View*** é o interface gráfico que interage com o cliente, constituído pelos componentes do Flex (botões, painéis, etc) e onde os eventos Cairngorm são lançados após as interacções do utilizador (*clicks, rollovers, etc)*

- ***O Front Controller*** recebe os eventos lançados pelas *views* e mapeia-os *para* os *Commands*.

- ***O Command*** contem a *lógica* de negócio, faz a chamada do *Delegate correspo*ndente e/ou de outros comandos, e actualiza os *Value Objects como das* variáveis alojadas no *ModelLocator*

- ***O Delegate*** faz a chamada das funções no servidor através do *Service* e *devolve* os resultados obtidos para o *Command*.

- ***O Service*** define o *Remote Procedure Call (http, Web service, etc) para ac*eder aos dados que estão no servidor.

Juntar os componentes

Para elucidar melhor este ciclo e a integração dos componentes, utilizando a figura 5 como base, é explicado mais pormenorizadamente o ciclo juntando todos os componentes da Framework [6],[7],[8]:

1. Existem as *views* que interagem com o utilizador.

2. Quando, por exemplo, um utilizador pede para ver os dados de um produto, este na *view* insere o número do produto que quer analisar e é lançado um evento (*dispatch*).

3. O *FrontController* está sempre atento aos eventos que são lançados e ao receber um pedido procura qual o *Command* que corresponde ao *Event*. Assim ele faz o requisito ao *Command*.

4. Este *Command* delega o serviço de comunicação de pedido ao seu *Business Delegate*. Os *Business Delegates* normalmente estão divididos por gestores, ou seja, neste caso existiria um *Business Delegate* chamado *ProtudosDelegate* onde está a comunicação com o servidor para chamar funções de adição, modificação, consulta ou eliminação de produtos.

5. O *Business Delegate* de seguida comunica com o servidor pedindo os dados que lhe foram comunicados.

6. No servidor é feito a consulta à base de dados e devolve ao *Business Delegate* que por sua vez envia a resposta ao *Command*.

7. No *Command* existem duas hipóteses de resultado. Ou o *Command* recebe um *fault* caso o pedido no servidor não tenha sido bem sucedido ou então é devolvido um *result*.

8. Caso este *result* se verifique é aqui que os dados são passados para o *ModelLocator* que coloca as referências no *Value Object* correspondente.

9. As *views*, que por boas práticas da programação devem estar à escuta da alteração dos dados do *ModelLocator*, apercebem-se das alterações e alteram-nos automaticamente na interface do utilizador.

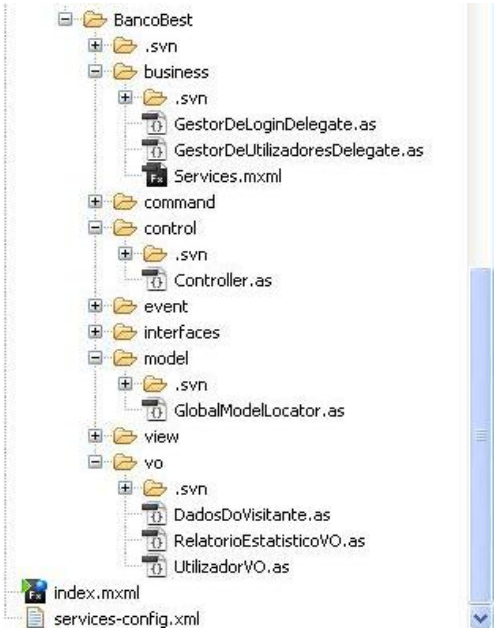

Fig.4 – Esquema das pastas e ficheiros utilizando a *framework* Cairngorm

2.2.2 - O estado corrente das *frameworks* para Flex

No início da expansão do Flex surgiram algumas *frameworks* adaptadas para Flex sendo que a mais desenvolvida e utilizada era a *framework* Cairngorm já explicada atrás. Na sua maioria conseguiu resolver todos os problemas de desenvolvimento de aplicações mas também conseguiu criar novos.

Com a evolução do Flex e a sua maior utilização pela comunidade de desenvolvedores de aplicações web começaram a ser criadas *frameworks* especificas para resolver problemas que antes não tinham sido resolvidos. Foram então criadas várias *frameworks* sendo que actualmente duas das mais utilizadas e melhor criticadas são a *framework* Mate e a Swiz. Qual a melhor a ser utilizada é difícil de escolher sendo que as suas diferenças passam pela implementação das mesmas, acaba por ser uma escolha pessoal e de adaptação.

A escolha de *framework* para utilizar foi a Mate principalmente por ter melhor documentação e assim ser mais fácil compreender a sua lógica de padrões de desenho.

2.2.3- Mate

Mate é uma *framework* que é totalmente implementada em MXML e é orientada aos eventos, sendo este o seu foco central para tornar mais fácil a definição das respostas aos mesmos [11],[13]. Para implementar Mate é apenas necessário ter um ou mais eventos e ter um mapa de eventos, sendo este um ficheiro MXML incluído no código da aplicação, que define os eventos que queremos escutar e qual a resposta a dar ao seu comportamento. Pode ser utilizado mais do que um mapa de eventos se assim preferirmos para organização do nosso código.

No mapa de eventos é também possível implementar a injecção de dados, através da construção de objectos que injectam os dados recebidos noutros objectos da nossa aplicação de modo a estes não precisarem de ir buscá-los pois estes são logo colocados automaticamente. A injecção de dados traz a vantagem de não ser necessário estes estarem contidos num *model* mas também a desvantagem de ser necessário fazer várias injecções caso os dados sejam partilhados em vários sítios da aplicação, como tal, é sempre importante perceber onde é realmente necessário criar injectores ou utilizar um ficheiro de dados central.

Um dos possíveis problemas desta *framework* é não definir uma estrutura para a aplicação, podendo isto favorecer quem desenvolve uma aplicação pois assim pode adaptar o projecto ao seu modo de trabalho, mas obrigando a que uma equipa de trabalho sincronize a sua estrutura de um modo compatível, enquanto que com uma estrutura pré-definida seria mais útil para um projecto onde existam vários desenvolvedores a trabalhar em conjunto.

Como tal, algumas *frameworks* podem obrigar à utilização de uma certa e determinada metodologia de desenvolvimento por terem uma estrutura já predefinida. Com o Mate a nossa aplicação fica livre de uma estrutura pré-definida pois a comunicação para a base de dados pode ser feita no modo que nos for mais confortável em construir através de eventos.

A documentação com que podemos contar é bastante vasta e útil, com bastantes exemplos e explicações de como funciona e que pode ser encontrada, especialmente para quem está a começar a utilizá-la, em http://mate.asfusion.com

2.2.3.1 - Como funciona o Mate

O mais importante a perceber nesta *framework* é como os eventos são tratados [11],[13]. Estes são essenciais no desenvolvimento de uma aplicação com Mate, sendo que são simples extensões dos eventos de uma aplicação normal (flash.events.Event). Isto faz com que os módulos construídos numa aplicação com Mate possam ser reutilizados noutra aplicação que não use *frameworks* de desenvolvimento.

Eventos

Os eventos permitem a um programador saber quando estão a acontecer alterações na aplicação. Estas alterações podem ser causadas pela interacção de um utilizador através do rato ou do teclado, podem ser causadas por ser alguma alteração na aparência dos objectos da aplicação, seja a sua criação ou destruição, ou também por algum resultado proveniente da base de dados.

O resultado destes eventos pode e deve ser utilizado para que a utilização continue a funcionar, e para isso é possível utilizar um controlador/manipulador (*handler*) de eventos. Estes controladores são funções ou respostas escritas para especificar a resposta ao evento.

Quando os eventos são chamadas a um servidor, podemos considerar a existência de uma estrutura diferente nos eventos. Existe a utilização dos *listeners* dos eventos, sendo que estes estão à espera que o evento dispare para fazerem a chamada à base de dados, e quando o resultado é retornado o controlador do evento recebe o resultado para completar o ciclo desse evento. Alguns programadores consideram este ciclo como dois eventos, o de chamar a base de dados e o de receber os dados, enquanto outros consideram-no como um único, considerando ter um *listener* e um controlador.

```
<mx:Button click="criarPasta()" y="68" label="Criar ficheiros"/>
```

Sublinhado em cima a amarelo está um evento associado a um botão, o qual é accionado quando um utilizador clica com o rato sobre este botão. Irá lançar uma função que pelo seu nome irá criar uma pasta no seu sistema informático.

```
var logEvent:LogsSaveEvent = new LogsSaveEvent(LogsSaveEvent.GET);
logEvent.descricao = str;
logEvent.id_admin = geralLocator.admin.ID;
logEvent.nome_admin = geralLocator.admin.Nome;
dispatchEvent(logEvent);
```

Em cima está um evento de *log* (guardar em base de dados os passos feitos por um utilizador numa aplicação, para futura identificação), que pode ou não ser lançado por alguma interacção do utilizador. Este irá lançar um evento para fazer uma chamada à base de dados. Neste caso ele não sabe qual é a função que deverá chamar, apenas lança através do dispatchEvent( ), depois existe um *listener*, ou no caso do Mate um EventMap (que contem o *listener*), que irá escutar o evento e lançar a função correspondente ao evento.

EventMaps (Mapa de Eventos)

O EventMap é uma peça fundamental em aplicações que usem a *framework* Mate. [11],[13] Este é um ficheiro MXML que irá gerir os eventos, lançando as funções correspondentes seja para pedir dados à base de dados como a tratar os resultados provenientes da mesma. Os eventos dentro do EventMap também podem lançar outros eventos. Numa aplicação que use a *framework* Mate é obrigatório ter pelo menos 1, mas também podem ser usados mais caso seja melhor para a estruturação e organização do código.

Dentro de um EventMap encontra-se os EventHandlers e os Injectors (já explicado anteriormente)

```
<EventMap xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
      <![CDATA[
            import events.AdministradoresReadEvent;
      ]]>
</mx:Script>

<EventHandlers type="{AdministradoresReadEvent.GET}" debug="true">
      <RemoteObjectInvoker destination="GestorSociosAdministradores"
            method="Administradores_Search"
            arguments="{[event.id,event.username,event.password,
            event.nome]}" debug="true">

            <resultHandlers>
                  <MethodInvoker generator="{LoginManager}"
                        method="readAdminResult"
                        arguments="{resultObject}"/>
            </resultHandlers>
            <faultHandlers>
                  <MethodInvoker generator="{LoginManager}"
```

```
                        method="readAdminFail" arguments="{resultObject}"/>
            </faultHandlers>
      </RemoteObjectInvoker>
</EventHandlers>

</EventMap>
```

Este exemplo mostra a constituição de um EventHandler num EventMap. A definição de *type* diz ao EventMap que tipo de evento está a ser escutado pelo EventHandler. Assim, neste caso, quando acontecer um evento igual ao que está definido em `AdministradoresReadEvent.GET`, então lançará "Administradores_Search" (definido em RemoteObjectInvoker) que é uma função do *backend* programada no GestorSociosAdministradores. Esta é uma chamada à base de dados, e como tal, irá retornar um valor que poderá ser "ligação com sucesso" ou "ligação com falha". O EventHandler responsabiliza-se por tratar qualquer que seja o seu resultado. Se for positivo então é chamado o método readAdminResult onde os dados irão ser tratados. Por outro lado, se for negativo então será chamado o método readAdminFail que passará alguma informação ao utilizador indicando o insucesso da operação.

É baseado nesta *framework* que se criou uma aplicação de gestão de associados para os Serviços Sociais do Montepio. O objectivo era utilizar a *framework* Mate e tentar melhorá-la à medida que era melhor conhecida, através da prática, para assim ser possível alcançar uma conclusão do estudo sobre as *frameworks*. De seguido encontra-se o estudo com os melhoramentos efectuados à *framework*.

[11,13]

2.3 - Enquadramento tecnológico

2.3.1 – Aplicações *Desktop* vs Aplicações *Web*

Este ponto tem por objectivo fazer uma distinção entre os dois tipos de aplicações, quais as vantagens e desvantagens que cada um apresenta e a razão pelo qual as aplicações *web* não substituem por completo as aplicações *desktop*, mas complementam-nas.

As aplicações *desktop* são um tipo de *software* que já existe em mercado desde o início dos sistemas operativos. Estas são aplicações que têm de ser instaladas nos computadores dos utilizadores que as necessitem. Como exemplos destas aplicações temos Adobe Photoshop, Microsoft Word, Winamp entre muitos outros milhares de aplicações.

Dando exemplos de casos reais de arquitecturas cliente-servidor, algumas empresas necessitam de instalar programas clientes nos computadores de vários trabalhadores para que estes acedam às bases de dados centrais que estão no seu servidor. Na altura em que este programa passa para uma nova versão, na generalidade é necessário proceder a uma nova instalação deste cliente em todos os computadores, para que todos possam usufruir das novas funcionalidades e que geralmente envolve custos tanto em manutenção como na paragem de produção pelo tempo necessário para proceder às modificações.

Com a utilização de aplicações *web* esta desvantagem é ultrapassada pelo simples facto que estas se encontram unicamente instaladas no servidor. Os computadores dos utilizadores apenas desempenham o papel de *thin clients* na medida que estes apenas necessitam de actuar como terminais de interface, sem haver necessidade de nenhuma instalação por parte destes. Com as aplicações *web* um trabalhador apenas necessita de um *web-browser* para estabelecer a ligação http com o servidor e trocar tanto os dados necessários ao funcionamento da aplicação, como a estrutura do próprio interface com o utilizador, sendo este interface trocado uma única vez.

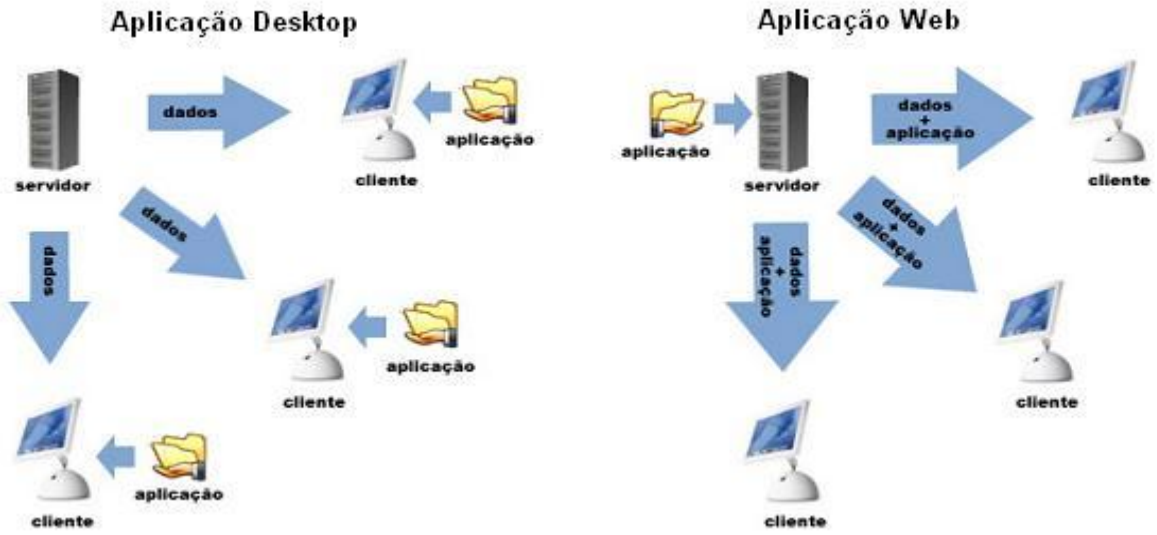

Fig.5 – Aplicação *desktop* vs Aplicação *Web*

A Google é uma das portadoras de maior leque de aplicações *Web* tendo por exemplo o Google Maps, Google Mail, Google Calendar entre muitas outras ferramentas que eles disponibilizam para utilização. Estas são aplicações muito conhecidas e que demonstram o enorme potencial destas ferramentas. O facto de não ser necessário a instalação das aplicações no computador e ter uma acessibilidade a todas elas em qualquer computador com Internet, são as principais razões da sua popularidade. Todas as suas ferramentas apresentam-se como possíveis substitutas de aplicações que antes eram as únicas possibilidades. Existem outros exemplos como Buzzword, uma ferramenta de processamento de texto com uma qualidade quase equivalente ao Microsoft Word com a vantagem de ser uma aplicação *web* e disponibilizada gratuitamente.

Como tal, é de interesse enumerar vantagens e desvantagens das aplicações *web* relativamente às aplicações *desktop*. [9,15,16]

As vantagens são as seguintes:

- Não necessitam de ser instaladas nos computadores dos utilizadores
- Acessíveis em qualquer lugar, independentemente do computador e sistema operativo, através de uma ligação à Internet;
- Os utilizadores não precisam de se preocupar com *upgrades*, correcções e actualizações de segurança;
- Muito pouco vulneráveis a vírus e *spyware;*
- Devido à própria natureza, são logo à partida aplicações distribuídas;
- Redução significativa nos custos de distribuição, manutenção e configuração.

As desvantagens são as seguintes:

- Tempos de espera dependentes da velocidade da ligação, que podem ser intoleráveis em determinados tipos de aplicação;
- É necessário abordar a segurança de uma forma mais séria, visto o tráfego circular "livremente" pela Internet;

- Existe desconfiança por parte dos produtores de *software* relativamente à construção de aplicações em tecnologias diferentes das habituais, especialmente quando se tratam de aplicações que envolvem responsabilidade acrescida (como *software* de gestão);
- Regra geral, as aplicações *web* possuem um GUI limitado, normalmente baseado em HTML.

Com o aparecimento do Adobe Flex e outras tecnologias que permitem a criação de RIAs muitas destas falhas conseguem ser resolvidas devido à maior facilidade de criar um interface mais agradável e rico em tudo semelhante aos das aplicações *desktop*.

2.3.2 - *Rich Internet Applications*

Definir *Rich Internet Application*, mesmo quando se trabalha na área, não é fácil. Esta consegue juntar o melhor de todas as áreas das aplicações, mantendo as inúmeras vantagens em termos uma aplicação *web* com o grande conforto e usabilidade que uma aplicação *desktop* pode oferecer. Uma RIA pode oferecer a um utilizador um interface *WINP* assim como também pode passar algumas tarefas para o lado do cliente, repartindo assim a carga de processamento pelo cliente e pelo servidor. Outra grande vantagem é que dispensam o refrescamento da página a cada comunicação com o servidor em semelhança ao que acontece com as aplicações *desktop*.

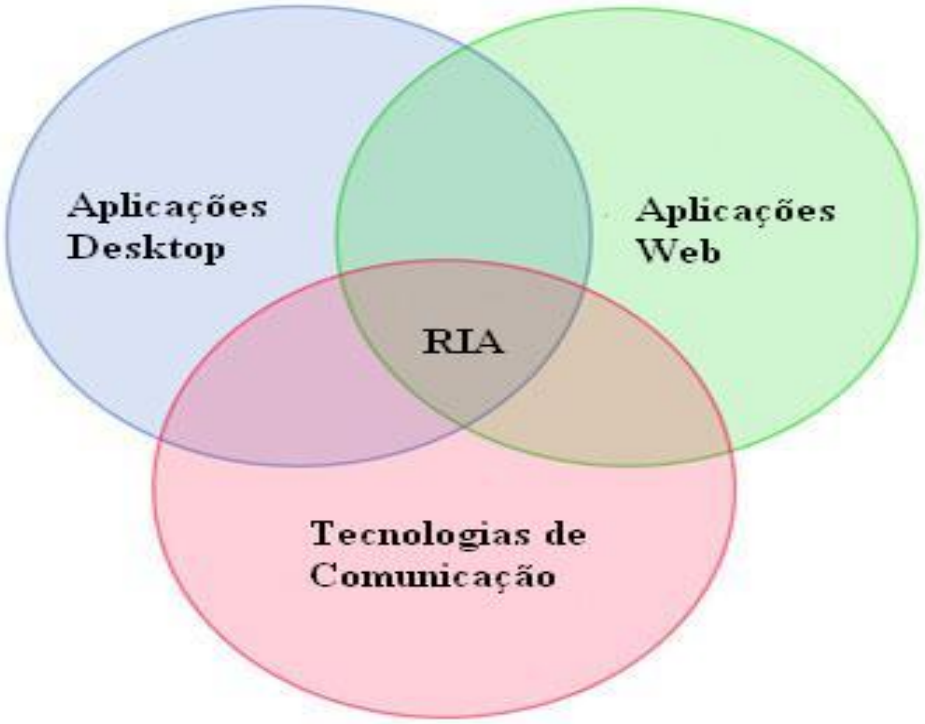

Fig.6 – Enquandramento das RIAs nas soluções actuais

Quando um cliente pretende alguma informação, executa um pedido http ao servidor, normalmente APACHE, verifica se o pedido deverá ser encaminhado para alguma extensão instalado como o PHP. Estas tratam do pedido encarregando-se de gerar dinamicamente a página HTML que é de seguida entregue ao servidor http. Este então refresca a página do web-browser. As aplicações desktop têm a desvantagem de terem que esperar pela comunicação com o servidor e o respectivo refrescamento da página, e quando estes dados são devolvidos têm de ser acompanhados por toda a página HTML para ser refrescada, tornando assim poucos *bytes* de dados em muitos *kylobytes* de informação por um simples pedido.

Nas *Rich Internet Applications*, este problema é resolvido recorrendo ao AJAX, FLEX, ActiveX, ou Silverlight, permitindo a construção de interfaces em tudo semelhantes aos das aplicações *desktop*, e reduzindo largamente a quantidade de informação trocada entre o cliente o servidor, na medida em que só são trocados os dados estritamente necessários.

Então qual a necessidade das RIAs ?

As RIAs permitem a criação de aplicações personalizadas e interactivas que melhoram significativamente a experiência do utilizador, revolucionando a maneira como um utilizador interage com a *web*. Poder criar e fornecer uma experiência eficaz ao utilizador é o que marca a diferença entre um bom produto e outro que não tem sucesso. Por isso, a necessidade de cada vez mais evoluir as aplicações e como tal também as tecnologias para as suas criações.

O aparecimento das RIAs não surge com o objectivo de substituir as aplicações *desktop* mas sim de as complementar. O aumento da largura de banda e o desenvolvimento de novas tecnologias para construção de aplicações mais sofisticadas trouxe ao nosso mundo informático aplicações como Buzzword (www.buzzword.com) um editor de texto, semelhante ao Microsoft Office Word, Picnik (www.picnik.com) um editor de fotografias e Slide Rocket (www.sliderocket.com), semelhante ao Microsoft Office PowerPoint. Muitas destas novas RIAs chegam a ter funcionalidades e eficiência de resposta melhor que algumas aplicações *desktop* com o complemento de ter também outras vantagens maioritariamente em termos de trabalho colaborativo.

O futuro mais provável é que cada aplicação seja escolhida consoante a solução necessária ao problema a resolver de acordo com as vantagens e desvantagens de ambas.

As aplicações *web* apresentam inúmeras vantagens em relação às aplicações de *desktop*, mas obviamente também têm as suas desvantagens. Neste documento serão descritas as principais vantagens, num foco mais pormenorizado, como é o exemplo das aplicações *web* reduzirem os custos de distribuição, de instalação e manutenção pelo facto de ser apenas necessário disponibilizar a aplicação num servidor http para que os seus clientes através de um *web-browser* possam utilizá-la não dependo do local onde estão nem do sistema operativo que utilizem.

Actualmente o grande problema não está em construir aplicações complexas mas em construi-las de modo a serem de fácil manutenção, escaláveis e que permitam um fácil trabalho de equipa. Como tal, com este projecto pretendesse evoluir e conhecer as *frameworks* de desenvolvimento em Flex assim como tentar adaptar uma nova *framework* adaptando as já

existentes que melhore o trabalho colaborativo para a utilização da mesma em outras linguagens de programação.

[9]

2.4 – Conclusão

É então perceptível que as RIAs são o provável futuro nas aplicações informáticas sendo por isso importante que o seu desenvolvimento seja cada vez mais apropriado a cada caso mas sobretudo que sejam cada vez mais desenvolvidas num ambiente de boa Engenharia de Programação para aumentar a sua estabilidade e escalabilidade. Para isso é necessário que as equipas de desenvolvimento utilizem frameworks para que este desenvolvimento seja coerente a cada passo dos projectos. Conseguir utilizar frameworks implica ter alguns conhecimentos sobre padrões de desenho para que a utilização das frameworks seja perceptível a quem desenvolve. Como cada Framework é desenvolvida para determinados objectivos, nem todas eles se preocupam com o trabalho colaborativo nas equipas de desenvolvimento, daí a necessidade de tentar propor uma nova Framework que possa melhorar todo esse processo.

3 – Proposta Conceptual

3.1 – Visão geral das frameworks

Importa, antes de compreender as alterações que foram efectuadas para o desenvolvimento de uma nova framework, perceber a utilização dos padrões de desenho nas mesmas através da elevação de um nível de abstracção

No caso da Framework Cairngorm torna-se perceptível a rigidez da Framework e como tal também a rigidez com a qual deve ser mantida a utilização da mesma.

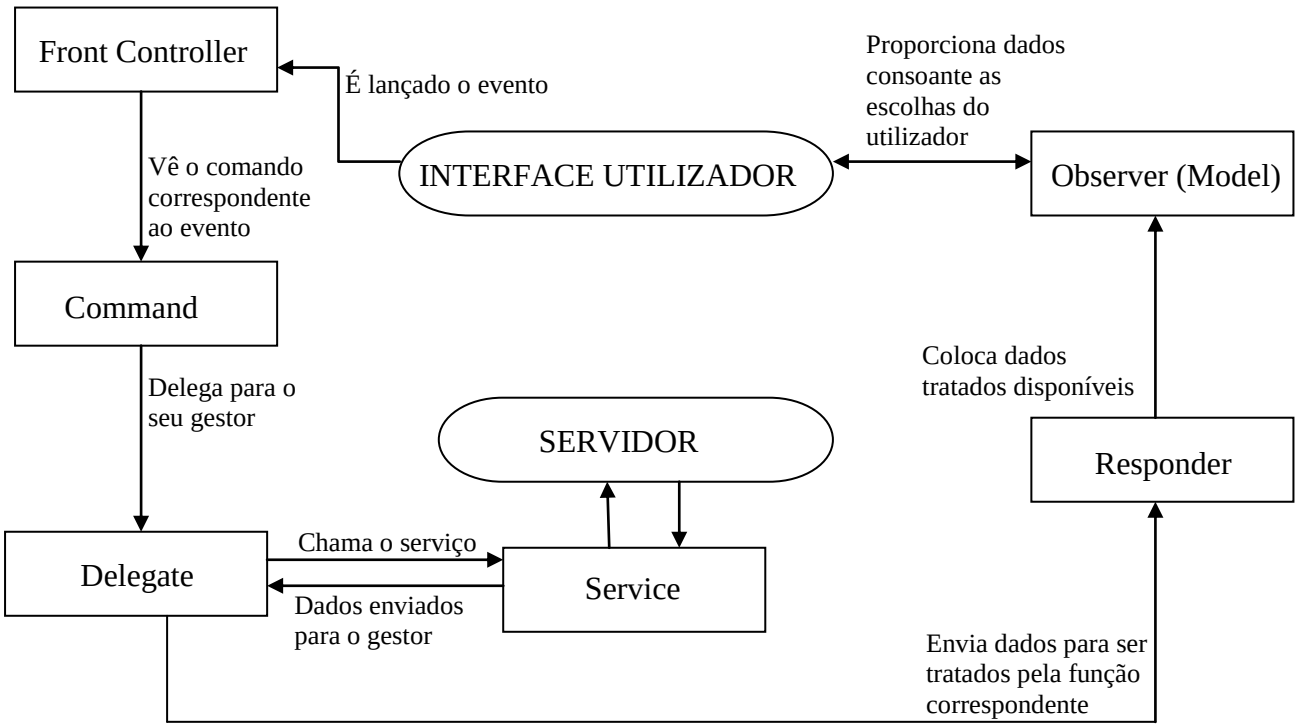

Fig. 7 - Estrutura dos padrões de desenho na Framework Cairngorm

Esta estrutura e dinâmica de fluxo já foi explicada em pontos anteriores.

No caso da Framework Mate torna-se perceptível a simplicidade da Framework e como tal também a facilidade com que pode existir conflitos no desenvolvimento colaborativo de uma aplicação na utilização desta Framework.

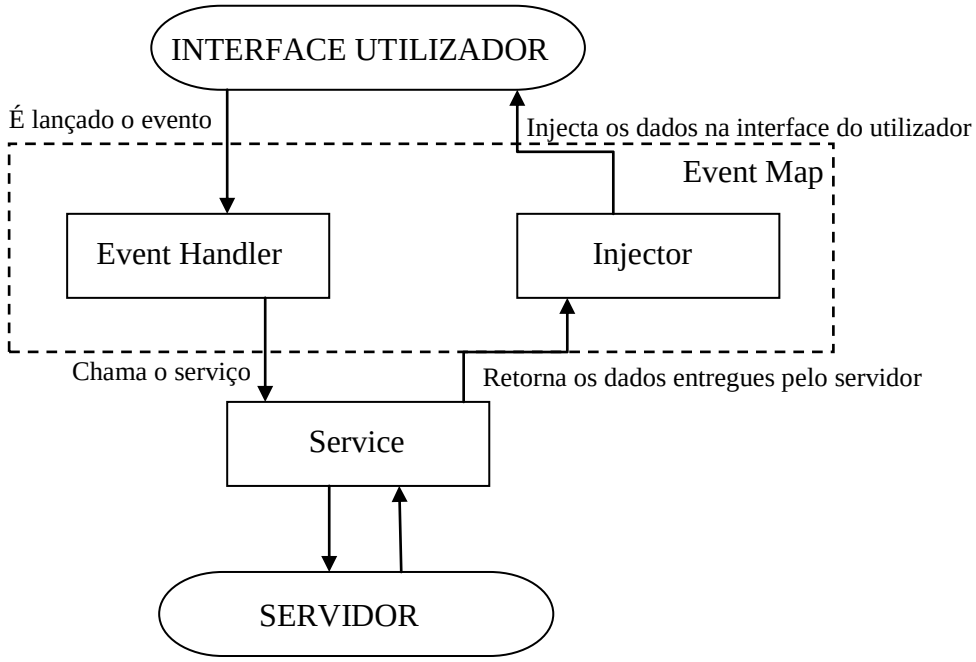

Fig.8 - Estrutura dos padrões de desenho na Framework Mate

Esta estrutura e dinâmica de fluxo já foi explicada em pontos anteriores.

3.2 – Proposta de *framework*

É preciso então encontrar um meio termo entre estas duas frameworks de forma a poder proporcionar um mecanismo rígido o suficiente para evitar conflitos de programação entre colegas de projecto mas também livre o suficiente para que o arquitecto de sistemas possa tomar opções de configuração da estrutura consoante o projecto em questão. Assim é importante acompanhar um exemplo para se chegar a uma estrutura abstracta que possa ser utilizada também noutras tecnologias.

Este exemplo baseia-se na aplicação que foi construída para a compreensão da Framework Mate e desenvolvimento de uma Framework melhorada. Para começar de uma maneira simples, primeiro foi criado o evento para trazer uma listagem de todos os associados

presentes na base de dados. Este é um evento que sucede quando é aberta uma página de associados.

Assim que esta página é mostrada é lançado um evento para a base de dados especificando o que é pedido.

```
var assocEvent:AssociadosSearchEvent = new
AssociadosSearchEvent(AssociadosSearchEvent.GET);
assocEvent.id = -1;
dispatchEvent(assocEvent);
```

A especificação de assocEvent.id = -1 serve para que a base de dados saiba que são todos os associados, fazendo com que não seja utilizado o filtro de *ids* criado nas *queries* em SQL. Assim que se especifica uma ou mais variáveis (havendo a possibilidade de também não ser especificada nenhuma) é então feito o disparo do evento. Um dos problemas do Mate enquanto *framework* para trabalho colaborativo é não definir como obrigatório a criação de ficheiros de eventos à parte, dando a hipótese de este código ser colocado junto com o disparo do evento ou então dando a hipótese que foi escolhida de criar um ficheiro para cada evento à parte.
Por isso decidiu-se que para a *framework* que estava a ser ajustada, para um melhor trabalho colaborativo, que esta deveria ter todos os eventos colocados à parte numa única pasta. Cada ficheiro de eventos é então definido do seguinte modo:

```
“package events{
	import flash.events.Event;

	public class AssociadosSearchEvent extends Event{
		public static const GET: String = "associadosSearch";

		public var id:Number = -1;
		public var nome:String = "";
		public var morada:String = "";
		public var telefone:String = "";
		public var telemovel:String = "";
		public var localidade:String = "";
		public var email:String = "";
		public var bi:String = "";
		public var nib:String = "";
		public var sitprof:String = "";
		public var nsocio:String = "";
		public var ncartaogalp:String = "";
		public var nemp:String = "";

		public function AssociadosSearchEvent(type:String,
bubbles:Boolean=true, cancelable:Boolean=false){
```

```
                super(type, bubbles, cancelable);
            }
        }
}”
```

Assim, tomando esta opção como obrigatória, passamos a ter que toda a arquitectura de eventos fica dividida por pastas, facilitando a cada desenvolvedor de uma equipa saber onde os eventos criados pelos colegas estão localizados. As variáveis de cada ficheiro de eventos são constituídas pela constante que contém o nome que irá ser escutado pelos mapas de eventos, e pelas variáveis que irão ser utilizadas para fazer chamadas à base de dados.

Assim que é feito o disparo, os mapas de eventos estão à escuta do nome para saber qual deles terá de reagir. Neste caso o mapa de eventos que se encarrega de tratar deste evento é o associadosEventMap.mxml.

É neste mapa que se encontra o seguinte manipulador do evento:

```
<EventHandlers type="{AssociadosSearchEvent.GET}" debug="true">

      <RemoteObjectInvoker destination="GestorSociosAssociados"
            method="Associados_Search"
            arguments="{[event.id,event.nome,event.morada,event.telefone,ev
            ent.telemovel,event.localidade,event.email,event.bi,event.nib,e
            vent.sitprof,event.nsocio,event.ncartaogalp,event.nemp]}"
            debug="true">
            <resultHandlers>
                  <MethodInvoker generator="{AssociadosManager}"
                        method="recebeAssociadosResult"
                        arguments="{resultObject}"/>
            </resultHandlers>
            <faultHandlers>
                  <MethodInvoker generator="{AssociadosManager}"
                        method="recebeAssociadosFail"
                        arguments="{resultObject}"/>
            </faultHandlers>
      </RemoteObjectInvoker/>
```

Tal como já explicado anteriormente, este manipulador de evento tem várias definições a ter em conta. O seu *type* onde é definido qual é o evento que está a ser escutado, o seu *destination* onde é definido a biblioteca de *backend* que irá ser utilizada assim com o *method* que corresponde à função existente na biblioteca escolhida para fazer a chamada à base dados. Para esta chamada temos também os *arguments*, que são as variáveis a serem utilizadas para as *queries*. O *type* do evento pode ser definido o nome directamente a ser escutado caso, a opção de *framework* não tenha sido a de utilizar ficheiros para cada evento mas defini-los nos

sítios dos disparos, mas no nosso caso assim é a melhor opção pois torna a aplicação mais acessível a possíveis alterações.

Interessa para os melhoramentos da *framework* compreender como funciona este gestor (Manager), e o que faz com o resultado que lhe é retornado. Quando a chamada ao servidor é bem sucedida então é usado um controlador do gestor para os resultados, quando esta não é bem sucedida então é usado um controlador do gestor para as falhas de resultados. Aqui nesta fase existem várias opções a ter em conta para a formação da *Framework* que serão analisadas:

1. Os métodos para tratamento das situações de sucesso ou não sucesso são escritos no próprio ficheiro do mapa de eventos, ficando o manipulador do evento e as funções de tratamento de resultados todas no mesmo ficheiro. Este é um método bastante fraco de programação pois torna os ficheiros demasiado confusos e demasiado grandes, dando pouca leitura para posteriormente virem a ser analisados por outros programadores.

2. Os métodos para tratamento das situações de sucesso ou não sucesso são divididos em pelo menos dois ficheiros, em que um ficheiro serve só para as situações de sucesso, e outro apenas para as situações de falha. Esta é uma opção bastante viável para um projecto de dimensões pequenas pois agrupa as funções num modo lógico e de fácil acesso. Mas para projectos de maiores dimensões, o ficheiro que trata das chamadas ao servidor com sucesso, pode-se tornar demasiado extenso e complicado de compreender, e tendo em conta a possibilidade de este se tratar de um projecto a ser feito por uma equipa, o facto de os métodos de sucesso estarem todos no mesmo ficheiro torna impraticável a possibilidade de haver vários programadores a trabalharem nesse ficheiro ao mesmo tempo.

3. Os métodos para tratamento das situações de sucesso ou não sucesso são divididos por gestores lógicos de maneira a ser fácil perceber onde é possível encontrar cada método. Assim torna-se mais fácil o trabalho colaborativo pois cada desenvolvedor pode estar a trabalhar num gestor diferente diminuindo os conflitos de código em repositório.

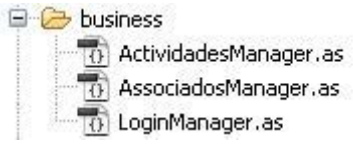

Fig.9 – Divisão lógica dos gestores de resultados de eventos

Por fim, tomasse como opção a utilização do Observer ao invés do Injector pois com este é possível proporcionar o acesso dos dados a toda a aplicação sem ser necessário injectá-los nas várias views onde estes são utilizados.

Assim é possível obter um desenho final de abstracção da Framework resultante:

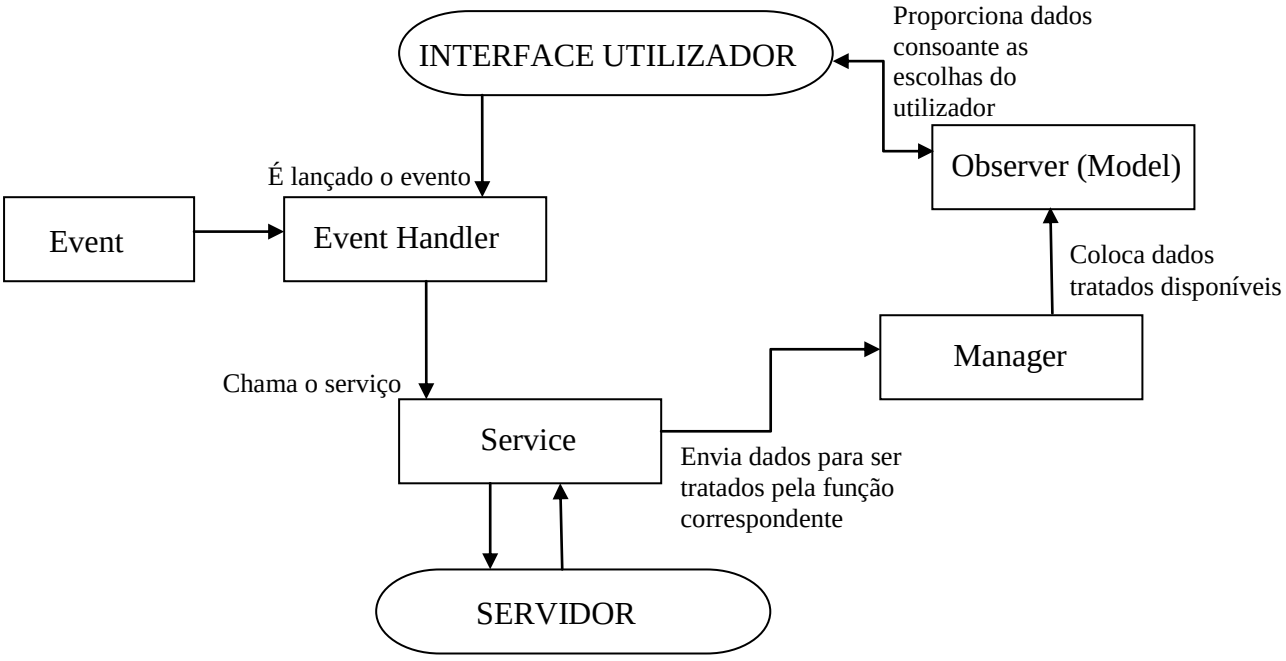

Fig.10 - Estrutura dos padrões de desenho na Framework desenvolvida

De seguida é apresentada uma tabela para que seja mais fácil compreender o que foi aproveitado de cada geração de *frameworks* e em que modo isso influencia o desenvolvimento colaborativo entre programadores.

	Cairngorm	Mate	*Framework* Proposta
Eventos	Os eventos percorrem um grande caminho para a sua chamada ao servidor ser executada.	Mapas de eventos para gerir os eventos e seus resultados	Utiliza os mapas de eventos do Mate mas os gestores de resultados de Cairngorm de modo a diminuir os conflitos de código em repositório.
Model	Bastante utilizado para fazer com que as *views* tenham sempre os resultados actualizados	Não utiliza, usando como opção os Injectors para colocar os dados nas *views*	Optou-se por usar um model global para que os dados estejam disponíveis para qualquer *view* sem ser necessário colocar mais Injectors nos mapas de eventos.
Rigidez de estrutura	Estrutura muito rígida sendo positivo na prevenção de conflitos de código em repositório	Estrutura muito livre sendo positivo para um arquitecto de sistemas que queira inovar na estrutura da aplicação mas negativo na liberdade que dá aos programadores podendo criar alguns conflitos de código em repositório	Estrutura semi-livre tendo em conta que as opções mais críticas de estruturação já estão tomadas, utilizando partes de estrutura de Cairngorm e de Mate.

Facilidade de compreensão	Difícil compreensão mesmo após leitura de tutoriais	Fácil compreensão após leitura de tutoriais	Fácil/Média compreensão após perceber a *framework* Mate e futuramente após leitura de tutoriais.
Gestores de resultados	É utilizado um command que indica o que fazer com os resultados	Tem de ser uma opção do arquitecto como criar os gestores dos resultados dos eventos	Utiliza um gestor de resultados semelhante ao de Cairngorm associado ao mapa de eventos do Mate para trabalhar os resultados e colocá-los no *model*.

Em termos conclusivos sobre a adaptação que foi feita das *frameworks* para a criação de uma nova *framework* é importante referir este é um processo bastante complicado e passa por uma opção de trabalho bastante pessoal. As escolhas feitas podem parecer as mais acertadas e aquelas que mostraram mais eficiência mas com certeza que alguns outros arquitectos de estruturas de desenvolvimento poderiam ter tomado outras opções. Todas as opções que foram tomadas foram com intuito de privilegiar o desenvolvimento em equipa tornando esta nova *framework* como uma que tenta diminuir a possibilidade de conflitos de programação. Em todos os projectos que no qual tive a minha participação este era um dos maiores problemas, o de conflitos de código e é em relação a esse problema que todas as opções se enquadram.

Claro que o total sucesso da nova *framework* só pode ser considerado após esta ser utilizada em inúmeros projectos por inúmeras equipas diferentes que possam partilhar o seu *feedback* acerca da sua utilização. Esse é um passo futuro para poder presenciar tal sucesso.

4 – Implementação

4.1 - Desenvolvimento de *Rich Internet Applications*

Na Europa, o desenvolvimento de RIAs é já algo bastante usual e são inúmeras as ferramentas construídas de sucesso que podem ser utilizadas. O termo *Rich Internet Application* foi usado pela primeira vez em 2002 e desde então que a evolução nas tecnologias para a sua construção tem tido um crescimento exponencial. Recentemente a Adobe desenvolveu o Flash Builder 4.0 (versão seguinte à do Flex 3.0) *e* a Microsoft percebendo que teria de investir neste mercado acabou por lançar o Silverlight. Outra ferramenta que entretanto também é bastante utilizada na construção de RIAs é o AJAX. Surgiu em 2005 e trata-se de um conjunto de tecnologias (HTML, CSS, JavaScript, XMLHttpRequest) que permitem aos *web-browsers* fazer pedidos ao servidor sem necessitarem de refrescar toda a página HTML. A sua popularidade surgiu quando o Google optou por esta para criar as suas RIAs.

Qualquer das soluções, Flex, AJAX ou Silverlight, apresentam as suas vantagens e desvantagens, pelo que foi feito um breve estudo de forma a escolher a mais apropriada para os nossos objectivos. Optei pelo Flex porque:

- Permite integrar interfaces ricos e atractivos que sejam criados através de Flash, não dependendo do suporte dos *web-browsers* para interpretar e exibir o conteúdo multimédia.
- É garantido que a aplicação funcionará correctamente independentemente do *web-browser* ou do sistema operativo.
- Criar uma aplicação em AJAX é uma tarefa complexa, pois é necessário que várias tecnologias diferentes interajam para ser alcançado um determinado objectivo, sendo necessário bastante esforço para garantir a compatibilidade entre diferentes *web-browsers;*
- No Flex é usada uma linguagem de marcação (MXML) para definir a interface e uma linguagem de programação para fazer a parte lógica (ActionScript 3.0). Ambas as linguagens são baseadas em padrões – a primeira nada mais é do que uma aplicação

XML e a segunda é baseada no ECMAScript. O Flex também trabalha com os padrões SOAP, CSS e HTTP.

- O Flex trabalha com *containers* para gerir o *layout* do aplicativo, possuí um sistema de *Data Binding*, possuí um recurso denominado *Validators* para auxiliar na validação dos dados de entrada, possuí um esquema de gestão de dados (*Data Management*) no cliente que lhe permite implementar as melhores práticas de Orientação a Objectos, possuí uma "abordagem padrão" para comunicação com o *Backend* independente do seu serviço remoto ser um Objecto Java, .NET ou um *Web service* , além dos *Formatters, Behaviors*, etc.
- Em AJAX são utilizadas milhares de linhas JavaScript *e* DHTML para fazer pequenas coisas e não possui uma IDE com *debug* integrado.
- Em relação ao Silverlight, este é incompatível com versões antigas do Windows, como o Windows 2000 e 98.
- O Silverlight tem a necessidade de *software* específico para suporte e como tal uma fraca penetração no mercado a curto prazo
- O Silverlight é uma tecnologia recente, logo ainda pouco desenvolvida

Quadro comparativo Flex vs Silverlight: [24],[25],[26]

Adobe Flex	**Silverlight**
Trial de 30 dias (completamente funcional)	*Trial* de 90 Dias (completamente funcional)
Objectivo: Criação e exportação de RIAs (*Rich Internet Aplications*) com suporte entre plataformas.	Objectivo: Criação e exportação de RIAs (*Rich Internet Aplications*) orientadas para a Web
Exporta aplicações no formato SWF, executável com o *plugin* do Flash Player, com uma taxa de 97% de penetração do mercado	Exporta aplicações no formato Silverlight, executável com o *plugin* Silverlight com uma taxa de penetração baixa
Baseado em MXML; linguagem suportada: ActionScript 3	Baseado em XAML; plataforma .NET (linguagens: VBasic, C#, Python, Ruby)
Processo de *streaming* standartizado: aplicação começa a tocar antes de estar completamente descarregada.	Definições de *streaming* com falhas. Código tem de ser "compilado" antes de correr a aplicação. Os vídeos, no entanto, recorrem a *streaming* normal.

Boa documentação disponível online, com o suporte de muitas comunidades online.	Documentação ainda incompleta, com poucas comunidades online
Compatível com qualquer sistema operativo.	Incompatível com versões antigas do Windows, como o Windows 2000 e 98.

Quadro comparativo Flex vs Ajax: [26],[27],[28],[29],[30]

Adobe Flex	Ajax
Na animação tem um grande suporte nativo, bem como possibilidade de personalização.	Na animação tem pouco suporte limitado apenas a efeitos lineares
Bom suporte para manipulação de imagens	Necessário bastante código para uma boa manipulação de imagens.
Suporte de HTML muito limitado, não permitindo tabelas, css ou frames	Bom suporte para HTML
Streaming bastante suportado.	Necessita de *plugins* para um eventual suporte de *streaming*
Em termos de segurança, o código dificilmente será violado devido à distorção do código fonte na compilação, assim como a possibilidade de encriptação.	Na segurança, o seu código pode sofrer violações de terceiros visto existirem algumas reticências quanto à sua segurança
Boa documentação disponível online, com o suporte de muitas comunidades online.	Razoavelmente suportado na documentação como alguns tutoriais e suporte online.

4.2 - *Framework* Creator – Facilitando o trabalho de quem desenvolve

Agora que a *framework* já estava pensada, era necessário criar algo que pudesse facilitar o trabalho para quem desenvolve as aplicações sem ter de estar constantemente a pensar em todos os ficheiros que tem de criar sempre que se quer criar um evento para tratar os dados. Como tal, decidiu-se criar uma aplicação que facilitasse este trabalho.

Criou-se então uma aplicação (ainda sem nome, apenas com o nome de *framework* Creator), que através de uns simples passos, consegue criar toda a estrutura do projecto facilitando

assim o trabalho colaborativo de todas as pessoas participantes no desenvolvimento de uma aplicação.

Esta é uma aplicação feita em Adobe Air e nesta primeira fase cria toda a estrutura do código para Adobe Flex, mas o grande objectivo é que esta seja espalhada por fóruns e comunidade online, para que outras pessoas possam utilizar o código e acrescentar-lhe módulos, de modo a que a aplicação consiga construir a estrutura dos projectos em várias linguagens, seja Adobe Flex, seja Microsoft Silverlight, Ajax ou JavaFx.

São poucos e simples os passos necessários para a criação da estrutura do projecto.

1º - Caminho do projecto

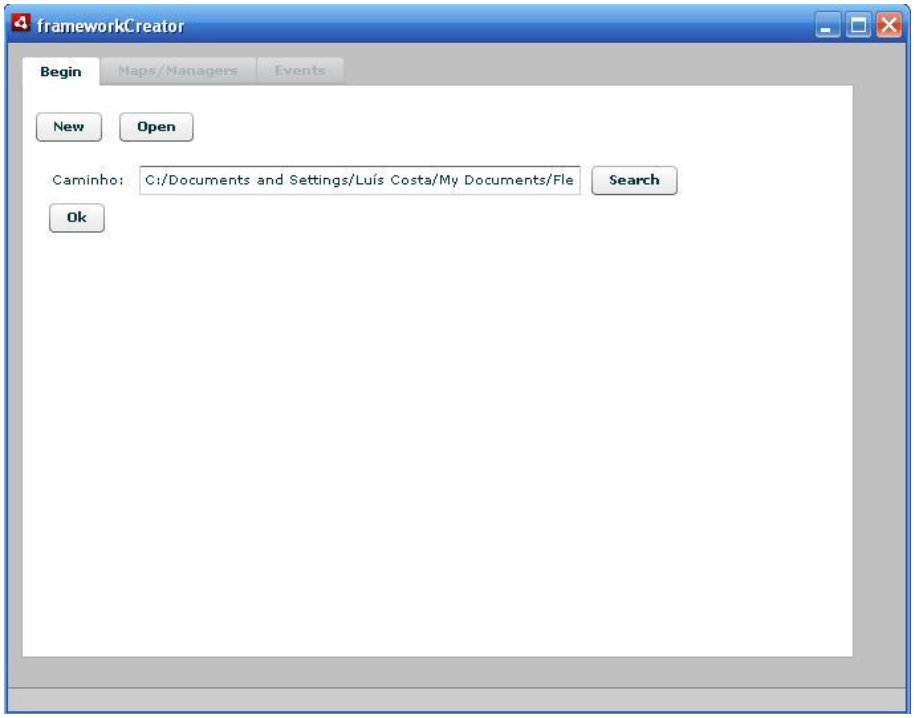

Fig.11 – Escolha do caminho do projecto

Nesta zona da aplicação, é possível criar um novo projecto, ou abrir as configurações de um projecto já existente (explicado mais à frente). Quando se selecciona o caminho para um projecto (projecto que já deve ter sido criado no Flex), o programa vai utilizar o seu caminho absoluto para criar todas as pastas e código a partir do caminho inserido.

Clicando no botão de OK o caminho é guardado como uma variável global do programa e utilizada no fim para caminho da criação dos ficheiros e pastas. Após clicar em OK fica disponível outra área da aplicação, a dos *Maps/Managers*.

2º - Mapas de Eventos e Gestores de Eventos (*Maps/Managers*)

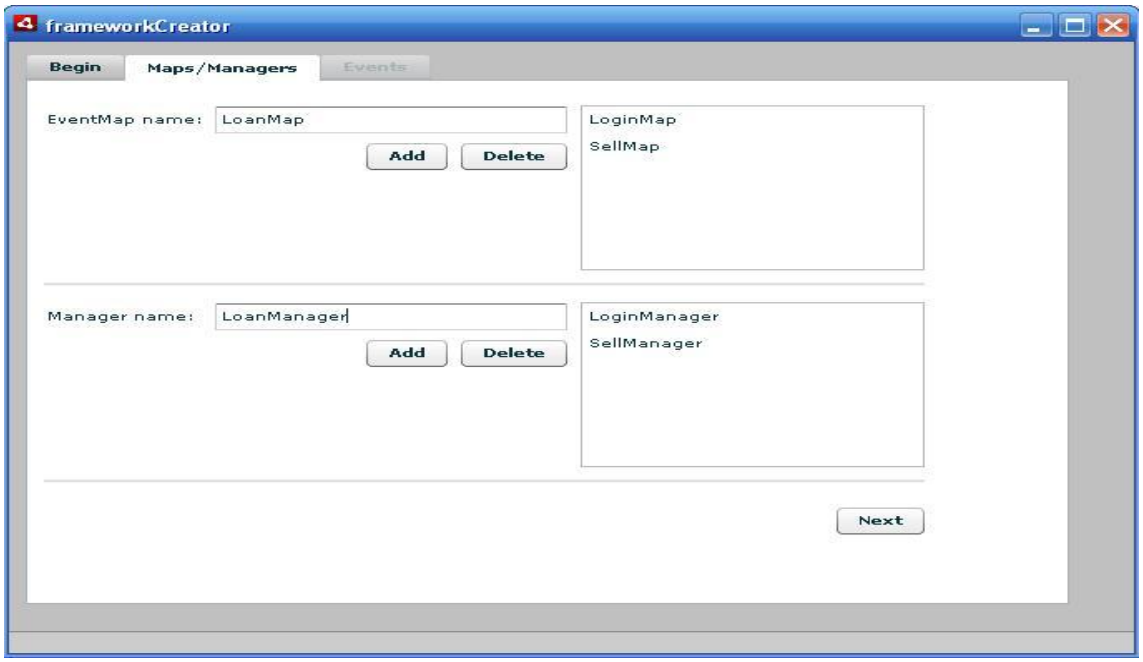

Fig.12 – Criação dos mapas de eventos e dos gestores de resultados

Nesta área da aplicação, o arquitecto do projecto deve adicionar todos os mapas de eventos e gestores que quiser definir para no próximo passo, quando criar os eventos, poder atribuir cada evento ao mapa e gestor correspondente. Cada mapa de eventos e cada gestor será um ficheiro criado na arquitectura do projecto para uma melhor leitura do código por parte dos desenvolvedores.

É tanto possível adicionar como apagar qualquer objecto que já tenha sido criado, e para passar para o quadro seguinte da criação de eventos só é necessário pressionar o botão *Next*.

3º - Criação de eventos

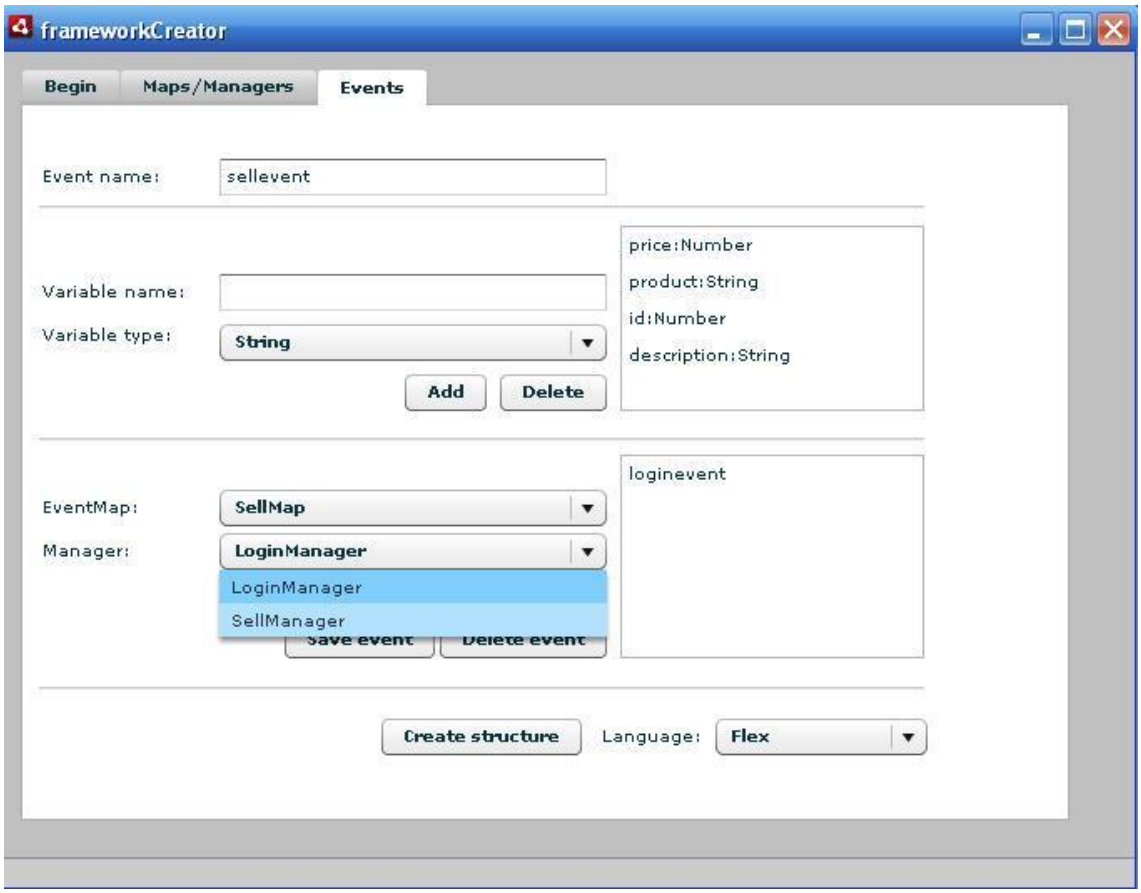

Fig.13 – Criação dos eventos

Esta é a última área de configuração do projecto antes de criar a estrutura. Nesta área o utilizador criará os eventos que tiver arquitectado com as variáveis correspondentes que vai utilizar na chamada à base de dados. Antes de fazer a criação do evento também terá de escolher qual o mapa de eventos a qual este pertence assim como o gestor. À medida que o utilizador for clicando em *Save event*, estes vão sendo criados numa variável global até que seja clicado em *Create structure*.

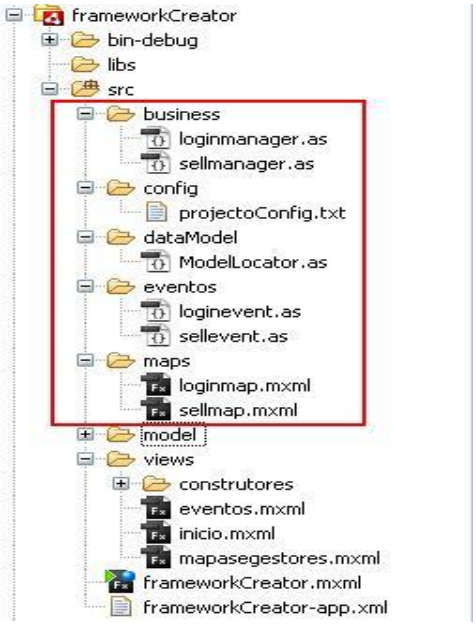

Fig.14 – Esquema das pastas e ficheiros utilizando a nova *framework*

É também possível escolher a linguagem em que queremos que a estrutura e ficheiros sejam criados, apesar de, actualmente, apenas funcionar para Flex, sendo as outras linguagens um testemunho a deixar para próximos investigadores que queiram melhorar a aplicação.

Após clicar em *Create Structure*, a aplicação irá criar os correspondentes ficheiros de mapas de eventos, de gestores de dados e de eventos, mas também criará mais dois ficheiros diferentes. Um é o *dataModel*, ficheiro utilizado na arquitectura MVC como a parte de Model (já explicado em pontos anteriores), e um ficheiro de texto onde estarão guardadas todas as configurações do projecto.

Com a criação deste ficheiro, passa a ser possível no inicio da aplicação utilizar a opção de *Open*. Assim, quando se faz *Open* e se escolhe o ficheiro de configurações do projecto este vai ler todas as variáveis que estarão lá guardadas e passá-las para as variáveis globais da aplicação, para estas poderem ser alteradas e assim criar a estrutura ou noutro computador ou noutra linguagem.

O ficheiro de texto das configurações apresenta-se ordenado por: *Managers*, *Maps*, *Events* (e todas as suas variáveis de cada evento).

[10]

frameworkCreator.mxml | eventos.mxml | projectoConfig.txt

```
1 Project configurations
2
3 Events-Maps:
4 loginmap
5 sellmap
6 Events-Managers:
7 loginmanager
8 sellmanager
9 Events:
10 Nome evento: loginevent
11 Manager: loginmanager
12 Mapa: loginmap
13 Variaveis:
14 username:String
15 password:String
16 Nome evento: sellevent
17 Manager: loginmanager
18 Mapa: loginmap
19 Variaveis:
20 price:Number
21 product:String
22 id:Number
23 description:String
24
```

Fig.15 – Ficheiro de configuração do projecto

5 – Utilização

5.1 - Solução *Client-side*: Adobe Flex

Inicialmente chamado de Macromedia Flex, este foi lançado em 2004 pela Macromedia, e foi produzido com o intuito de ser uma nova tecnologia para a construção de *Rich Internet Applications*. O conteúdo construído em Flex pode ser visualizado através do Flash Player, que é distribuído como um *plugin* que acompanha a grande maioria dos *web-browsers*. Segundo a Adobe, a penetração da versão 9 do Flash Player tem uma penetração sensivelmente de 97% em todos os computadores do mundo com ligação de Internet, tornando assim o conteúdo de Flex perfeito para todos os computadores.

O Flex é geralmente utilizado para a construção da parte funcional do lado do cliente de uma *Rich Internet Application*, mas através da integração com o Flash CS3 é também possível construir interfaces gráficos com toda a qualidade de uma animação Flash. O Flex também é bastante apreciado pela sua capacidade de lidar com todo o tipo de ficheiros multimédia, seja vídeo ou som.

O Flex como ferramenta de programação é bastante versátil, este tem como principais linguagens de programação o MXML e o ActionScript, mas com os *plugins* correctos é possível programar também em Java, PHP, .Net entre outras. O Actionscript, é baseada na sintaxe ECMAScript. Esta é também a sintaxe de Javascript, mas enquanto o Document Object Model (*DOM*) desta última é centrada na janela do *web-browser*, no documento, e nas *forms*, o DOM de Actionscript é centrado em *movieclips*, incluindo animação, áudio, texto e tratamento de eventos. O MXML é uma linguagem baseada no XML e é uma linguagem de construção de componentes visuais para o utilizador.

Pontos fracos no desenvolvimento em Flex

O Flex, apesar de possuir um imenso potencial para a construção de RIAs, ainda necessita de vários melhoramentos para ser a solução perfeita para a implementação destas. O verdadeiro objectivo do Flex é ser uma ferramenta de construção de RIAs como tal será necessário

perceber que uma RIA não é só o lado do cliente. Como tal existem algumas falhas que, como opinião pessoal, dificultam a construção das RIAs. A principal considerada é:

- **Criação de *movieclips* para animação da UI**

  O Flash continua a ser preferencial na construção de *movieclips* para criar animações na interface gráfica do utilizador.

Apesar do ponto fraco o Flex continua a ser a melhor tecnologia na actualidade para a construção de RIAs escaláveis e complexas podendo ser complementado com interfaces muito ricos graficamente utilizando os kits de integração de Flash com Flex.

5.2 - Solução *Server-side*: .NET + SQL Server Express

Para o lado *Server-side* a tecnologia que se utilizou foi .NET em junção com o SQL Server Express. A escolha destas tecnologias baseou-se na recente experiência em desenvolvimentos de projectos adquirida na empresa Apriva, onde trabalhei durante 1 ano, e na qual se desenvolveu vários projectos em Flex com esta solução no lado do servidor. Com a recente aquisição de um servidor pessoal em Windows aproveitei para continuar a utilizar o mesmo processo.

Como opção para base de dados utilizou-se o SQL Server Express. Primeiro dentro das opções do SQL Server esta foi escolhida por ser de livre utilização, e comparando com MySQL e Oracle, esta opção foi tomada, por já existir uma maior habituação à sua utilização, por existir maior apoio em termos de criação de queries, e aproveitando o servidor Windows e o .NET esta é a solução com melhores resultados perante a estrutura de servidor montada. Oracle também seria uma boa opção mas é bastante mais complicada em termos de desenvolvimento, relativamente à experiência que foi obtida em projectos anteriores.

Esta escolha, contudo, não se mostrou demasiado preocupante visto a utilização do .NET também satisfazer todas as necessidades para as funcionalidades requisitadas.

5.3 - Comunicação entre o cliente e o servidor

Com as soluções de desenvolvimento tanto para o lado do cliente como do servidor, é necessário então escolher um processo e protocolo de comunicação entre ambos. Existem alguns processos habituais no Flex para a comunicação:

- ***SOAP***

  Este é um protocolo utilizado na troca de informações estruturadas e consiste em carregar ficheiros com uma estrutura XML, sendo este o formato ideal para devolver estruturas de informações mais complexas, como objectos dentro de outros objectos. Estes documentos XML aderem a um especificação fornecida pelo órgão *W3C.*

- ***Web services***

  Com os *web services* é possível a interacção entre novas tecnologias com as que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Esta é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Este é também uma solução que utiliza dados em formato XML, em que cada aplicação tem a sua linguagem que é traduzida para XML.

- ***Weborb***

  Este é um eficiente método de comunicação *RPC (Remote Procedure Call*) que consiste em chamar um método remoto (no *backend*) de uma forma simples e transparente como se existisse do lado do cliente.

As vantagens desta solução em relação a outras, tornam-na bastante atractiva visto o servidor disponibilizar serviços independentemente da linguagem de programação do servidor. São trocados objectos em que os dados são traduzidos da linguagem do servidor para ActionScript. Tem também a vantagem de os pacotes serem comprimidos e descomprimidos

automaticamente de uma forma eficiente diminuindo drasticamente o tamanho da informação a transmitir.

Tendo em conta as soluções descritas acima, a escolha foi a de utilizar Weborb. Este é uma ferramenta *open-source* feita em Flex, muito rápida e eficiente, que disponibiliza diversas funcionalidades embutidas tal como um *service browser*, um sistema para lidar com autenticação e autorização, paginação automática de *resultsets* extraídos da base de dados, e muitas outras.

5.4 - Construção da aplicação – Utilização da *framework*

O estudo de uma *framework* é algo que não se pode basear meramente numa leitura teórica sobre a mesma. Conseguir compreender ao ponto de utilizar uma *framework* passa por colocar em prática a mesma de modo a conseguir compreender todos os seus componentes e conseguir perceber as dúvidas que vão surgindo à medida da sua implementação.

Foi isso que originou a construção de uma aplicação, surgido através de um convite para o seu desenvolvimento, para que assim fosse possível utilizar a *framework* Mate para a poder compreender melhor e tentar alcançar os seus melhoramentos.

A aplicação que foi proposta para desenvolver é um gestor de associados para os Serviços Sociais do Montepio, que já se encontra online e em testes para garantir que toda a parte de estatísticas se encontra a funcionar correctamente. Esta aplicação pode ser acedida através do URL:
http://77.235.57.104/weborb30/GestorDeSocios-debug/GestorDeSocios.html

Uma *framework* deve, principalmente, ser utilizada na construção de aplicações de grandes dimensões de modo a poder torná-la escalável e de mais fácil manutenção. Como tal será feito uma breve descrição do projecto para se compreender o contexto em que a utilização da *framework* foi considerada, demonstrando os pontos em que foi utilizado os seus mapas de eventos e onde se distingue de algum modo da *framework* original mostrando as razões que originaram a tais decisões.

5.4.1 - Projecto Gestor de Associados

A importância deste projecto e em que o mesmo fosse criado e baseado numa plataforma estável começa pelo cliente em questão. Este, foi pedido pelos Serviços Sociais do Montepio, e o grande objectivo é que fosse uma aplicação online onde dentro de um grupo restrito de utilizadores fosse possível gerir associados, gerir actividades entre outras funcionalidades importantes. O projecto não foi pedido com urgência, o que permitiu ter tempo serem feitas investigações, experiências e testes para se poder construir uma aplicação muito mais estável e já com os melhoramentos que estavam a ser projectados para a *framework*.

O levantamento de requisitos é sempre algo que demora algum tempo e que por vezes faz com que o projecto tenha que ser alterado ao longo do seu desenvolvimento, algo que com a utilização de uma *framework* bem aplicada faz com que não seja um problema de grandes dimensões a resolver.

O projecto englobava então um sistema de *Login* onde os perfis de utilizadores se dividem em dois tipos: Administradores (têm acesso a todas as funcionalidades da aplicação) e Colaboradores (só não têm acesso à gestão de Administradores). Este perfil é definido quando é feita a inserção das credenciais e verificada quando é feito o *Login* através de um campo da base de dados que define o perfil das credenciais que estão a ser utilizadas.

Fig.16 – Ecrã de *Login* da aplicação

Ao entrar na aplicação, esta está separada em quatro zonas de trabalho:

- Associados
- Actividades
- Gestão de dados
- Estatísticas

Fig.17 – Área de navegação da aplicação

Todas as áreas têm a sua importância específica pois cada uma delas permite trabalhar diferentes dados com diferentes finalidades.

5.4.1.1 - Associados

A área de associados tem inúmeras funcionalidades que permitem trabalhar os dados dos membros. Nesta área não é possível inscrever novos associados, sendo esta uma funcionalidade reservada para a Gestão de dados.

Nesta área é possível editar os dados do associado assim como fazer uma pesquisa utilizando campos de filtragem como por exemplo: Nº de Associado, Nº Cartão Galp, Nº de Empregado ou Nome.

É também possível adicionar ou editar Beneficiários que estejam ligados aos Associados (Cônjuge, Filhos, Pais), ver as actividades em que o Associado esteja inscrito e exportar todos estes dados para Excel.

Fig.18 – Área da gestão dos associados.

5.4.1.2 - Actividades

Outra área semelhante à anterior é a área de gestão das actividades. Nesta área tal como na dos associados também não é possível inscrever novas actividades, sendo também uma funcionalidade da Gestão de dados.

Nesta área é então possível editar todas as informações referentes à actividade inclusivamente os valores pagos para participar nestas. É também possível ter uma listagem de todas as actividades, podendo filtrar a pesquisa através de campos como o: Nº de identificação da actividade, Grupo de Actividades, Mês, Ano e Nome da actividade.

Por último também é possível inscrever ou apagar associados na actividade assim como ver a listagem de todos os que já estão inscritos. A exportação do Excel é também uma possibilidade para todas as listagens presentes nesta área.

Fig.19 – Área de gestão de Actividades

5.4.1.3 - Gestão de Dados

Esta é a área onde podem ser inseridos todos os dados de raiz. Pode ser adicionado e gerido os administradores (username, password e perfil de conta), podem ser adicionados Associados onde existe verificação de alguns dados, pois só com estes correctamente inseridos é que é permitido gravar. É também possível inserir novas actividades incluindo os seus preços, assim como adicionar os grupos de actividades.

Por último e uma das áreas mais complicadas da aplicação é a área de plano de pagamentos, onde é possível gerir todos os pagamentos efectuados ou por efectuar relacionado com uma actividade e com os associados. Aqui é possível exportar os mapas dos planos de pagamento assim como os mapas de débitos do corrente mês.

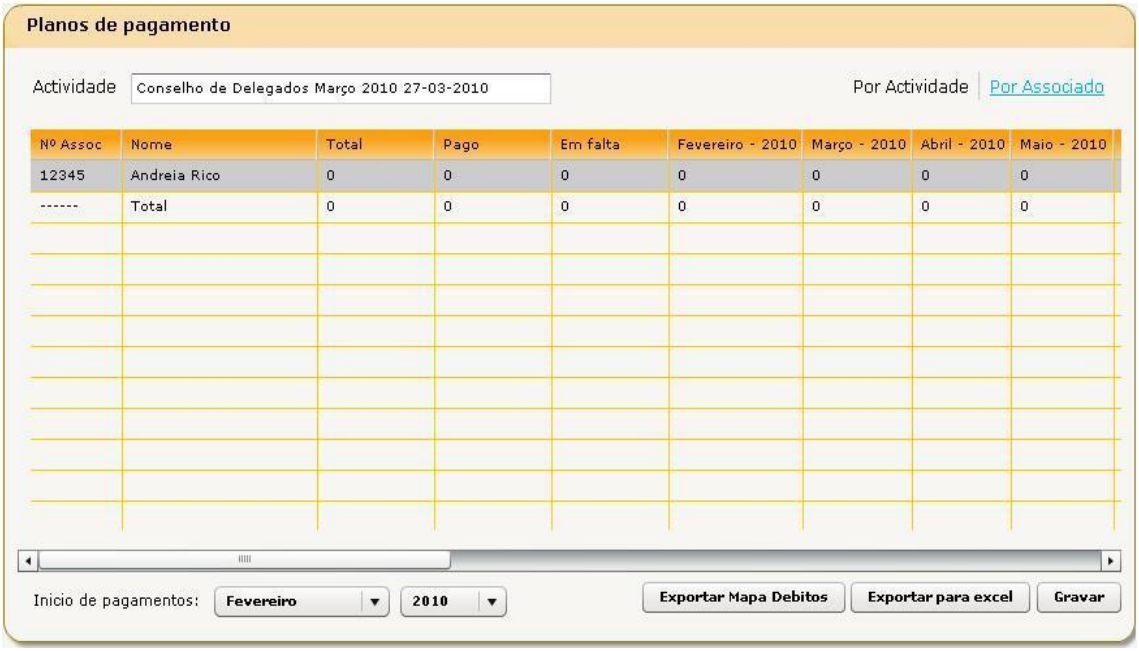

Fig.20 – Plano de Pagamentos

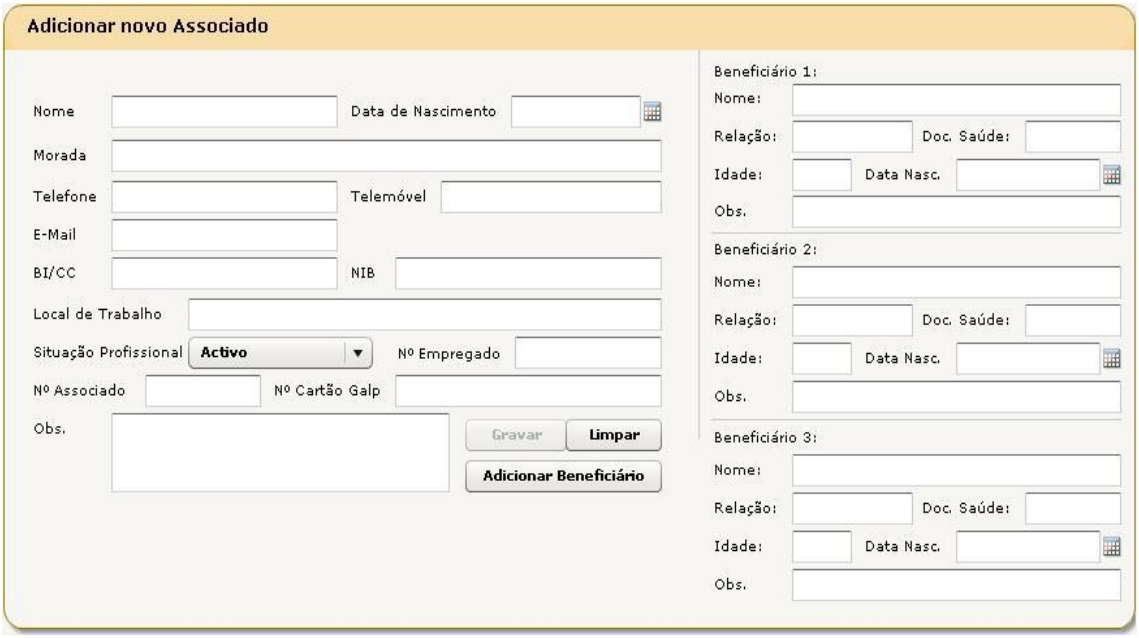

Fig.21 – Inserção de Associados

Fig.22 – Inserção de Actividades

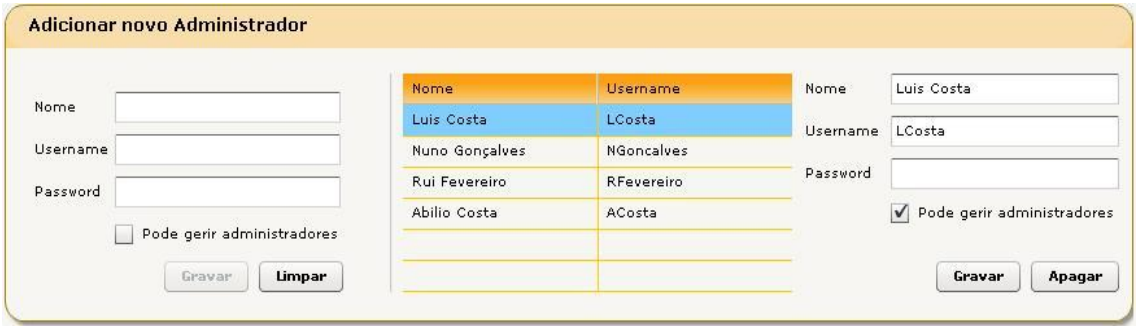

Fig.23– Gestão de Administradores

5.4.1.4 - Estatísticas

Esta área corresponde a uma área de *Business Intelligence* onde é possível aceder aos dados trabalhados em *queries* na base de dados.

Aqui é possível aceder a tabelas e a gráficos com valores para:

- Relação de tipo de associados
- Nº de associados por actividade
- Nº de associados por grupo de actividades
- Nº de actividades por associado

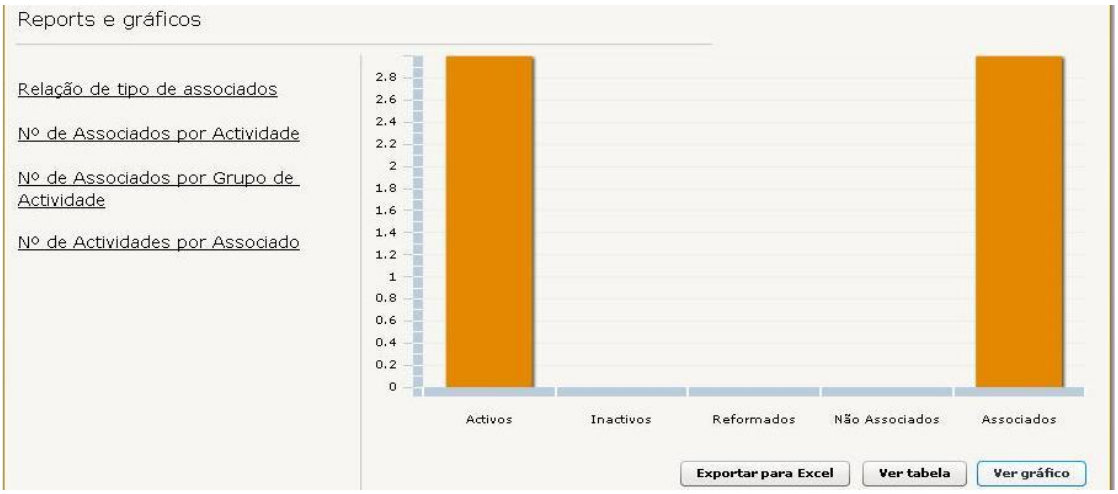

Fig.24 – Zona de estatísticas

Esta é uma área importante pois permite aos seus utilizadores perceber quais as actividades que os associados preferem e assim poder investir mais dinheiro para satisfazer as opções dos associados.

6 – Conclusões

Em termos de conclusões finais, quando foi apresentada esta proposta de dissertação havia a noção que seria bastante complicado poder criar uma nova *framework*, mas devido ao interesse pelas boas práticas de programação pensou-se ser um bom desafio para poder evoluir os conhecimentos pessoais e atingir um novo grau na compreensão e utilização de *frameworks* de desenvolvimento colaborativo.

É perceptível ao longo do documento a importância do conhecimento dos conceitos teóricos para se poder evoluir nas boas práticas da programação e poder criar uma Framework que aumente a capacidade de trabalho colaborativo.

Foi após uma vasta investigação destes conceitos e de frameworks já existentes que surgiu a posposta da qual se espera que através da sua utilização em inúmeros projectos se consiga chegar à conclusão que é mesmo uma das melhores opções possíveis para a utilização em trabalho colaborativo.

Quanto à sua implementação e utilização o processo correu como esperado e todos os pormenores que não tinham sido pensados até à data puderam ser reparados de maneira a tornar a Framework mais estável.

Em relação aos objectivos que tinham sido proposto no inicio deste ano quase todos foram cumpridos com sucesso. Apenas o objectivo de criar possa não ter sido o mais bem sucedido pois na realidade o que foi conseguido fazer foi adaptar as *frameworks* já existentes numa *framework* que ao que tudo indica parece tornar-se uma melhor opção para o desenvolvimento colaborativo em Adobe Flex

Observando os resultados finais, é de crer que se conseguiu fazer uma boa investigação criando inclusive duas aplicações, uma com a nova *framework* e outra para produzir a mesma *framework*, que mostram funcionar até à data sem qualquer problema.

6.1 – Passos futuros

Apesar de todo o trabalho que foi realizado, ainda há um longo caminho a percorrer, principalmente por estas tecnologias terem um constante desenvolvimento. Como próximos passos pretende-se disponibilizar esta ferramenta de criação da *framework* em comunidades de desenvolvedores de *Rich Internet Applications* para que estes possam melhorá-la assim como aumentar novas funcionalidades para que esta também consiga construir a *framework* em outras linguagens de programação. É também objectivo futuro, criar um portfólio pessoal para se tentar começar a angariar clientes para construção de aplicações ou portais e assim progredir através da prática e quem sabe num futuro não tão distante poder ser criada uma empresa de soluções informáticas totalmente orientada a aplicações e sites na *Web* utilizando as boas práticas da programação.

7 - Referências Bibliográficas

[1] - Beck, K. (2007). Implementation Patterns. Addison-Wesley.

[2] - Borchers, J. (2001). A Pattern Approach to Interaction Design. John Wiley & Sons. .

[3] - Coplien, J.; Douglas C. Schmidt (1995). Pattern Languages of Program Design. Addison-Wesley. .

[4] - Helm, Richard; Johnson, Ralph; Vlissides, John (1995). . Design Patterns: Elements of Reusable Object-Oriented *Software*. Addison-Wesley Profissional.

[5] - Freeman, Elisabeth; Freeman, Eric; Bates, Bert; Sierra, Kathy.(2004) . Head First Design Patterns. O'Reilly Media.

[6] - Adobe. *http://www.adobe.com [Online], 2008, "Developing Flex RIAs with Cairngorm Microarchitecture" http://www.adobe.com/devnet/flex/articles/cairngorm_pt1.html*

[7] - Tucker, David. http://www.davidtucker.net [Online], 2007, "Getting Started with Cairngorm"
http://www.davidtucker.net/2007/10/07/getting-started-with-cairngorm-%e2%80%93-part-1/

[8] - Costa, Luis [Online], 2007, "Onde aprender Cairngorm - a *framework* MVC da Adobe"
http://www.riapt.org/2007/12/07/onde-aprender-cairngorm-a-*framework*-mvc-da-adobe/

[9] - Saleiro, João [Online], 2006, "Introdução teórica às tecnologias de desenvolvimento de RIAs baseadas em Flash"
http://www.riapt.org/2006/12/24/introducao-teorica-as-tecnologias-de-desenvolvimento-de-rias-baseadas-em-flash/

[10] - Saleiro, João [Online], 2007, "FlexFuel: gerar código para aplicações Flex/Air baseadas em cairngorm"
http://www.riapt.org/2007/08/12/flexfuel-gerar-codigo-para-aplicacoes-flexair-baseadas-em-cairngorm/

[11] - Asfusion, http://www.asfusion.com [Online], "Getting Started" on Mate Flex *framework* - http://mate.asfusion.com/page/documentation/getting-started

[12] - Adobe. *http://www.adobe.com [Online], 2009, "Choosing a Flex framework" http://www.adobe.com/devnet/flex/articles/flex_framework.html*

[13] - Asfusion, http://www.asfusion.com [Online], 2009, "Mate Flex Framework - Using Flex *frameworks* to Build Data-Driven Applications"
http://mate.asfusion.com/assets/content/presentations/mate_max_2009.pdf

[14] - Powell, Andrew [Online] 2008**, "**Which Flex *framework* Is Right For You?"
http://www.infoaccelerator.net/blog/post.cfm/which-flex-*framework*-is-right-for-you

[15] - SALEIRO, João; FERNANDES, João. Rich Internet Applications**:** Uma visão geral. Disponível em: http://www.riapt.org/files/Designers%20em%20RIAs.pdf. Acesso em: 17 jun. 2007.

[16] - Dana Moore, Raymond Budd, Edward Benson - Professional Rich Internet Applications: AJAX and Beyond; Wrok; March 2007

[17] – Alexander, C, Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S. "A Pattern Language". New York, NY (USA): Oxford University Press, 1977.

[18] – Cooper, J. W. "Using Design Patterns". CACM, junho, 1998, Vol.41

[19] – Coplien, J. O. "Software Patterns". New York, NY (USA): SIGS Books, 1996.

[20] – Gamma, E., Helm, R., Johnson, R., Vlisides, J. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison Wesley, 1995.

[21] Robert C. Martin - "Design Principles and Design Patterns", 2000, http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[22] - Java BluePrints Model-View-Controller, 2000, http://www.oracle.com/technetwork/java/mvc-detailed-136062.html

[23] - Coad, Peter. "Object-Oriented Patterns. Communications of the ACM",1992.

[24] - The Battle for the RIA Throne: Flex vs. Silverlight - Toolbox for IT- (2008), http://it.toolbox.com/blogs/madgreek/the-battle-for-the-ria-throne-flex-vs-silverlight-26764

[25] - Flash/Flex vs Silverlight, uma comparação. – Mario Santos, (2009): http://rederia.net/2009/05/14/flashflex-vs-silverlight-uma-comparacao/

[26] - Flex Vs Ajax Vs Silverlight - http://vinaytechs.blogspot.com/2009/09/flex-vs-ajax-vs-silverlight.html

[27] - Ted Patrick: Inside Yahoo! Interview "AJAX vs. Flash" http://www.onflex.org/ted/2006/09/ajax-vs-flash-inside-yahoo_27.php

[28] - Pete Cashmore: "Flash, AJAX and Yahoo Maps: Does the Technology Matter?" http://mashable.com/2005/11/03/flash-ajax-and-yahoo-maps-does-the-technology-matter

[29] - James Ward: "AJAX and Flex Data Loading Benchmarks" http://www.jamesward.org/wordpress/2007/04/30/ajax-and-flex-data-loading-benchmarks

[30] - Jeffrey Hammond, Forrester Research: "Ajax or Flex?: How to Select RIA Technology" www.forrester.com/go?docid=40989