UNIVERSIDADE DE ÉVORA

Mestrado em Engenharia Informática

Miradouro Virtual®

Tiago Miguel da Conceição Bilou

orientador: Prof.ª Dr.ª Teresa Romão

Esta dissertação não inclui as críticas e sugestões feitas pelo juri.

26 de Março de 2009

UNIVERSIDADE DE ÉVORA

Mestrado em Engenharia Informática

Miradouro Virtual®

Tiago Miguel da Conceição Bilou

orientador: Prof.ª Dr.ª Teresa Romão

Esta dissertação não inclui as críticas e sugestões feitas pelo juri.

26 de Março de 2009

Resumo

Esta tese pretende descrever o desenvolvimento e arquitectura do software que constitui o Miradouro Virtual®, mais especificamente do componente referente à interface.

O Miradouro Virtual® é um dispositivo cujo propósito à semelhança dos tradicionais binóculos turísticos, é observar a paisagem, mas cuja interacção não está limitada à simples observação individual. Recorre à realidade aumentada para sobrepôr imagens geradas por computador a imagens reais, capturadas por um dispositivo para aquisição de imagem real (tipicamente uma câmara de vídeo), e mostra-as num ecrã *touchscreen*, permitindo deste modo, combinar elementos virtuais e multimédia com a paisagem real. A imagem final, composta, dá ao utilizador uma nova dimensão do espaço envolvente, permitindo-lhe explorar uma nova camada de informação não visível anteriormente. Sendo sensíveis à orientação do Miradouro Virtual®, os elementos virtuais e multimédia adaptam-se de acordo com os movimentos do dispositivo.

O Miradouro Virtual® é um produto composto por diversos elementos de hardware e software. O foco desta tese recai apenas nos componentes de software, mais especificamente na interface. Pretende dar a conhecer as limitações da versão anterior do software e mostrar as soluções encontradas que permitiram ultrapassar algumas dessas limitações.

Abstract

Virtual Sightseeing™

This thesis focuses on the design and development of the Virtual Sightseeing™ software, more specifically on the interface component.

The Virtual Sightseeing™ is a device similar to the traditional scenic viewers, that takes advantage of its generally known and popularity to build an innovative system. It works by using augmented reality to superimpose, in real-time, images generated by a computer onto a live stream captured by a vídeo camera and displaying them on a touchscreen display. It allows adding multimedia elements to the real scenery by composing them in the image that is presented to the user. The multimedia information and virtual elements that are displayed are sensitive to the orientation and position of the device. They change as the user manually changes the orientation of the device.

The Virtual Sightseeing™ is comprised of several hardware and software components. The focus of this thesis is on the software part, more specifically on the interface component. It intends to show the known limitations of the previous software version and how they where overcome in this new version.

Conteúdo

Capítulo 1

Introdução

Este primeiro capítulo apresenta uma breve história do Miradouro Virtual®, descreve a versão anterior, tecnologias e limitações, e apresenta os motivos para o trabalho desenvolvido nesta dissertação. Mais concretamente, na seccção 1.1 é apresentado o enquadramento do trabalho proposto e descrita a motivação que levou à realização do mesmo. Na secção 1.2 são descritos os objectivos definidos para o trabalho e na secção 1.3 é descrita a estrutura desta dissertação.

1.1 Enquadramento e Motivação

Iniciado em 2004 e com a primeira unidade instalada em setembro de 2005, no Castelo de Pinhel, o projecto do Miradouro Virtual® utiliza a realidade aumentada para sobrepôr elementos virtuais em ambientes reais, resultando num ambiente interactivo, onde se misturam elementos reais e virtuais para uma experiência única e diferente.

As versões anteriores do software para o Miradouro Virtual® foram desenvolvidas usando o Virtools. O Virtools é uma plataforma criada e desenvolvida pela Dassault Systemes [5], que possibilita uma rápida criação de aplicações com conteúdos ricos em 3D, suporte para elementos 2D e outros objectos de mídia como imagem e som. No núcleo do Virtools encontra-se o *behavioral engine*[1].

O Virtools disponibiliza uma grande colecção de *behaviors building blocks* (designados daqui por diante de *bbb*) que podem ser utilizados para criar quase todo o tipo de aplicações utilizando uma interface gráfica, i.e, sem escrever uma única linha de código. Este tipo de desenvolvimento gráfico parte de *bbb* simples, agrupando-os em *bbb* mais complexos, que por sua vez servirão de base para *bbb* ainda mais complexo. Este tipo de desenvolvimento permite a construção de aplicações complexas de forma sistemática, tornando o Virtools numa poderosa ferramenta para prototipagem rápida.

[1] Um *behavior* é simplesmente a descrição de como um dado elemento funciona num determinado ambiente.

Devido à complexidade e particularidades da aplicação do Miradouro Virtual®, não foi possível desenvolver o software de forma visual recorrendo à vasta colecção de *bbb* disponibilizada pelo Virtools. Assim sendo, construiu-se um *bbb* totalmente de raíz utilizando C++, para alojar toda a lógica da aplicação Miradouro Virtual®. Esta forma de desenvolvimento oferece ao programador liberdade total para programar todas as funcionalidades necessárias, no entanto possui uma grande desvantagem. Uma vez que o programador não está a tirar partido da interface gráfica, fica impedido de utilizar *bbb* existentes no desenvolvimento do seu *bbb*, o que faz com que muitas das funcionalidades disponíveis nos *bbb* não possam ser utilizadas. O programador vê-se assim obrigado a recorrendo às funções de baixo nível disponibilizadas pelo SDK para implementar manualmente todas as funcionalidades necessárias ao funcionamento da sua aplicação, mesmo as existentes em outros *bbb*. Esta forma de desenvolvimento levou a problemas de engenharia de software e elevou consideravelmente, quer o tempo de desenvolvimento inicial, quer o tempo de desenvolvimento de novas funcionalidades.

Ao nível da interface, e à semelhança de muitos dos ambientes de desenvolvimento 3D, o Virtools possui bastantes limitações ao nível 2D. A implementação da interface foi completamente feita à base de rectângulos (planos) e de uma "inteligente" escolha de texturas. Esta limitação técnica revelou-se numa grande restrição ao desenho da interface por parte do equipa de design, que juntamente com outras limitações ao nível de suporte de fontes, tornaram o desenho da interface do Miradouro Virtual® uma tarefa árdua cujo resultado final se encontrava bastante distante do concebido e pretendido pela equipa de design.

1.2 Objectivos

Esta nova versão do Miradouro Virtual® pretende resolver algumas das limitações e problemas detectados na versão anterior, bem como acrescentar novas funcionalidades, como por exemplo, uma camâra com zoom.

O objectivo é construir uma nova aplicação utilizando ferramentas capazes de diminuir a complexidade e tempo de desenvolvimento. Pretende-se que esta nova aplicação seja mais versátil e modular, isto é, um sistema baseado em componentes que executem funções independentemente uns dos outros e onde a comunicação entre componentes obedeça a um protocolo perfeitamente definido. Deste modo torna-se possivel substituir um componente por outro, sem que tal implique uma reformulação profunda no sistema.

Ao nível da interface pretende-se uma maior aproximação do conceito criado pela equipa de design no resultado final. Para tal a ferramenta escolhida deverá ser capaz de ultrapassar as limitações que o Virtools possui ao nível da interface, como obrigar a que todos elementos gráficos sejam construídos à custa de rectângulos, e as fortes restrições ao uso de fontes. A interface deverá simplificar a utilização e permitir uma maior liberdade, nas formas e comportamentos dos elementos gráficos. O posicionamento das etiquetas deverá ser mais flexível e o seu movimento ser menos rígido e

mais orgânico.

O trabalho desenvolvido nesta dissertação incidiu principalmente na concepção e implementação do componente da interface.

1.3 Estrutura da Tese

O capítulo seguinte apresenta uma revisão bibliográfica que aborda os conceitos relacionados com o trabalho apresentado nesta dissertação. O capítulo 3 descreve o sistema do Miradouro Virtual® e as ferramentas escolhidas para o desenvolviemento desta versão. No capítulo 4 é apresenta a arquitectura utilizada na aplicação, encontrando-se a implementação do componente da interface descrita no final do capitulo. No capítulo 5 são apresentados os resultados dos testes realizados à versão anterior e é descrito o estudo efectuado a esta versão. Por último, no capítulo 6, são apresentadas as conclusões do trabalho realizado.

Capítulo 2

Trabalho Relacionado

O presente capítulo apresenta alguns conceitos relacionados com o trabalho realizado, sendo dado maior relevância aos que contribuíram para o desenvolvimento do componente da interface.

Na secção 2.1 é apresentada uma definição do conceito de realidade aumentada, parte integrante da interface e do conceito do Miradouro Virtual®. Na secção 2.2 é introduzido o conceito de interfaces intuitivas que teve uma grande influência no desenho da nova interface do Miradouro Virtual®. A secção 2.3 aborda a temática do posicionamento de etiquetas de forma a evitar a sobreposição ou oclusão revelando alguns dos algoritmos capazes de o fazer, como os algoritmos de pesquisa descritos nas secções 2.3.2 e 2.3.3 ou os algoritmos baseados em forças da secção 2.3.4 ou 2.3.5. São também abordados na secção 2.4 alguns dos padrões de desenho utilizados do desenvolvimento da componente da interface.

2.1 Realidade Aumentada (A tecnologia por trás da ilusão)

A Realidade Aumentada (RA) pode ser definida como uma linha de pesquisa dentro da ciência da computação, que lida com a integração de elementos virtuais, criados por um computador, com o mundo real [10]. Um sistema de realidade aumentada, combina imagens reais, vistas pelo utilizador, com elementos virtuais gerados por computador, acrescentando informação adicional que permite ao utilizador ter uma visão mais abrangente do mundo que o rodeia.

A RA pode ser experiênciada de diversas formas, usando por exemplo um *Head Mounted Display* (HMD) ou um simples telemóvel, mas independentemente da forma como é experiênciada o propósito é sempre o mesmo: melhorar a percepção do utilizador em relação ao mundo que o rodeia. O objectivo máximo da RA é a criação de um sistema tão perfeito que o utilizador não consiga perceber a diferença entre o mundo real e a versão aumentada. Para o utilizador deste sistema seria como se estivesse a olhar apenas para o mundo real [11].

A RA elimina em grande parte a necessidade de aprendizagem, pelo facto, de trazer para o mundo real elementos virtuais, facilitando a aprendizagem do utilizador,

visto que parte do sistema se baseia no mundo real [14].

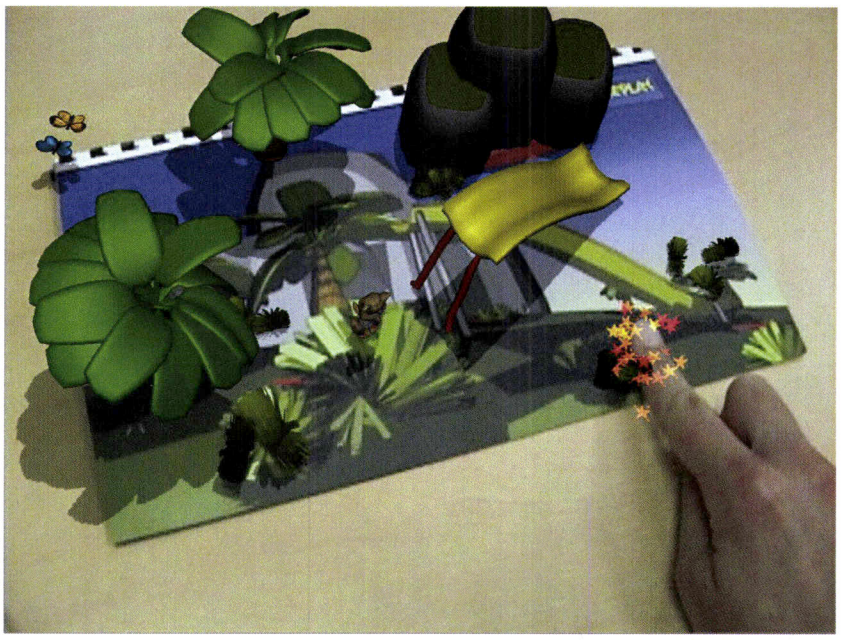

Figura 2.1: Exemplo do Interactive Magic book, um livro com realidade aumentada desenvolvido pela Total Immersion

A figura 2.1 mostra uma aplicação interactiva desenvolvida pela Total Immersion [6] chamada *Interactive Magic Book*, que recorre à RA para dar vida às personagens e transformar a experiência da leitura de um livro em algo diferente.

A história da Realidade Aumentada começa ainda na década de 60 pelas mãos de Ivan Sutherland, que utilizou um HMD para projectar gráficos 3D[1] sobre o ponto de vista do utilizador [1]. Só mais tarde na década de 90 é que surgiu o poder computacional e conhecimento necessário para dar lugar a uma verdadeira expansão da RA.

Para permitir a visualização e a interacção com os elementos virtuais é necessário software e hardware capaz de observar o ambiente real, analisar os dados e retirar informações sobre a localização e orientação do utilizador, permitindo determinar o seu campo de visão e os elementos do ambiente por ele observados, para posteriormente poder sobrepor à imagem real vista pelo utilizador os elementos virtuais [12].

Muitas das vezes confunde-se o conceito de realidade aumentada e de realidade virtual. Na realidade virtual todos os ambientes com os quais o utilizador interage são totalmente virtuais [9]. O utilizador não observa nada do mundo real. Tudo quanto vê é gerado artificialmente. A figura 2.2 ilustra o Contínuo de Milgram [16] que representa o intervalo existente entre a realidade e os ambientes totalmente virtuais. A realidade e os ambientes virtuais encontram-se em extremos opostos do intervalo,

[1]Devido às limitações tecnológicas da altura estes gráficos 3D eram apenas simples wireframes

no qual toda a região intermédia é descrita como "Mixed Reality". Milgram designou por "virtualidade aumentada" (Augmented Virtuality) a zona com predominância de elementos virtuais e alguns reais, e por RA, a região onde os elementos virtuais apenas complementam a informação real já existente.

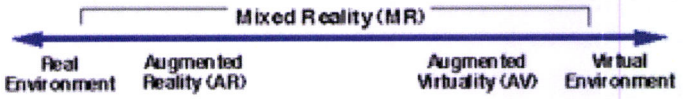

Figura 2.2: Virtuality Continuum de Milgram

Podemos considerar a RA como um meio-termo entre os ambientes totalmente virtuais e a realidade tal como a conhecemos.

Azuma [13] redefiniu em 1997, as principais características de um sistema de RA como tendo as seguintes características:

1. Deve combinar o real e o virtual
2. Ser interactivo e em tempo real
3. Ser capaz de registar e alinhar os elementos virtuais em três dimensões

Azuma [13] identificou seis classes de aplicações onde a RA estava a ser estudada e utilizada: visualização médica (Medical visualization), manutenção e reparos (Maintenance and repair), anotação (Annotation), planeamento de caminhos para robots (Robot path planning), entretenimento (Entertainment) e navegação e aquisição de objectivos em aviões militares (Military aircraft navigation and targeting).

Desde os primeiros protótipos do KARMA [2], passando pelo projecto ANTS [3] e mais recentemente o ARToolkit [4], a evolução nestes últimos anos na área da RA tem sido notável. Hoje em dia o número de aplicações que utiliza RA é bastante vasto e diverso e extende-se por vários domínios do conhecimento humano. A RA pode ser vista em simples aplicações que funcionam num dispositivo móvel como o *Wikitude AR Travel Guide*[2], ou que obrigam o utilizador a carregar um computador às costas como é o caso do projecto *ARQuake*[3].

2.2 Interfaces

Durante anos, os designers têm aperfeiçoado a arte de desenhar interfaces, melhorando muitas das boas práticas e reutilizando ideias. Hoje em dia existem diversas

[2]O Wikitude AR Travel guide é uma aplicação que utiliza um dispositivo móvel equipado com GPS para descobrir a localização do utilizador e mostrar conteudo proveniente da wikipedia ou Qype. É uma espécie de Miradouro Virtual® móvel

[3]O ARQuake é uma versão do famoso jogo Quake que utiliza RA. O jogador utiliza um HMD, computador móvel e GPS, deslocando-se pelo mundo real a lutar contra monstros virtuais.

aplicações e dispositivos considerados fáceis de usar. Muitos não o são. Desenhar uma boa interface é o objectivo principal de um designer, mas conseguir fazê-lo implica encontrar um equilíbrio entre uso, funcionalidade e boa aparência

Inicialmente as opções disponíveis para desenhar uma interface eram limitadas. O designer apenas podia contar com uma mão cheia de controlos, como caixas de texto, botões ou pequenos ícons. Hoje em dia existe uma imensa panóplia de componentes, no entanto toda esta liberdade e diversidade não fez com que criar um bom design se tornasse mais fácil, pelo contrário tornou-o ainda mais difícil. Esta liberdade deu origem a novas ideias e novas interfaces, e obrigou a redefinir a pegunta: O que é que caracteriza uma interface fácil de usar?

Faria sentido dizer que "As aplicações fáceis de usar foram desenhadas para ser intuitivas", no entanto a palavra "intuitivas" é um pouco enganadora. As coisas que nos são inatas vêm do conhecimento com o qual nascemos. Intuitivo é algo que se aprende. Um rato de computador, por exemplo, não é intuitivo para quem nunca viu nenhum porque não há nada inato ou intuitivo no cérebro humano que nos permita relacionar com ele. Mas depois de passar 2 minutos a utiliza-lo, este torna-se familiar e nunca mais se esquece. Jef Raskin [35] disse que a palavra "intuitivo" no contexto do software deveria ser substituída pela palavra "familiar".

"Familiar" não significa necessariamente que tudo à cerca da aplicação é idêntico a algo que já conhecemos. Desde que o utilizador reconheça certas partes e a relação entre elas seja fácil de entender, ele consegue aplicar o seu conhecimento prévio e compreender o que é novo. A mesma frase deveria ser então dita como "As aplicações fáceis de usar são desenhadas para serem familiares".

Jared Spool criou o conceito de *knowledge continuum*, para tentar detectar e melhorar a usabilidade das interfaces. Para explicar o conceito ele utiliza um exemplo bastante simples. Imaginemos uma parede onde vamos alinhar todos os utilizadores do nosso design. Vamos alinhá-los segundo algumas regras, do lado esquerdo colocamos os utilizadores que desconhecem por completo a interface, do lado direito colocamos os utilizadores que sabem algo acerca do design da mesma (é possível que apenas os designers e programadores consigam ocupar este lado), desta forma ficamos com os utilizadores organizados por conhecimento. A figura 2.3 representa a nossa parede.

Olhando para a parede, a distância a que os utilizadores se encontram do lado esquerdo representa o quanto eles conhecem da interface.

Podemos assim definir para cada utilizador o seu ponto de conhecimento actual (*current knowledge point*). Esse ponto representa o conhecimento que o utilizador possui quando utiliza a interface pela primeira vez. Existe outro ponto bastante importante para o designer, o ponto do conhecimento pretendido *(target knowledge point)*. Este ponto representa a quantidade de conhecimento que o utilizador deverá possuir para conseguir cumprir o seu objectivo.

Sempre que um utilizador tenta realizar uma tarefa específica, os pontos do conhecimento actual e pretendido são muito importantes para o estudo da interface. A figura 2.4 mostra a parede com um exemplo dos pontos de conhecimento actual e

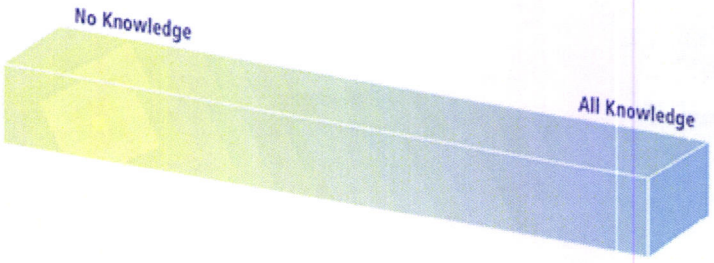

Figura 2.3: O conceito de *knowledge continuum* definido por Jared Spool, permite pensarmos no conhecimento de uma interface como um espaço continuo que vai desde o desconhecimento absoluto da interface ao limite de conhecer tudo o que há a saber sobre ela.

pretendido.

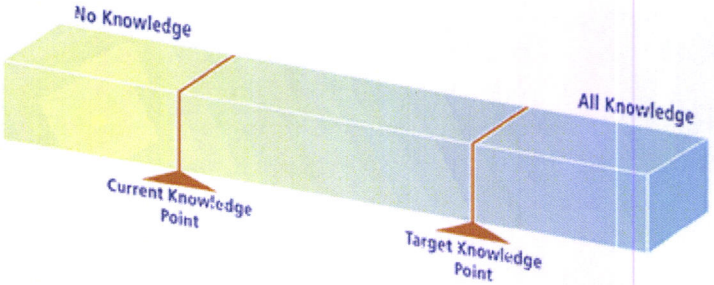

Figura 2.4: Sempre que um utilizador tenta realizar uma data tarefa, existem dois pontos importantes para o estudo do desenho da interface: o ponto de conhecimento actual e o ponto de conhecimento pretendido. O conhecimento actual representa o conhecimento que o utilizador possui a primeira vez que utiliza a aplicação e o conhecimento pretendido é o conhecimento que o utilizador necessita para conseguir realizar a tarefa.

A distância entre o ponto de conhecimento actual e o ponto de conhecimento pretendido designa-se de falha de conhecimento (*Knowledge Gap*) e encontra-se representado na figura 2.5.

É na falha de conhecimento que o design surge. No design da interface, não é necessário ter em conta os valores à esquerda do ponto de conhecimento actual, porque estes representam conhecimento que o utilizador já possui, da mesma forma que não é necessário considerar os valores à direita do ponto de conhecimento pretendido, dado que estes representam conhecimento que não será necessário o utilizador possuir para realizar a tarefa pretendida. É apenas necessário concentrar os esforços de design no espaço entre os pontos corrente e pretendido de conhecimento.

Os utilizadores são capazes de realizar a tarefa pretendida quando o conhecimento actual é igual ou superior ao conhecimento pretendido. Há duas formas de o tornar possível. Podemos treinar o utilizador, o que vai levar a um aumento do

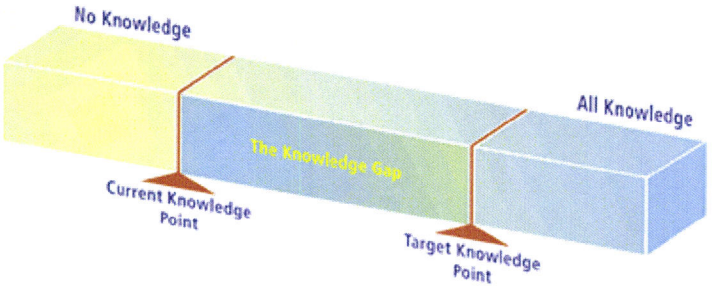

Figura 2.5: O espaço entre os pontos de conhecimento actual e conhecimento pretendido é chamado de falha de conhecimento (*Knowledge Gap*). Este intervalo do espaço de conhecimento é o mais importante e que mais contribui para o desenho de interfaces.

seu conhecimento actual da interface, até ele conseguir realizar a tarefa, ou podemos tentar reduzir o conhecimento necessário, tornando a interface mais simples, até que o conhecimento pretendido seja o mesmo que o que utilizador possui.

De facto, um bom design envolve ambas as formas. Os utilizadores são treinados e ao mesmo tempo o designer simplifica a interface, reduzindo efectivamente o comprimento da falha de conhecimento simultaneamente em ambos os sentidos.

Na pesquisa efectuada por Jared Spool [36] ele concluiu que existem duas situações onde os utilizadores dirão que uma interface lhes é intuitiva. Basta que esteja presente uma das duas condições descritas em baixo para que o utilizar diga que a interface é intuitiva. Quando nenhuma das condições é verificada, o utilizador dirá que a interface é pouco intuitiva.

1. Os pontos de conhecimento actual e pretendido são idênticos. Quando o utilizador interage com a interface ele já sabe tudo o que necessita para realizar as tarefas pretendidas.

2. Os pontos de conhecimento actual e pretendido estão separados, mas o utilizador não percebe que o design está a ajudar a minimizar a falha. O utilizador está a ser treinado, mas de uma forma que lhe parece natural.

São as pessoas que validam se uma interface é ou não intuitiva. Se o utilizador conseguir interagir com a interface imediatamente, sem curva de aprendizagem, ele dirá que esta é intuitiva.

2.3 Posicionamento de etiquetas

Como vimos anteriormente a RA mistura elementos virtuais com elementos reais. Como o mundo real é em três dimensões, muitos dos elementos virtuais utilizados pela RA também o são, como por exemplo os da figura 2.1. Porém em algumas aplicações, como é o caso do Miradouro Virtual®, a informação virtual é apresentada em duas dimensões, sendo um dos seus usos mais recorrentes, a colocação de etiquetas de

informação à cerca do mundo real. A decisão de como e onde colocar as etiquetas, que determina o layout da informação no espaço, é um problema de uma área de estudo da RA relativamente nova, denominada de *View Management* [21].

Colocar etiquetas em objectos gráficos é uma tarefa fundamental na concepção de muitas aplicações gráficas, mas assume uma maior importância no domínio da cartografia, onde é necessário colocar legendas nos mapas evitando sobreposições entre os símbolos cartográficos e outras legendas[4].

Quando falamos no domínio da RA, temos de ter em conta as características dinâmicas do sistema. Ao contrário do domínio da cartografia, onde os elementos são estáticos, na RA, os algoritmos para posicionamento de etiquetas têm de ter em conta diversos factores, como a visibilidade, posição e prioridade das etiquetas: demasiadas etiquetas, mal posicionadas ou pouco perceptíveis são consequências que diminuem as vantagens trazidas pela RA.

De seguida é enquadrada a problemática do posicionamento de etiquetas no domímio da cartografia sendo apresentados nas secções 2.3.2, 2.3.3, 2.3.4, 2.3.5 alguns algoritmos capazes de solucionar o problema do posicionamento de etiquetas.

2.3.1 Domínio da Cartografia

Tradicionalmente, quem estuda o problema do posicionamento de etiquetas examina o problema de colocar num mapa estático, um grande número de etiquetas, segundo as regras definidas por Imhof e Yoeli [26] :

1. As etiquetas deverão ter um tamanho legível
2. Não deverá ser ambígua a identificação correspondente a cada etiqueta
3. As etiquetas não se devem sobrepôr a outras etiquetas nem a outros elementos gráficos da interface.

Apesar de terem sido descobertas bastantes técnicas e métodos para automatizar a colocação de legendas em mapas, este processo ainda é feito de forma manual em muitas aplicações, sendo bastante moroso e trabalhoso. Assim encontrar um algoritmo capaz de efectuar automaticamente o posicionamento ideal das etiquetas assume uma grande importância.

Em cartografia, existem três tarefas diferentes de legendagem: Colocar legendas em áreas, como oceanos ou países, colocar legendas em linhas, como estradas ou rios e legendar pontos, como cidades [27]. Apesar de serem três tarefas bastante diferentes, as características necessárias para determinar o local ideal onde colocar a legenda são comuns. Sendo assim, podemos referir-nos apenas à legendagem de um ponto, sem perder generalidade em relação aos restantes.

O problema de legendar pontos é um problema de optimização, definido pelo seu espaço de pesquisa e função objectivo. Queremos identificar um algoritmo capaz de encontrar um bom elemento a partir do espaço de pesquisa.

[4]Quando falamos em legendas referimo-nos aos nomes dos continentes, oceanos, rios, paises ou cidades

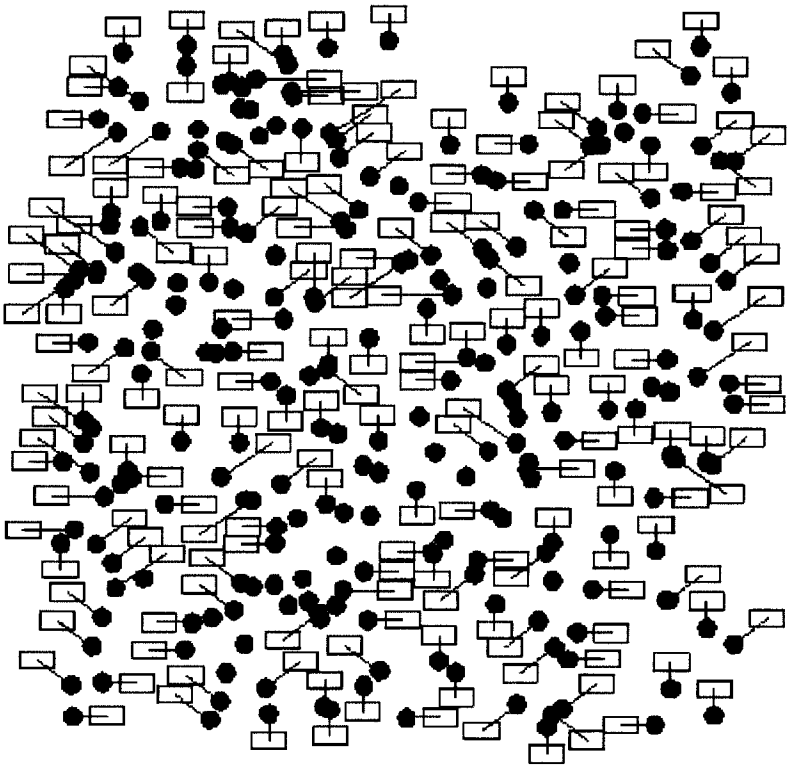

Figura 2.6: Exemplo do posicionamento de etiquetas no espaço evitando a sobreposição

Antes de explorar o espaço de pesquisa é necessário ter em conta a complexidade deste tipo de problemas. Estudos anteriores demonstraram que este tipo de problema é bastante difícil e a sua complexidade é NP-hard, isto é, o número de possíveis posições para as etiquetas cresce exponencialmente com o número de etiquetas. [28, 22]

Os algoritmos propostos para resolver o problema de legendagem podem dividir-se em dois grandes grupos: algoritmos que efectuam uma pesquisa exaustiva em todo o espaço de pesquisa procurando um solução aceitável ou óptima, e os que se focam num espaço de pesquisa local, mais restrito.

2.3.2 Algoritmos de pesquisa exaustiva

Os algoritmos de pesquisa exaustiva utilizam satisfação de restrições para chegar a uma solução e são geralmente categorizados com base na técnica de backtracking que utilizam, por exemplo, consideremos um algoritmo que percorre um conjunto de pontos e coloca uma legenda para cada ponto num local onde não se sobreponha a nenhuma outra legenda. Se, à medida que o algoritmo é executado, surge um ponto cuja legenda não é possível colocar (ou porque não há nenhuma posição cuja legenda

não se sobreponha a nenhuma outra ou porque todas as posições já foram tentadas), o algoritmo retorna ao último ponto cuja legenda foi colocada com sucesso e procura uma nova posição para essa legenda (backtracking). O algoritmo procede desta forma até encontrar uma solução ou até esgotar o espaço de pesquisa. Existem diversas modificações , que se podem fazer a este tipo de algoritmos, recorrendo a heurísticas para melhorar a sua performance. Apesar destes algoritmos se comportarem bastante bem para problemas com um espaço de pesquisa pequeno, à medida que o espaço de pesquisa aumenta e devido à sua natureza exponencial rapidamente as heurísticas se tornam inadequadas e o algoritmo se torna pouco eficiente. Independentemente da sofisticação das heurísticas utilizadas, uma pesquisa exaustiva, mesmo para um espaço de pesquisa de tamanho moderado torna-se pouco prático e eficiente.

2.3.3 Algoritmos "greedy"

Uma forma mais prática para pesquisar o espaço de pesquisa, evitando as estratégias de backtracking dos algoritmos de pesquisa exaustiva, são os chamados algoritmos "greedy". Ao limitar o espaço e contexto da pesquisa é possível construir algoritmos mais eficientes, que requerem menos memória e são, geralmente, muito rápidos a pesquisar o espaço.

É claro que estes algoritmos ao limitarem o espaço de pesquisa podem nunca chegar a encontrar a solução óptima, mas o princípio é de que seja possível encontrar uma solução mais rapidamente, que não sendo óptima é satisfatória. Ao invés de voltarem atrás na última solução encontrada, como fazem os algoritmos de pesquisa exaustiva sempre que surge um ponto que não consegue ser colocado, os algoritmos "greedy" utilizam outras estratégias dependendo do problema. Por exemplo, a legenda do ponto pode permanecer sobreposta caso seja aceitável, ou no caso do ponto se repetir várias vezes, pode mesmo não ser colocado. Este tipo de algoritmos apesar das suas limitações comporta-se de forma mais realista com o espaço de pesquisa utilizado neste tipo de problemas, embora a falta de backtracking gere soluções menos óptimas que os anteriores. Para que um algoritmo "greedy" consiga ser bem sucedido é essencial que existam heurísticas que permitam melhorar a pesquisa.

Alguns dos algoritmos "greedy" mais utilizados no posicionamento de legendas são:

- Discrete gradient descent
- Overlap vectors
- Mathematical programming
- Stochastic search (Simulated annealing)
- Genetic Algorithms

2.3.4 Algoritmos force-based

Simular formas que repelem todas os objectos adjacentes é uma estratégia utilizada em grafos de visualização, que repelem os nós vizinhos para maximizar a leitura[25], e pode ser aplicado até certo grau a problemas de posicionamento de etiquetas.

Algoritmos baseados em forças, *Force-based algorithms*, são uma classe de algoritmos utilizados geralmente para desenhar grafos de forma harmoniosa e equilibrada. O propósito destes algoritmos é posicionar os vértices de um grafo no espaço em duas ou três dimensões de modo a que todas as arestas fiquem com um comprimento semelhante e o número de cruzamentos seja o menor possível [30]. Um dos métodos mais usados, passa por atribuir forças aos vértices e arestas, assumindo que as arestas são molas (Lei de Hook[5]) e que os vértices são partículas carregadas electricamente (Lei de Coulomb[6]). Desta forma todo o grafo se comporta como se de um sistema físico se tratasse. As forças são aplicadas a todos os vértices, fazendo com que estes se afastem ou aproximem uns dos outros[7]. Este processo é repetido de forma iterativa até o sistema entrar em equilíbrio, isto é, quando não existir alteração na posição de nenhum dos elemento desde a iteração anterior. Nesse momento o equilíbrio é alcançado e o grafo é desenhado.

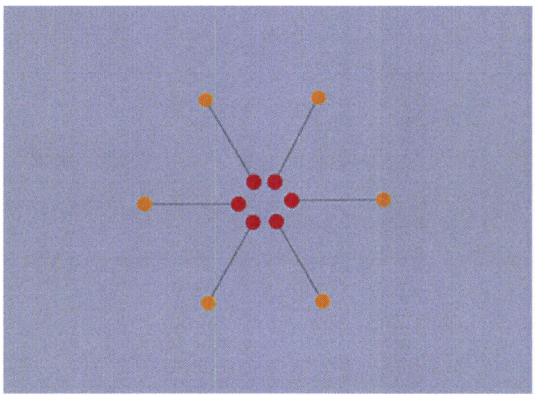

Figura 2.7: Exemplo da disposição dos vértices utilizando um algoritmos baseado em forças. Os pontos vermelhos são fixos. Sempre que o algoritmo alcança a posição de equilibrio os pontos laranja encontram-se nesta posição.

[5]A lei de Hooke é a lei da física relacionada com a elasticidade de corpos, que serve para calcular a deformação causada pela força exercida sobre um corpo, tal que a força é igual ao deslocamento da massa a partir do seu ponto de equilíbrio vezes a característica constante da mola ou do corpo que sofrerá a deformação: $F = k.\Delta l$ [33]

[6]A Lei de Coulomb foi descoberta pelo físico francês Charles Augustin de Coulomb, e trata-se do princípio fundamental da eletricidade. Em particular, diz-nos que o módulo da força entre duas cargas eléctricas puntiformes (q1 e q2) é directamente proporcional ao produto dos valores absolutos (módulos) das duas cargas e inversamente proporcional ao quadrado da distância r entre eles. Esta força pode ser atractiva ou repulsiva dependendo do sinal das cargas. É atractiva se as cargas tiverem sinais opostos. É repulsiva se as cargas tiverem o mesmo sinal.

[7]Também designado de forças de atracção repulsão entre elementos

A interpretação física do estado de equilibrio é que todas as forças mecânicas do sistema estão em equilíbrio. Este tipo de algoritmos pode envolver outras forças para além das referidas anteriormente, como por exemplo, forças magnéticas ou campos gravíticos.

O resultado visual deste tipo de algoritmos tende a ser harmonioso, pois simula as mesmas leis que existem na natureza e às quais estamos habituados. No caso dos grafos com molas e partículas carregadas electricamente, as arestas destes grafos tendem a ter um comprimento uniforme (devido às forças associadas às molas) e os vértices que não se encontram ligados a outros vértices tendem a ficar cada vez mais afastados (devido à repulsão eléctrica).

2.3.5 Motores de Física

Um motor de física é um programa que simula os modelos de física Newtoniano usando variáveis como massa, velocidade, fricção ou resistência ao vento. Pode simular e prever situações sob diferentes condições que se aproximam do que acontece na vida real ou num mundo de fantasia. São usados principalmente em simulações científicas ou em jogos de vídeo [31], mas devido à sua capacidade de detectar e resolver colisões podem ser utilizador para o posicionamento de etiquetas.

Geralmente os motores de física enquadram-se em duas classes: tempo real (*real-time*) e precisão (*high-precision*). Os motores de física de precisão necessitam de um grande poder computacional para efectuar simulações de física muito precisas, e são utilizados pelos cientistas ou em filmes de animação por computador. No caso dos jogos de vídeo, ou outras formas de computação interactiva, que necessitam de *feedback* mais rápido os motores de física utilizam modelos simplificados de cálculo e como consequência são menos precisos. Este tipo de motores de física é denominado de tempo real.

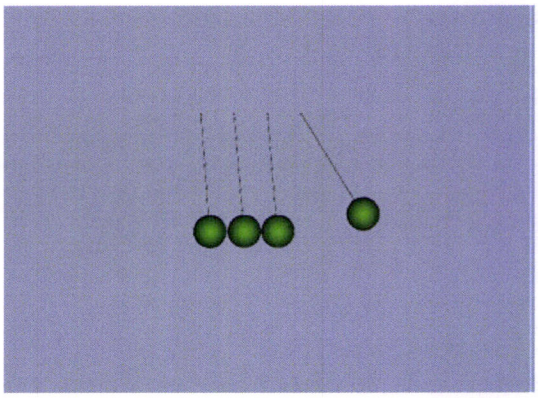

Figura 2.8: Exemplo de colisões de corpos usando um motor de física designado de APE. O motor de física está a simular o pêndulo de Newton

Os motores de física possuem dois componentes principais: um sistema de de-

tecção de colisões e um outro de simulação dinâmica responsável por calcular e resolver as forças aplicadas aos objectos da simulação.

Existem três grandes paradigmas para a simulação de física em corpos sólidos:

- Métodos de penalização, onde as interacções são modeladas como sistemas de molas e massa. Este tipo de motores é popular por permitir deformar ou utilizar física de corpos moles.
- Métodos baseados em restrições, onde são utilizadas equações de restrições para resolver e estimar as leis da física.
- Métodos baseados em impulsos, onde são utilizados impulsos aplicados às interacções dos objectos.

É possível combinar características dos paradigmas descritos em cima para criar métodos híbridos.

2.4 Padrões de desenho

As interfaces gráficas hoje em dia são parte natural e integrante de muitas das aplicações interactivas que utilizamos. Tornaram-se bastante comuns quer para os utilizadores finais, quer para os programadores que as desenvolvem. No entanto, como diz Martin Fowler, desenhar uma boa interface gráfica não é fácil [17].

Se desenhar software usando linguagens orientadas a objectos (OO) é difícil, desenhar software OO reutilizável é ainda mais difícil [19]. A primeira etapa é compreender e definir muito bem o problema, para que na fase de desenho da aplicação se possa olhar para o problema em mão e pensar qual a melhor forma de o solucionar. A arquitectura da aplicação deve ser estruturada e direccionada para o problema em questão, mas abstracta o suficiente para permitir resolver problemas futuros ou proceder facilmente a alterações nos requisitos. Muitos dos problemas encontrados ao desenvolver aplicações com base em linguagens OO são recorrentes. Ao longo dos tempos, estes problemas foram sendo resolvidos de uma forma inteligente e consistente por diversos programadores, verificando-se rapidamente que existiam umas opções melhores que outras para resolver determinados problemas. Os programadores mais experientes em arquitectura de software dirão que criar uma arquitectura reutilizável e flexível é difícil e impossível de se fazer "à primeira". Algumas das arquitecturas mais utilizadas hoje em dia foram conseguidas à custa de muita experimentação e de muitas iterações e adaptações [19].

Os padrões de desenho, *design patterns*, são soluções simples e elegantes para problemas específicos, que surgem inevitavelmente no desenho e implementação de aplicações desenvolvidas com linguagens OO. Os padrões retratam soluções que foram desenvolvidas, experimentadas e adaptadas ao longo do tempo. Embora não sejam soluções que as pessoas tendam a usar inicialmente, reflectem a forma como diversos programadores ultrapassaram obstáculos para tornar o seu software mais flexível e

reutilizável. Os padrões captam essas soluções e permitem aplicá-las de forma simples e sucinta.

De seguida são apresentados alguns padrões de desenho utizados no desenvolvimento de aplicações OO, e que foram utilizados no desenvolvimento da interface do Miradouro Virtual®.

2.4.1 Singleton

Muitas vezes, é necessário garantir que numa aplicação existe apenas uma, e uma só, instância de uma dada classe. Por exemplo, não faz sentido que exista mais que uma instâncias de uma classe responsável por carregar um ficheiro de XML, porque iríamos desperdiçar recursos ao manter várias cópias do mesmo ficheiro em memória. Como então, é que se consegue garantir que existe apenas uma dada instância de uma classe e que essa mesma instância está facilmente acessível a partir de qualquer parte do código? Qualquer programador diria que basta criar uma instância da classe e disponibiliza-la através de uma variável global. Desta forma permite-se o acesso fácil à instância, e ao usar a variável garante-se a sua unicidade. Este raciocínio faria sentido se o programador trabalhasse sozinho, mas quando está integrado numa equipa as coisas não funcionam bem assim. Ao utilizar classes com variáveis globais não é possível garantir que outros programadores não irão criar novas instâncias de classes que já tenham sido criadas por outros programadores noutras partes do código.

Uma melhor solução é deixar à responsabilidade da classe a criação e gestão das suas próprias instâncias. Ao passar essa responsabilidade para o domínio da classe, podemos interceptar os pedidos de criação de um novo objecto e assegurar que só é criada uma instância, providenciando um método para aceder à mesma. Isto é o padrão Singleton.

O Padrão Singleton deve garantir que apenas existe uma instância da classe em cada instânte. A maneira mais comum de o fazer é declarar o método construtor da classe como privado. Se o programador tentar instanciar a classe directamente o compilador retornará um erro.

Em algumas linguagens OO, particularmente no caso do Actionscript 3.0 que foi a linguagem utilizada no desenvolvimento da interface do Miradouro Virtual®, não é possível declarar um construtor como privado. Sendo assim foi necessário recorrer a outro método para garantir a singularidade da instância. O método utilizado passa por declarar uma classe interna e passar uma referência dessa classe como argumento no construtor da classe onde se pretende aplicar o padrão Singleton. Uma vez que a classe é interna não é possível aceder-lhe fora do âmbito da classe onde se pretende implementar o padrão Singleton, tornando impossível instanciar esta classe usando o seu construtor.

Depois de limitar o acesso ao construtor da classe, é criada uma variável de classe que irá guardar uma referência de si própria, e disponibilizado um método

público para permitir instanciar objectos.[8] A primeira vez que o método de criação é chamado, este verifica o estado da variável, se estiver vazia, então ainda não existe nenhuma instância da classe. A instância é criada, retornada, e a variável é alterada. Após a criação da primeira instância, qualquer pedido posterior de criação será negado, sendo retornada a instância guardada na variável ao invés de uma nova instância. Este método assegura que a classe apenas é instanciada na altura da sua primeira utilização.

Depois de examinar a estrutura da classe que implementa o padrão Singleton poderemos questionar-nos: - "Porque não transformar todos os métodos e propriedades da classe em métodos e propriedades estáticas? Porque razão precisamos de criar uma instância dessa classe?" Há algumas razões pelas quais um Singleton pode ser melhor opção que uma classe estática [20].

A primeira razão está relacionada com a herança. Em algumas linguagens OO, particularmente no caso do ActionScript 3.0, não é possível herdar propriedades ou métodos estáticos. Se as funcionalidades da classe estiverem implementadas em métodos estáticos não é possível estender a classe e herdar as funcionalidades.

A segunda razão para utilizar Singletons é a possibilidade de alterar a classe para permitir criar mais que uma instância. Existe um padrão menos conhecido chamado Multiton, bastante similar ao Singleton, mas que permite gerir mais que uma instância da classe. Se definirmos inicialmente a classe como Singleton podemos facilmente transformá-la num Multiton caso seja necessário utilizar mais que uma instância. Por exemplo, se a nossa aplicação utilizar um Singleton para estabelecer uma ligação a um servidor poderemos mais tarde transformá-la num Multiton para usar uma pool de ligações.

A terceira razão é que os Singletons utilizam o conceito de *lazy instantiation*, isto é, o objecto é apenas criado quando se faz a primeira chamada à classe. As classes estáticas inicializam todos os recursos na altura de construção, o que pode levar a um desperdício de recursos e a aumentar o tempo de inicialização da aplicação.

2.4.2 Observer

O objectivo do padrão Observer, também chamado de *publish-subscribe*, é definir uma relação de dependência 1-n entre objectos, de forma a que quando o estado de um objecto é alterado, todas as classes dependentes são notificadas e actualizadas automaticamente.

Um dos problemas mais comuns quando se constrói a arquitectura de um programa e se dividem as responsabilidades por diversas entidades é conseguir manter a consistência entre elas. Não queremos obter essa consistência entre classes criando ligações fortes entre elas, pois isso reduz a sua flexibilidade e reutilização.

Por exemplo, muitas *frameworks* dedicadas a interfaces gráficas separam a camada de dados da forma com estes são apresentados graficamente. Desta forma, as classes de dados e as classes visuais podem ser utilizadas de forma independente

[8] Este método é geralmente chamado de `getInstance()`

ou em conjunto. Imaginemos uma interface para mostrar as horas com uma parte analógica e outra digital. Tanto a classe que representa a parte analógica, como a classe que representa a parte digital mostram de forma diferente a informação contida na classe dos dados, a hora. Ambas as classes acedem à mesma classe de dados. A classe da interface digital não sabe que existe uma interface analógica e a classe da interface analógica não sabe que existe uma interface digital. Isto permite reutilizar uma independentemente da outra. No entanto parece que sabem uma da outra, porque quando a hora muda tanto a interface analógica como a digital são alteradas simultaneamente.

O ponto central do padrão Observer é possuir o estado da aplicação centralizado num local, sempre que o estado é alterado a informação é enviada para todas as classes dependentes. Esta relação 1-n permite ao programador criar classes com ligações fracas, mas ao mesmo tempo manter a consistência da informação entre elas. Partilhar a informação desta forma entre diversas classes é eficiente e dá espaço para expansão.

Este padrão é utilizado no padrão Model View Controller (MVC), que iremos ver na secção 2.4.3, para comunicação entre as *views* e o *model*. O *model* é tratado como a classe de dados e as *views* como classes dependentes.

Ao nível de implementação do padrão Observer, a classe de dados chama o método *update()* das diversas classes dependentes sempre que existe uma alteração ao seu estado, para que estas se actualizem. Muitos dos programadores têm tendência para passar informação adicional sobre o estado como parâmetros do método update. A quantidade de informação transmitida desta forma pode variar muito. Existem dois modelos que definem a quantidade de informação: *push* e *pull*. Num dos extremos, existe o que designamos de modelo push. Segundo este modelo, a classe de dados envia informação detalhada sobre o novo estado, quer as classes o queiram quer não. No outro extremo temos o que designamos de modelo *pull*. Segundo este modelo, a classe de dados apenas notifica as classes dependentes de que existem alterações ao estado, e são as classes dependentes que pedem posteriormente informações mais detalhadas. Este modelo, dá ênfase à ignorância das classes dependentes, enquanto o modelo push assume que a classe de dados sabe algo acerca das necessidades das classes dependentes. O modelo *push* pode tornar as classes dependentes menos genéricas e desta forma menos reutilizáveis, porque a classe de dados assume certas informações acerca das classes dependentes que podem não ser sempre verdade. Por outro lado o modelo *pull* pode ser menos eficiente, porque obriga as classes dependentes a verificarem o que foi alterado do lado da classe de dados sem a ajuda da mesma. Na implementação da interface do Miradouro Virtual® foi utilizado o modelo *push*.

2.4.3 MVC - Model View Controller

Há alguns anos atrás, praticamente todos os programas de computador tinham uma interface de utilização muito limitada. Na década de 70 a interface mais comum para interagir com os programas era através da linha de comandos. Hoje em dia praticamente todo os programas possuem uma interface gráfica bastante rica e completa,

que utiliza janelas e outros elementos gráfico e recorre ao uso do rato e outros dispositivos como forma de interacção. Pensado para responder à necessidade de criar uma interface gráfica para o Smalltalk-80, os princípios do *Model-View-Controller* (MVC) têm-se mantido até hoje e continuam a ser uma das melhores soluções para ultrapassar os desafios da criação de interfaces gráficas.

As aplicações consistem geralmente numa parte lógica e noutra de dados. Apesar desta divisão de responsabilidades fazer sentido e facilitar a reutilização das classes, muitos programadores não o fazem e combinam a parte gráfica, lógica e dados num só objecto monolítico em vez de vários a trabalhar em conjunto. A experiência diz que quando a parte gráfica e os dados estão englobados no mesmo objecto podem surgir alguns problemas:

- Ao englobar os dados na mesma classe responsável pela parte gráfica estamos a limitar o acesso aos dados. Por exemplo, se um objecto define um formulário e guarda os dados do formulário na mesma classe é muito difícil retirar os dados da classe para os enviar para o servidor. As opções nesse caso passam por colocar a responsabilidade de comunicação com o servidor na mesma classe ou disponibilizar os dados a outras classes recorrendo a interfaces. Qualquer uma das opções tem problemas e estruturas bastante rígidas que impedem o reaproveitamento das classes.

- É difícil alterar a parte gráfica mantendo os mesmos dados. Se a componente gráfica e os dados fazem parte da mesma classe, criar um novo elemento gráfico implica, para além da nova interface gráfica, transferir todos os dados da classe antiga para a nova. A alteração da parte gráfica é algo natural na evolução das aplicações.

- Implementar várias interfaces gráficas para o mesmo conjunto de dados é difícil. Se quisermos representar os dados de duas ou mais maneiras diferentes, teremos de replicar os dados pelas várias classes. Se os dados sofressem alterações seria necessário reflectir essas alterações nas diversas classes.

Os problemas descritos nos pontos anteriores devem-se ao facto da parte gráfica e os dados fazerem parte do mesmo objecto. O padrão MVC apresenta uma solução que permite a vários objectos trabalharem em conjunto, partilhando os mesmo dados. Este padrão permite alterações independentemente aos dados e output dos mesmos, criando aplicações mais flexíveis e promovendo a reutilização dos componentes. Como o nome implica consiste em três elementos:

- *Model* - Concentra em si os dados e define a lógica da aplicação para guardar e gerir o estado da mesma.

- *View* - Reflecte, de forma visual, o estado a aplicação. É a classe de output, representando todos os elementos gráficos.

- *Controller* - Responsável pela gestão dos inputs que vão alterar o estado aplicação.

O princípio fundamental do MVC, que marcou e influenciou outros padrões, é o que podemos chamar de separação da apresentação [17]. Este princípio promove a separação de responsabilidades de cada um dos elementos. Cada elemento sabe o que tem para fazer e as suas responsabilidades não se sobrepõem. O padrão MVC é um pouco diferente dos anteriores uma vez que recorre a diversos padrões para cumprir os seus objectivos. É classificado por uns como uma arquitectura de aplicações ou por outros como um padrão composto [20].

O MVC cria uma separação efectiva entre a parte gráfica e os dados, recorrendo ao padrão Observer para a comunicação entre eles. Sempre que o estado do *model* muda, este notifica a *view*, e esta tem oportunidade de se actualizar com base nos novos dados. Isto permite utilizar várias *views* para o mesmo *model*.

Capítulo 3

O Miradouro Virtual®

Este capítulo descreve a nova versão de software do Miradouro Virtual®. A secção 3.1 consiste numa breve apresentação do sistema e componentes de hardware do Miradouro Virtual®. A seccção 3.2 descreve as ferramentas escolhidas para a nova versão, indicando de que forma estas permitem ultrapassar algumas das limitações existentes na versão anterior.

3.1 O Sistema do Miradouro Virtual®

O sistema do Miradouro Virtual® é composto por diversos componentes de hardware e software. Na imagem 3.1 está representado um modelo 3D da unidade do Miradouro Virtual® onde podemos ver o esqueleto exterior em fibra de vidro, ecrã touchscreen, os punhos e a base giratória.

Figura 3.1: Representação de um modelo 3D do Miradouro Virtual®

O Miradouro Virtual® utiliza realidade aumentada para sobrepôr elementos virtuais na imagem real, o que permite numa experiência bastante diferente dos tradicionais binóculos turísticos. Nas figuras 3.2 e 3.3 podemos ver como o Miradouro Virtual® utiliza a realidade aumentada para melhorar a percepção do utilizador em relação ao que o rodeia.

Figura 3.2: Imagem real capturada pela câmara de vídeo

Sempre que o utilizador roda o dispositivo, a informação virtual presente no ecrã é alterada de acordo. A orientação do dispositivo é dada por um conjunto de encoders que permitem o correcto mapeamento dos elementos virtuais com o mundo real.

Um *encoder* é um dispositivo electro-mecânico usado para converter a posição angular de um eixo num código analógico ou digital. Geralmente são classificados em dois grupos: absolutos[1] e incrementais ou relativos [2].

O miradouro utiliza dois *encoders* absolutos, um horizontal para calcular a orientação do aparelho e um vertical para calcular a inclinação da câmara de vídeo. O funcionamento do encoder horizontal é directo, isto é, ao rodarmos o miradouro sobre o seu pé, estamos ao mesmo tempo a mover mecanicamente o *encoder*. No caso do *encoder* da câmara, este é movido com o girar do punho direito, mas não é feito de forma directa. O punho tem um inclinómetro que faz accionar um pequeno servo,

[1]Com os encoders absolutos o valor obtido é sempre único. Mesmo que se ligue ou desligue o valor obtido não é alterado.

[2]Neste tipo de encoders, quando este é ligado define o seu valor como valor base. Os restantes valores são obtidos a partir de incrementos, positivos ou negativos, a partir do valor base.

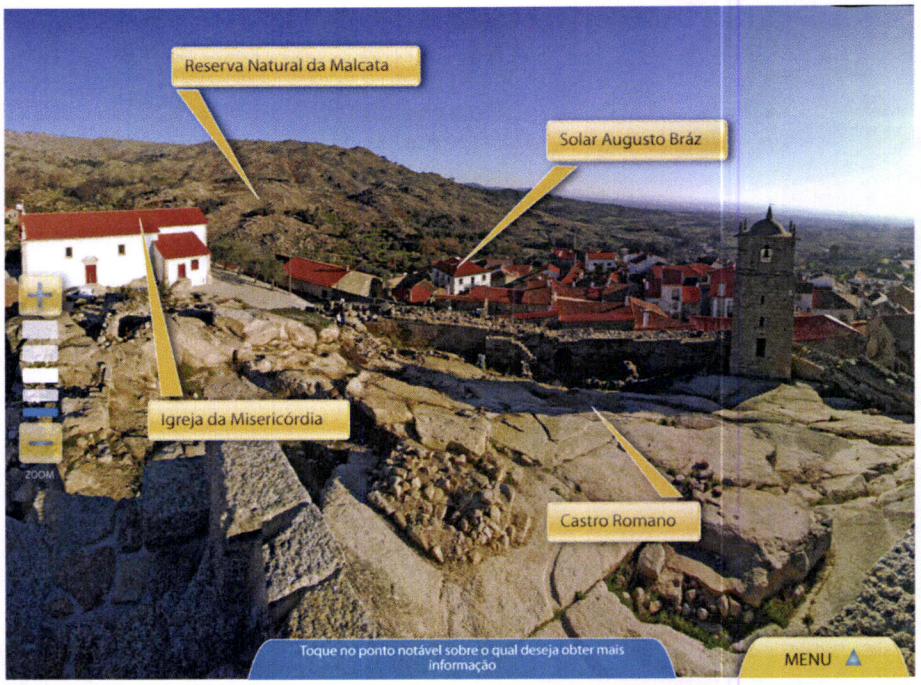

Figura 3.3: Ecrã principal do miradouro a utilizar realidade aumentada onde se pode ver a sobreposição de gráficos gerados por computador à imagem real capturada pela câmara.

que por sua vez roda o *encoder*. Esta solução permite alterar o ângulo de rotação da câmara, criando restrições ou dando maior liberdade de movimentos, sem haver necessidade de alterar a mecânica do punho.

Como foi referido no capítulo 1, o objectivo foi o de desenvolver uma nova versão da aplicação para substituir a versão anterior de forma a resolver os problemas descritos na secção 1.2. O primeiro passo foi encontrar ferramentas capazes de substituir o Virtools, de forma a conseguirem ultrapassar-se as limitações da versão anterior e obter um resultado mais satisfatório a nível gráfico.

3.2 Ferramentas

A escolha das ferramentas recaíu no Adobe Flash [7] para o desenho e implementação da interface, e no YVision[3] para o desenvolvimento da aplicação e consequente integração da interface.

O Flash teve a sua origem em 1993 pelas mão de Jonathan Gay [8]. Nasceu como ferramenta de desenho vectorial, mas rapidamente evoluiu para uma ferramenta completa, capaz de se integrar com diversas plataformas e linguagens. Apesar de ter

[3]O YVision é uma plataforma proprietária da YDreams, usada para desenvolvimento interno de aplicações interactivas

sido criado como uma ferramenta de animação, nos últimos anos evoluiu ao ponto de poder ser usado como plataforma de desenvolvimento de aplicações interactivas.

O YVision é uma *framework* de programação, criada para prototipagem rápida e desenvolvimento de aplicações multimédia *data-driven*. A *framework* integra diversas tecnologias, como um motor de *render 3D*, física de corpos rígidos, processamento de imagens e visão, necessárias para responder aos diversos desafios e problemáticas que surgem durante o desenvolvimento de aplicações interactivas.

3.2.1 Flash

A escolha do Flash como ferramenta para a implementação da interface veio resolver um grande número de problemas encontrados na versão anterior, como as fontes ou a obrigatoriedade de utilizar rectângulos em todos os elementos gráficos da interface, ao mesmo tempo que permitiu ao designer ter uma maior liberdade criativa e aumentar o manancial de ideias a utilizar na interface. O desenho da interface do Miradouro Virtual® tentou ao máximo minimizar a curva de aprendizagem do utilizador, dado que uma percentagem elevada dos utilizadores do Miradouro Virtual® irá utilizá-lo apenas uma vez, e eventualmente durante um periodo de tempo relativamente curto, pelo que se pretende que a interaccção seja o mais gratificante possível.

Quando nos referimos ao Flash estamos a referir-nos à ferramenta de desenho denominada de Adobe Flash Profissional e à linguagem de programação designada de Actionscript. A ferramenta de desenho é utilizada principalmente pelo designer, para desenhar de forma visual toda a interface da aplicação e o actionscript é utilizado pelo programador para criar a lógica da aplicação e programar o comportamentos dos objectos visuais. Tanto a ferramenta de desenho como o ambiente de programação geram ficheiros SWF que são executado pela *ActionScript Virtual Machine* (AVM) integrada no *flash player*.

Apesar do Flash ter surgido como ferramenta de animação, ao longo dos anos foram sendo acrescentadas funcionalidades para permitir automatizar determinadas tarefas recorrendo ao uso de pequenos scripts. Nos últimos anos esta linguagem de scripts denominada de actionscript evoluiu para uma linguagem de programação OO completa. A última versão denominada de ActionScript 3.0 é uma poderosa e dinâmica linguagem de programação orientada a objectos baseada no ECMAScript[4] que permite programar de forma eficiente desde pequenas animações a interfaces interactivas completas.

Sendo o Flash uma ferramenta 2D de desenho vectorial, foi de imediato eliminada uma das maiores limitações impostas pelo Virtools: o facto de todos os elementos terem de ser rectangulares. Usando o Flash, o designer tem a liberdade total de desenhar o que quer e o resultado final é em tudo idêntico ao desenhado, não havendo perdas ou alterações no processo. Não é necessário criar ficheiros intermédios, ou

[4]O ECMAScript é uma linguagem de script tornada standard pelo Ecma International, organização responsável pela criação de standards.

outros artefactos visuais pois a aplicação utiliza directamente os elementos gráficos desenhados pelo designer.

Relativamente às limitações que o Virtools possuía ao nível de fontes, também essas foram ultrapassadas graças ao uso do Flash. O Virtools possuía um conjunto de fontes nativas à aplicação que funcionavam relativamente bem, mas as primitivas disponíveis para trabalhar com as fontes eram muito básicas, factor que era bastante limitativo para o desenho da interface. Caso o designer optasse por utilizar uma fonte que não pertencesse ao conjunto de fontes nativas do Virtools, surgiam enumeras restrições ao seu uso.

Para conseguir utilizar uma fonte não nativa no Virtools, era necessário converter os diversos caracteres num bitmap. Seguidamente era necessário criar um bitmap individual para cada caracter de forma a estes puderem ser utilizados como texturas num plano do mundo 3D. Sempre que o designer decidisse alterar o tamanho ou estilo da fonte era necessário repetir todo o processo. Esta técnica era bastante morosa, trabalhosa e limitativa pois impedia o uso de *antialiasing* e fazia com que todas as fontes ficassem com um espaçamento idêntico (monospaced).

O Flash, por sua vez, permite o uso de qualquer fonte disponível no sistema, sem que esta tenha que passar por algum tipo de processamento prévio. O motor de fontes do Flash permite que estas sejam desenhadas de forma vectorial e as suas ferramentas de desenho de fontes dão toda a liberdade necessária ao designer.

Capítulo 4

Implementação

Neste capítulo é descrita a implementação do software do Miradouro Virtual®. São apresentados os diversos componentes que compõem a solução final do software e é descrito em maior pormenor a componente da interface. A seccção 4.1 descreve a arquitectura de software da nova versão, especificando os elementos que a compõem. A implementação detalhada da interface encontra-se descrita na secção 4.2

4.1 Arquitectura do software

Devido à diversidade de funcionalidades e tecnologias utilizadas ao nível do software a aplicação foi dividida em 4 componentes, nomeadamente: Backoffice, Proxy, Engine e Interface.

O engine é a componente responsável por todo o mapeamento 3D dos pontos no mundo virtual bem com toda a parte de cálculo e conversão de coordenadas. A interface é o componente responsável por mostrar toda a informação ao utilizador e permitir a interação com a mesma.

Como podemos ver na figura 4.1 o proxy assume um papel central na aplicação quer ao nível de comunicação quer ao nível de configuração dos clientes. Esta arquitectura centralizada permite que os componentes não tenham conhecimento dos restantes componentes do sistema, o que o torna mais modular e flexível.

O proxy é bastante versátil no que respeita à comunicação, comunicando com o backoffice através de *webservices*, com o hardware via porta série e com o engine e interface via sockets, encarregando-se de enviar, receber e reencaminhar mensagens entre os diversos componentes. Os componentes mais activos durante a execução da aplicação são o proxy o engine e a interface, sendo que o backoffice assume um papel importante mas pontual, no início da aplicação e na altura de calibração dos pontos. O proxy antes de iniciar os restantes componentes, estabelece uma ligação ao backoffice para sincronizar os conteúdos da aplicação, uma vez que estes estão guardados no backoffice.

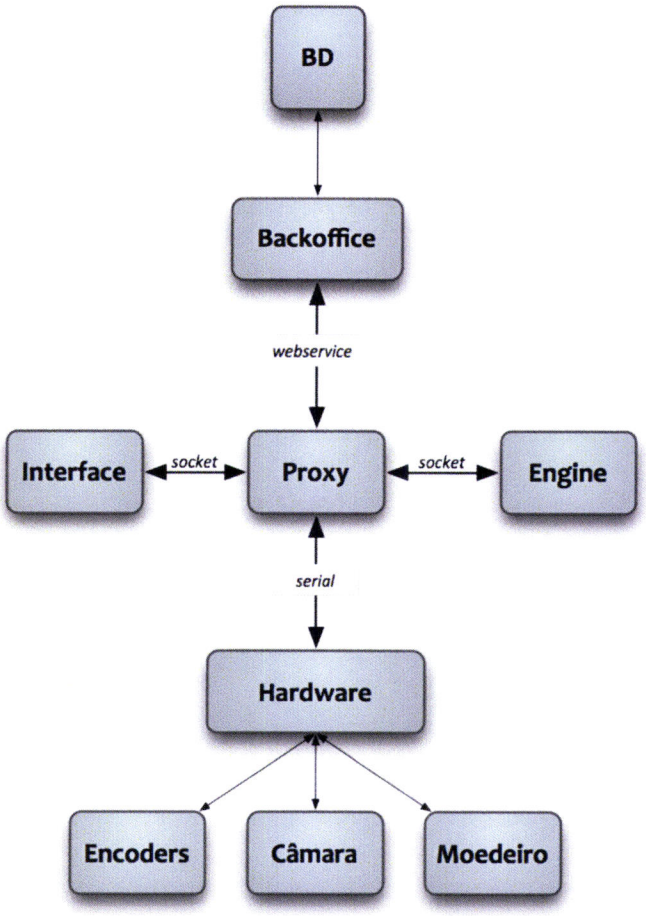

Figura 4.1: Esquema de componentes de software.
O proxy assume um papel central a nível de comunicação e configuração dos diversos componentes. As ligações entre os diversos componentes são sinalizadas através das setas e a legenda representa o tipo de ligação entre eles.

Para a comunicação entre proxy, engine e interface foi definido um protocolo, no formato XML para a troca de mensagens via socket. O protocolo completo é composto por 27 mensagens, sendo que uma das mensagens mais importantes é o DataFrame. O DataFrame é a mensagem que mais circula pelo sistema, cerca de 15 vezes por segundo, garantindo que a interface mostra as etiquetas correctas nas coordenadas certas.

Uma das tarefas mais importantes do proxy é o envio dos valores dos encoders ao engine. O engine com base nos dados dos encoders enviados pelo proxy actualiza a posição da sua câmara virtual, calcula os pontos visíveis pela câmara, converte as coordenadas esféricas em coordenadas de ecrã e cria um DataFrame com essa informação para enviar à interface.

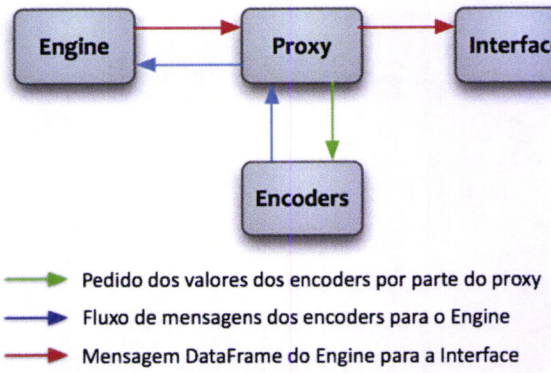

Figura 4.2: Esquema exemplificativo da criação e envio de um DataFrame.
O proxy faz um pedido aos encoders que retornam os seus valores actuais. De seguida o proxy envia uma mensagem ao Engine que actualiza a posição da câmara virtual e calcula os pontos visíveis através da câmara. De seguida cria e envia um DataFrame para a interface.

```
<DataFrame>
  <PointList>
    <Point id="10" x="20" y="30" visible="1" screenState="onscreen" />
    <Point id="40" x="50" y="60" visible="0" screenState="left" />
  </PointList>
  <NearestLeft>2</NearestLeft>
  <NearestRight>3</NearestRight>
  <NearestUp>4</NearestUp>
  <NearestDown>1</NearestDown>
  <FieldOfView>32.1</FieldOfView>
  <MotionDetected>0</MotionDetected>
  <ZoomLevelNumerator>3</ZoomLevelNumerator>
  <ZoomLevelDenominator>7</ZoomLevelDenominator>
</DataFrame>
```

Em cima temos descrita a mensagem Dataframe do protocolo. Podemos ver que o nó <PointList> contém a lista de pontos visíveis pela câmara virtual no mundo 3D, já com as suas coordenadas convertidas para coordenadas de ecrã em 2D. Com base nesta lista, a interface cria uma etiqueta para cada nó *<point>* e coloca-a no ecrã, nas coordenadas enviadas pelo engine, ficando o seu posicionamento a cargo do algoritmo de posicionamento descrito na secção 4.2.6. Esta mensagem é indispensável para o funcionamento da aplicação, ao percorrer todo o sistema, desde o hardware ao proxy, engine e interface.

Apesar do foco deste trabalho ser a intreface, é importante conhecer os restantes componentes, indispensáveis ao funcionamento da interface. De seguida é apresentada uma breve descrição dos vários componentes, dando maior destaque à interface na secção 4.2.

4.1.1 Backoffice

O backoffice, também denominado de gestor de conteúdos, é a aplicação encarregue de gerir os conteúdos associados ao projecto que serão depois apresentados na interface do Miradouro Virtual®.

Desenvolvido em php e utilizando o PostgreSQL como SGBD permite uma gestão detalhada do projecto e dos seus conteúdos e utilizadores. Toda a configuração do projecto pode ser efectuada através de um browser multimédia, como o Firefox ou o Internet Explorer, uma vez que o backoffice disponibiliza um *frontend* em php, html e javascript como pode ser visto na figura 4.3. O acesso aos dados por parte da aplicação, mais especificamente do componente proxy (secção 4.1.2), é efectuada recorrendo a webservices que permitem uma comunicação bidireccional para download ou upload dos conteúdos e informação contida na base de dados.

Ao nível de configuração, o backoffice possui dois perfis de utilização: Gestão de conteúdos e Administração.

1. Gestão de conteúdos: permite gerir os conteúdos que estão associados ao projecto, que serão depois apresentados na interface do Miradouro. É acedido pelos utilizadores com o perfil "Utilizador". Neste perfil é possível criar, apagar ou modificar pontos de interesse (POI) bem como associar-lhes recursos de midia como vídeos, fotos e sons.

2. Administração: permite gerir as definições do projecto e respectivos utilizadores. É acedido pelos utilizadores com o perfil "Administrador". Neste perfil é possível criar, apagar ou modificar projectos. Permite configurar também quais os idiomas disponíveis para o projecto.

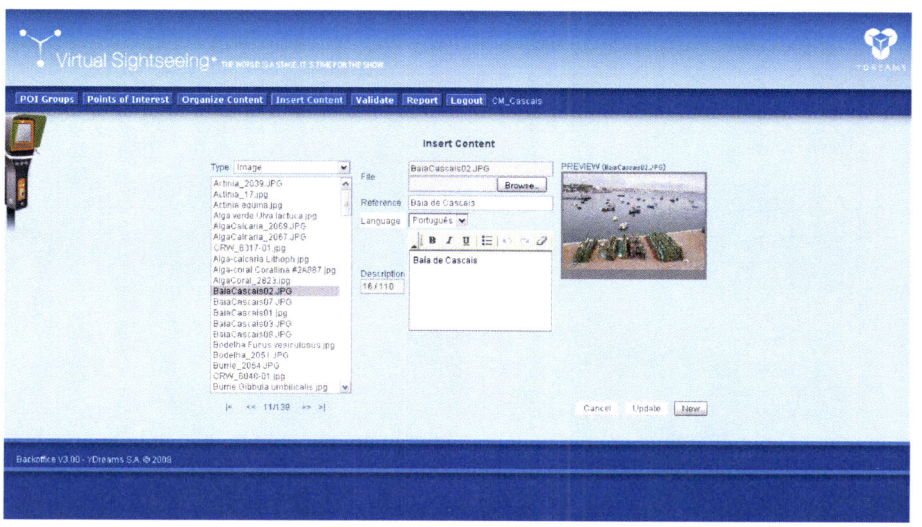

Figura 4.3: Frontend em html do backoffice. Página de configuração de conteúdos segundo o perfil "utilizador".

4.1.2 Proxy

Designado proxy, este componente de software é o ponto de entrada na aplicação e tem um papel central no seu funcionamento.

O proxy é responsável por garantir que todo o sistema se encontra correctamente configurado, isto é, que o hardware está a funcionar, que todas as partes envolvidas comunicam umas com as outras, e que tudo o que é necessário para o correcto funcionamento da aplicação está presente na altura da execução. Na versão anterior o conceito de proxy não existia, estando as suas responsabilidades distribuídas por diversos *bbb* dentro do Virtools. Nesta nova versão do software o proxy encontra-se explicitamente definido como um componente da aplicação e as suas responsabilidades bem delineadas.

Desenvolvido em C# e usando uma arquitectura por componentes com inversão de controlo, o proxy é um sistema modular que utiliza serviços para a comunicação e configuração do hardware (controlador da câmara, encoders, etc), sincronismo de projectos com o backoffice, configuração e comunicação com o engine e a interface.

No que diz respeito ao hardware, o proxy possui serviços específicos (módulos) que inicializam, configuram e testam cada componente. A comunicação com os dispositivos é geralmente feita através de portas série via usb.

No que respeita à comunicação, o proxy comporta-se como um hub[1], centralizando em si todos os fluxos de comunicação e redireccionando mensagens de um componente para outro, disponibilizando-lhes desta forma toda a informação necessária. É responsabilidade do proxy fazer download de todos os conteúdos e recursos necessários para o projecto, tais como informação textual (XML), imagens, vídeos, sons ou animações a partir do backoffice. Tanto a interface como o engine vão necessitar de acesso a partes desta informação, por isso o proxy envia-lhes mensagens indicando onde a podem ir buscar. Uma vez que os diversos componentes não se conhecem uns aos outros[2], o proxy assume o papel de reencaminhador de mensagens. Por exemplo, sempre que o engine recebe informação por parte do proxy de que os valores dos encoders foram alterados, o engine actualiza o seu estado e retorna ao proxy uma mensagem com os novos pontos visíveis pela câmara, cabendo-lhe o papel de reencaminhar as mensagens à interface.

4.1.3 Interface

Desenvolvida em Flash, a interface é a "cara" da aplicação, com a qual os utilizadores do Miradouro Virtual® interagem. Em traços gerais, a interface guia o utilizador por uma sequência de ecrãs de opções até chegar ao ecrã principal, onde o utilizador tem acesso à informação visualizada com recurso à tecnologia de RA.

Como foi referido na secção 2.2, o miradouro foi pensado desde início para ser o mais intuitivo possível. Com esta permissa, a interface foi desenhada para ser

[1] Um hub é a parte central de conexão de uma rede concentrando em si a ligação entre diversos computadores.

[2] Não existem referências explicitas no código para os diversos componentes

mais fácil e imediata que a versão anterior, com a menor curva de aprendizagem possível. Com base nos estudos ergonómicos e testes funcionais da versão anterior, efectuaram-se algumas mudanças à interface, nomeadamente, os botões foram melhor adaptados ao tamanho médio dos dedos dos utilizadores e ganharam um aspecto menos *flat* para serem mais facilmente reconhecidos como botões. Toda a iconografia presente nos botões da versão anterior desapareceu, dando lugar a descrições textuais e foram adicionadas tooltips para ajudar o utilizador em caso de dúvidas, facilitando a utilização da interface. Na figura 4.4 está representada a interface da versão anterior desenvolvida em Virtools.

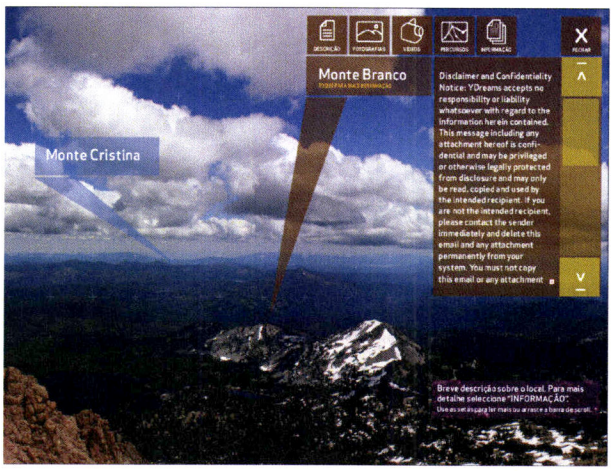

Figura 4.4: Interface da versão 2 do Miradouro Virtual®. A imagem não é um *screenshot* da aplicação real de virtools, mas sim uma maqueta digital feita pelo designer.

Foram realizados diversos testes empíricos, de tentativa erro, sobre os mais diversos fundos para chegar às conclusões das cores escolhidas: o amarelo e o azul. Estas cores para além de oferecerem uma combinação visualmente agradável possuem um grande contraste mostrando funcionar bem em diversas situações.

4.1.4 Engine

Denominado de Engine, este componente está encarregue da representação virtual 3D do mundo real, bem como de toda a parte de cálculo e conversão de coordenadas. Apesar de ter sido implementado em YVision e poder tirar partido do motor 3D disponível na framework, o engine foi o único componente que não sofreu grandes melhorias relativamente à versão anterior. O trabalho que foi feito no engine foi essencialmente um porte de código da versão virtools para a versão YVision.

Na versão anterior e devido a limitações do motor 3D do virtools foram implementados de raíz todos os algoritmos e equações necessários ao funcionamento do Miradouro Virtual®. Nesta versão continuamos a utilizar as mesmas equações implementadas na versão anterior, mas reescritas para C#.

De uma forma pouco técnica, pode dizer-se que a representação interna, do mundo 3D, por parte do engine, se assemelha a uma esfera dentro da qual todos os pontos do miradouro são colocado. No centro da esfera existe um modelo de câmara virtual, que tenta representar o mais fielmente possível o comportamento da câmara real.

Sempre que há alterações nos encoders, o engine e notificado pelo proxy e actualiza a posição da câmara virtual de acordo com os valores obtidos. Depois e de acordo com os parâmetros e posição da câmara virtual são calculados quais os pontos visíveis e convertidos para coordenadas 2D (x,y) a partir das suas coordenadas esféricas.

Ao nível do engine ainda há muito trabalho de melhoramento que pode ser feito. Na secção 6.3 estão descritas algumas das ideias e propostas para uma próxima versão.

4.2 Implementação detalhada da interface

Em termos de arquitectura, a interface é composta essencialmente por 6 entidades representadas na figura 4.5. A class Main é o ponto de entrada na aplicação, e tem como responsabilidade inicializar as classes SettingsManager e CommunicationManager com o propósito de estabelecer uma ligação socket ao proxy.

Assim que a ligação ao proxy é estabelecida, é efectuada a primeira troca de mensagens com o propósito de identificar a interface perante o proxy. A interface envia a mensagem *ConnectionRequest*, à qual o proxy responde com a mensagem *ConnectionResponse,* que contém a informação referente ao projecto. Em baixo encontra-se descrito um exemplo da mensagem *ConnectionResponse*

```
<ConnectionResponse>
   <ConfigurationFile>C:\MyPath\Cache\project_6\cfg.xml</ConfigurationFile>
   <CachePath>C:\MyPath\Cache\</CachePath>
   <ProjectPath>C:\MyPath\Cache\project_6\</ProjectPath>
</ConnectionResponse>
```

Com base nesta informação a classe Main acede ao ficheiro de configuração cujo caminho é dado pelo nó <ConfigurationFile> e utiliza as opções referentes ao projecto para seleccionar o ficheiro apropriado da máquina de estados e inicializar o ScreenManager. A partir deste momento a classe ScreenManager assume o controlo da aplicação.

Conceptualmente a interface é uma máquina de estados finitos[3], onde cada estado é representado por um ecrã. As transições entre ecrãs são efectuadas usando eventos, que são enviados em consequência da interacção do utilizador com a interface, ou devido a mecanismos internos. A classe responsável por implementar toda a lógica da máquina de estados é o *ScreenManager*.

[3]Uma máquina de estados finitos ou Autómato Finito representa a modelação de um comportamento, composto por estados, transições e acções. Um estado representa o conteúdo da memória interna, uma transição indica uma mudança de estado e é descrita por uma condição que precisa ser realizada para que a transição ocorra e uma acção é a descrição de uma actividade que deve ser realizada num determinado momento.

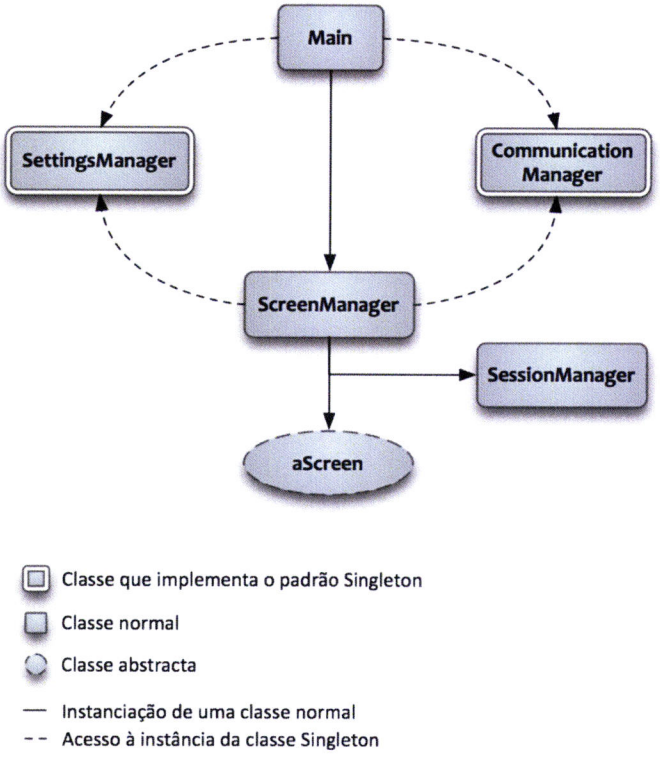

Figura 4.5: Esquema de classes da interface.
O Main é o ponto de entrada, passando o controlo da aplicação para o ScreenManager quando a ligação ao proxy é estabelecida

Na figura 4.6 está representado um esquema de uma possível máquina de estados utilizada pela interface. A máquina de estados possui um ficheiro de configuração, escrito em XML, que é carregado pelo *ScreenManager* em tempo de execução. Esta solução permite que a interface se consiga adaptar a diversas configurações, de hardware e software, sem obrigar o programador a alterar código ou a recompilar a aplicação. As opções de configuração para a máquina de estados são definidas no backoffice, existindo um número finito de configurações possíveis. É responsabilidade da class Main analisar as opções do projecto e escolher o ficheiro correcto para a máquina de estados.

Como foi referido, cada estado é representado por um ecrã da interface. O conceito de ecrã foi generalizado na classe *aScreen* que é a classe abstracta da qual todos os ecrãs derivam. Optou-se por utilizar uma classe abstracta ao invés de uma interface, porque existe uma lógica comum a todos os ecrãs que faria sentido ficar implementada ao nível mais alto.

Ao nível de comunicação a responsabilidade de estabelecer e gerir a ligação via socket com proxy, bem como processar todas as mensagens definidas no protocolo pertence à classe *CommunicationManager*. Uma vez que implementa o padrão Singleton,

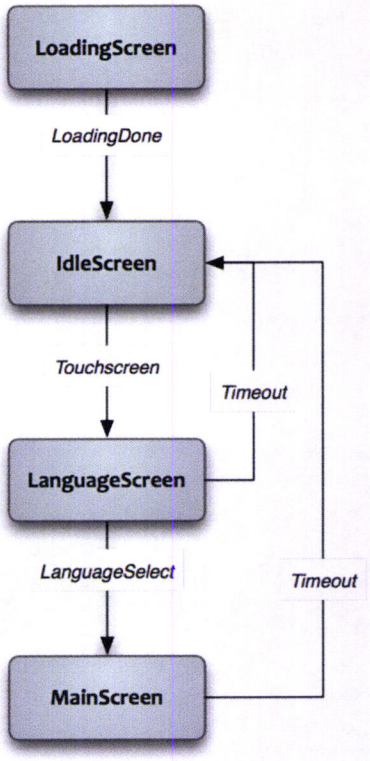

Figura 4.6: Exemplo de uma possível máquina de estados para o Miradouro Virtual®. Os nomes em itálico representam os eventos, responsáveis pela transição de estado.

todas as classes a utilizam para enviar e receber mensagens. Todas as mensagens recebidas pelo *CommunicationManager* são redireccionadas para o *ScreenManager* que por sua vez as redirecciona para a entidade correcta. Sempre que um ecrã, ou outra classe, necessita de enviar uma mensagem acede directamente ao *CommunicationManager*. Nas versões futuras pretende-se que o envio de mensagens por parte dos ecrãs siga o mesmo fluxo das mensagens recebidas, i.e., que o ecrã envie um evento para o *ScreenManager* que por sua vez o reencaminha para o *CommunicationManager*.

Em algumas configurações, o Miradouro Virtual® possui um moedeiro, e necessita que o utilizar coloque uma moeda para poder ter acesso à informação. O *SessionManager* é a classe responsável por fazer a gestão das sessões do utilizador. Sempre que uma moeda é introduzida no moedeiro, este comunica ao proxy que uma moeda de um dado tipo foi introduzida, que por sua vez notifica a interface através da mensagem *CoinInserted*. O *ScreenManager* encarrega-se de reencaminhar a mensagem para o *SessionManager* que trata de toda a lógica da sessão.

A classe *SettingsManager* funciona como um ponto central para guardar e gerir as configurações internas da aplicação. O *SettingsManager* também implementa o padrão Singleton providenciando desta forma a qualquer classe acesso às configu-

rações. Esta classe guarda entre outros parâmetros da aplicação como o endereço e porto do socket de comunicação com o proxy.

A sequência típica de ecrãns do Miradouro Virtual® é composta por 5 passos: LoadingScreen, IdleScreen, LanguageScreen, CoinScreen e MainScreen. Nas secções seguintes são descritos de forma breve cada um dos ecrãns.

4.2.1 LoadingScreen

O LoadingScreen é sempre o primeiro ecrã da máquina de estados, pois é ele o responsável por inicializar as estruturas de dados internas com a informação relativa ao projecto.

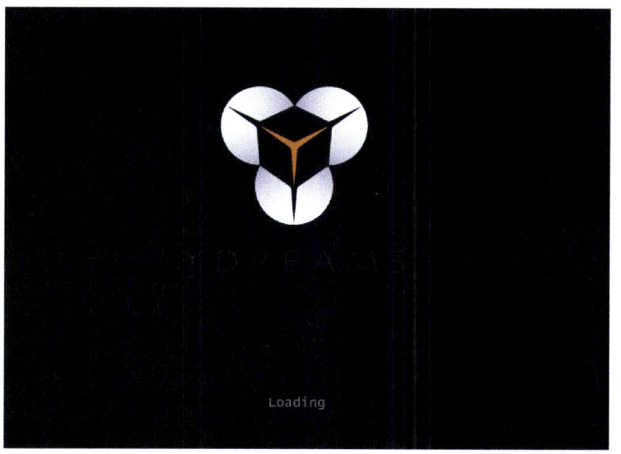

Figura 4.7: Ecrã do LoadingScreen, o primeiro ecrã da interface do miradouro cuja responsabilidade é carregar todos os elementos necessários ao funcionamento da interface do Miradouro Virtual®.

Com base na informação obtida pelo *ConnectionResponse*, o LoadingScreen instancía uma classe que converte a informação do ficheiro XML em objectos do tipo *DataElement*, um para cada língua configurada para o projecto. No caso do projecto possuir 3 línguas teríamos 3 instâncias da classe classe *DataElement*

Como podemos ver no esquema representado na figura 4.8, existem outras classes das quais o *DataElement* depende. A Classe Poi representa os pontos de interesse disponíveis no projecto. Esta classe contém toda a informação que a interface necessita para mostrar visualmente os pontos de interesse incluíndo imagens, vídeos ou descrição textual.

A classe *Resource* representa um elemento multimédia associado à interface. Os recursos de sistema utilizados nos diversos ecrãs, como o video de instrodução ou o audio de boas vindas, e comuns a todos os projectos são guardados na estrutura *systemResources* do *DataElement*, enquanto os elementos multimédia referentes aos pontos de interesse são guardados nas estruturas correspondentes da classe Poi.

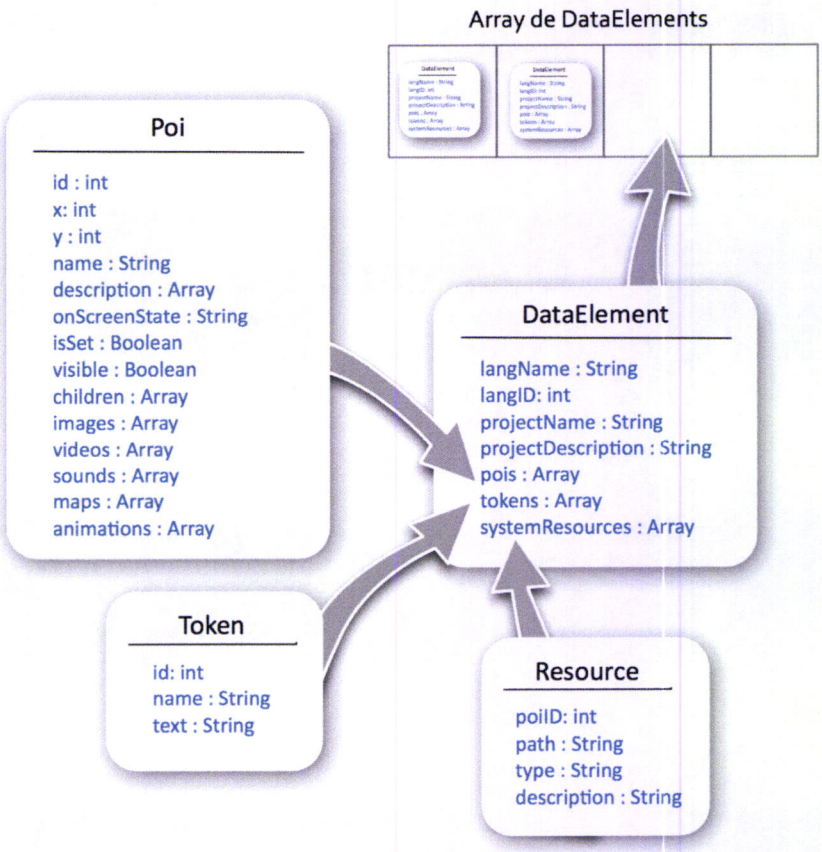

Figura 4.8: Esquema da classe DataElement

A classe Token representa *strings* estáticas da interface. Sempre que existe um texto estático num ecrã da interface, cujo texto dependa da língua seleccionada, é utilizado um Token. Por exemplo, o CoinScreen representado na figura 4.11 possui um campo de texto no canto inferior direito onde é apresentada a string "Insira Moeda", porque o idioma definido como principal era o portugês. Se fosse definida outra língua como principal, por exemplo o inglês, na mesma caixa de texto aparecia a mensagem "Insert Coin". Isto acontece porque é definido que na caixa de texto estará o conteúdo referente ao Token com o nome "COIN_INSERT_MESSAGE". Consoante o idioma seleccionado, é mostrado o conteúdo desse Token.

Os vários objectos da classe DataElement são guardados numa variável de classe do tipo array pertencente à classe aModel. Esta classe pretende concentrar em si todos os dados da interface fazendo parte do padrão MVC implementado pelo MainScreen, do qual falaremos na secção 4.2.5.

4.2.2 IdleScreen

Quando o LoadingScreen termina a conversão do ficheiro XML para objectos do tipo *DataElement* envia o evento "LoadingDone" que é recebido pela máquina de estados e faz transitar de estado e carregar o IdleScreen.

Figura 4.9: Imagem do ecrã IdleScreen que dá instruções ao utilizador sobre o funcionamento do miradouro

O papel do IdleScreen é dar informação ao utilizador sobre o funcionamento do miradouro. É mostrado um vídeo explicativo da interface e funcionamento da unidade e dada a indicação ao utilizador para tocar no ecrã para avançar. Como podemos ver no esquema da figura 4.6 este estado funciona quase como o estado inicial da aplicação.

Quando o utilizador toca no ecrã, é enviado o evento "Touchcreen" que é recebido pela máquina de estados e a faz transitar de estado e carregar o próximo ecrã, o LanguageScreen.

4.2.3 LanguageScreen

O LanguageScreen dá as boas vindas ao utilizador e apresenta uma lista de idiomas sob a forma de botões com a descrição textual na língua de origem. Os idiomas são configurados no backoffice a nível de projecto, isto é, da lista de idiomas disponíveis no backoffice, um projecto pode ser configurado para um ou mais idiomas. O número de botões disponíveis na lista é uma consequência directa desta configuração. Caso exista apenas um idioma configurado para o projecto este ecrã não é mostrado.

Quando o utilizador selecciona uma língua é lançado o evento "LanguageSelect" que ao ser recebido pela máquina de estados a faz transitar de estado, carregando o próximo ecrã, que pode ser CoinInfoScreen caso exista moedeiro configurado ou o MainScreen caso contrário. Se ainda neste ecrã, não existir qualquer interacção com a interface, através de toques no *touchscreen*, é enviado, ao fim de algum tempo, o

evento "Timeout", que ao ser recebido pela máquina de estados a faz transitar para o estado inicial, e carregar o ecrã IdleScreen.

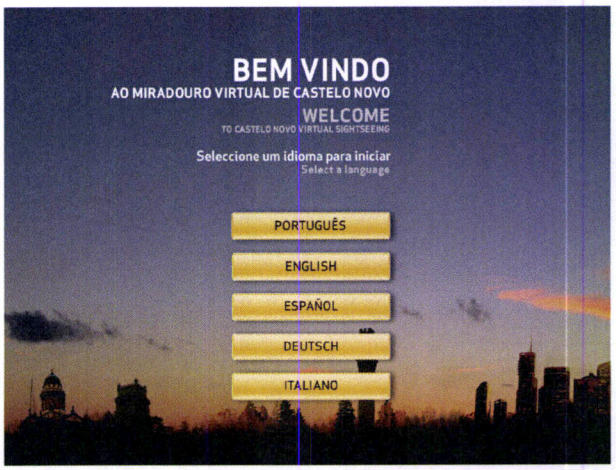

Figura 4.10: Imagem do ecrã de escolha de idioma

4.2.4 CoinInfoScreen

Se a unidade estiver equipada com um moedeiro, a máquina de estados irá carregar o ecrã CoinInfoScreen que informa o utilizador dos custos e duração da visita para cada moeda aceite. A configuração do tipo de moedas aceite pelo miradouro, bem como a duração, em minutos, correspondente a cada moeda é feita no backoffice. Tal como no ecrã LanguageScreen, se ao fim de algum tempo não existirem interacções é enviado o evento "Timeout", que ao ser recebido pela máquina de estados a faz transitar para o estado inicial, e carregar o ecrã IdleScreen.

Figura 4.11: Imagens do ecrã CoinInfoScreen que informa o utilizador dos custos e duração da visita

Quando uma moeda é introduzida, o moedeiro notifica o proxy que por sua vez envia a mensagem "CoinInserted" à interface. A máquina de estados ao receber este evento, desencadeia a transição de estado para o próximo ecrã, o MainScreen.

4.2.5 MainScreen

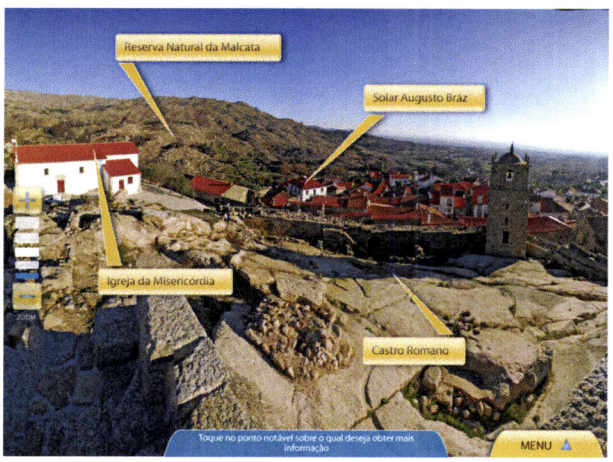

Figura 4.12: Ecrã principal da interface

Este é o ecrã mais complexo da interface e o mais importante, pois é aqui que a interface assume a sua forma principal de navegação pelo utilizador. Devido à sua complexidade foi construído segundo uma variação do padrão MVC. A figura 4.13 em baixo mostra o esquema de classes implementado pelo ecrã MainScreen.

Uma breve análise ao esquema de classes revela a omissão de um dos componentes do MVC, o Controller. Inicialmente o Controller fez parte da arquitectura, como gestor de inputs para a interface, mas ao fim de algum tempo percebeu-se que era desnecessário e as suas responsabilidades foram integradas na MainView.

A integracão de objectos gráficos na interface do Miradouro Virtual®, é um pouco diferente do modo tradicional de construção de interfaces, como por exemplo os forms do windows. Ao contrário dos forms que utilizam componentes, com uma lógica implementada, o Flash permite que qualquer elemento gráfico, como um simples círculo, possa ser manipulado directamente no código. O ecrã principal é composto por diversos objectos gráficos desenhados pelo designer, uns mais complexos que outros. Alguns dos elementos mais complexos implementam a sua própria lógica, comportando-se como uma espécie de controlos, enquanto a lógica dos mais simples fica implementada directamente na view. Para simplificar o tratamento de eventos e manter coerência entre os diversos tipos de objectos gráficos optou-se por colocar toda a lógica de interface na view, incluindo o tratamento de eventos de inputs do utilizador que estavam no Controller.

Apesar da não existência do Controller, o princípio fundamental do MVC é respeitado, existindo uma clara separação dos dados e da forma como estes são apresentados

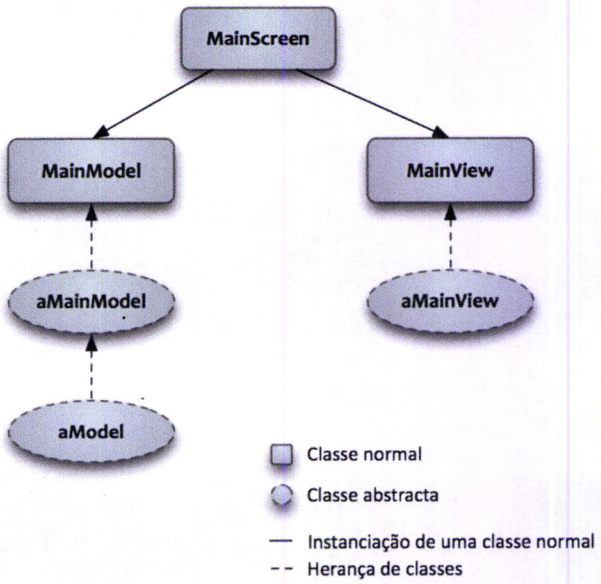

Figura 4.13: Esquema de classes usando uma variação do MVC utilizada no ecrã MainScreen

na interface.

A classe MainScreen é responsável por instanciar os diversos componentes do MVC, neste caso o Model, representado pela classe MainModel e a view representada pela classe MainView. A classe MainModel extende a classe abstracta aMainModel que por sua vez extende a classe aModel. Como vimos anteriomente, na secção 4.2.1 a estrutura de dados principal, à qual todas as classes acedem, é guardada na classe aModel. A classe aMainModel contém a lógica particular necessária à MainView. Os restantes ecrãns não foram implementados segundo o padrão MVC devido à sua simplicidade.

A view é constituída por etiquetas amarelas que identificam os pontos de interesse (poi), e que permitem visualizar mais informação acerca destes, uma barra azul de tooltips que fornece pistas ao utilizador para a navegação na interface, um botão de menu que permite aceder a opções para realizar acções complementares à navegação pelo interface, e o controlo do zoom que permite controlar o zoom da câmara.

As tooltips estão sempre visíveis na caixa azul que se encontra na parte inferior/-centro da interface de navegação. Estas dicas de navegação estão sempre indexadas às acções a tomar para a utilização da interface para ajudar o utilizador em caso de necessidade.

As etiquetas estão programadas com um comportamento que permite a coexistência de vários POIs sem se sobreporem, reagindo à posição uma das outras e aos movimentos do Miradouro Virtual® 2.3.

O utilizador tem a possibilidade de rodar livremente (360^{o}) a unidade do Miradouro Virtual® para observar todo o espaço em seu redor. À medida que roda o miradouro e vão surgindo etiquetas no ecrã *touchscreen* a sinalizar os pontos de

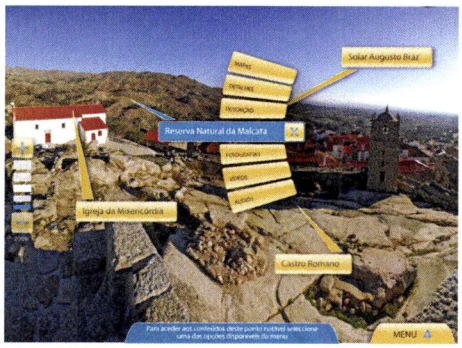

Figura 4.14: Visualização do menu de opções do POI. As opções incluem, descrição textual, fotografias, vídeos, mapas, sons, mapas e pontos interiores.

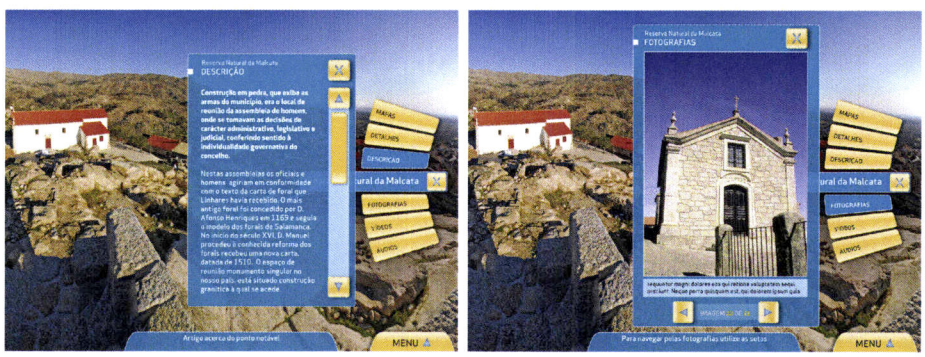

Figura 4.15: Interface mostrando as opções de descrição e fotografias de um ponto de interesse

interesse. Quando o utilizador encontra um ponto de interesse sobre o qual gostaria de receber mais informação, pode tocar na etiqueta para fazer aparecer um menu com as opções de conteúdo disponíveis para esse POI como está representado na figura 4.14.

Caso o utilizador escolha qualquer um dos conteúdos associados ao POI, as restantes etiquetas desaparecem do ecrã e a etiqueta seleccionada desloca-se-á para o lado direito do ecrã ao mesmo tempo que surge uma secção ao centro onde é mostrado o conteúdo seleccionado. Na figura 4.15 em baixo temos exemplificado dois conteúdos, descrição textual e fotografias de um POI.

Por vezes surgem um grande número de pontos de interesse concentrados no mesmo local, o que torna complicado o posicionamento de tantas etiquetas no mesmo espaço e dificulta a leitura ao utilizador. Para ultrapassar este problema foi criado o conceito de ponto interior. Um ponto interior é em tudo idêntico a um ponto de interesse, mas não tem uma etiqueta associada, uma vez que existe associado outro ponto de interesse. O ponto de interesse representa no fundo um grupo de pontos notáveis. É escolhido um para ser mostrado sob a forma de etiqueta, e os restantes surgem dentro da opção detalhes desse ponto. Um exemplo recorrente são

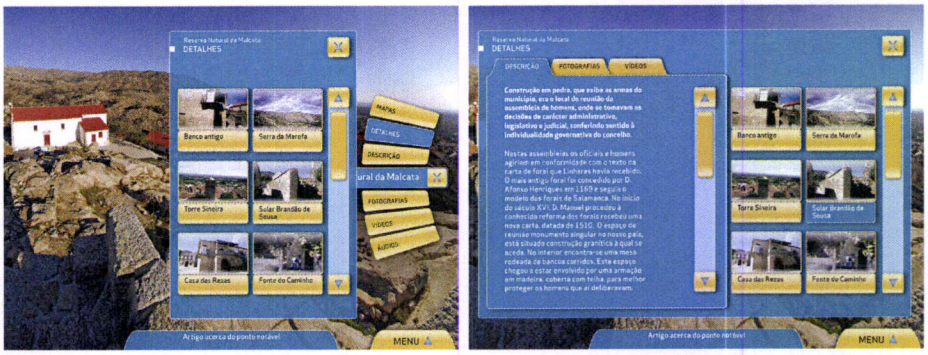

Figura 4.16: Interface dos pontos interiores de um ponto de interesse

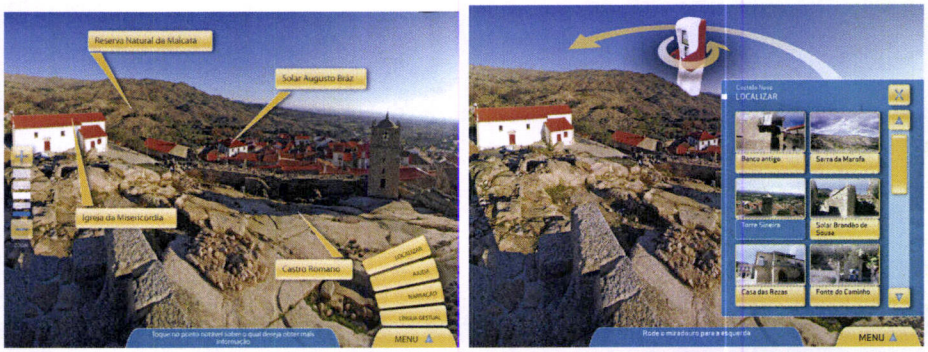

Figura 4.17: A imagem à esquerda mostra as opções do menu da interface. A imagem à direita mostra a aparência visual da opção de pesquisa

as localidades, que por vezes se encontram bastante longe do local do miradouro e para não colocar diversos pontos de interesse no mesmo local, são criados pontos interiores associados ao ponto de interesse que representa a localidade. Na figura 4.16 em baixo podemos ver os pontos interiores associados ao POI seleccionado.

A navegação nos pontos interiores processa-se através de uma lista com título e imagem dos POIs. Após selecção de um ponto interior, a janela expande-se para a esquerda disponibilizando os conteúdos existentes (descrição, fotografias, vídeos, áudios).

No canto inferior direito está sempre visivel o menu de opções da interface. O menu possui 4 opções: ligar/desligar língua gestual, ligar/desligar Narrador, Ajuda e Localizar.

A opção de localizar permite ao utilizador encontrar um ponto no espaço seguindo as instruções do ecrã. Desta forma o utilizador fica a conhecer de forma rápida quais são os pontos notáveis disponíveis no miradouro e se quiser procurar um em particular, basta selecciona-lo e seguir as instruções.

A opção de ajuda mostra um vídeo exemplificativo do funcionamento da interface do Miradouro Virtual®.

As restantes opções do menu, Língua Gestual e Narrador são opcionais. Se o idioma for configurado no backoffice como sendo acessível, terá associado videos de língua gestual e audios. Estes videos e audios podem ser activados usandos as opções do menu.

4.2.6 Posicionamento automático dos POIs

Como foi referido na secção 2.3 o problema do posicionamento das etiquetas foi um dos maiores, senão o maior, problema encontrado no desenvolvimento da interface. A bibliografia com mais de 250 referências para posicionamento de etiquetas no domínio da cartografia, foi uma ferramenta indispensável na concepção do algoritmos final [22].

Existem diversos tipos de algoritmos para tentar solucionar o problema do posicionamento de etiquetas como os apresentados na seccão 2.3. No caso particular do miradouro foram utilizadas 3 técnicas distintas que combinadas, criaram um solução considerada satisfatória para o problema [22, 23].

A versão anterior da interface do Miradouro Virtual®, desenvolvida em Virtools, respondia ao problema criando uma grelha e colocando as etiquetas nas posições da grelha. Sempre que era necessário colocar uma nova etiqueta, o algoritmo procurava um conjunto de quadrículas de tamanho suficiente para alojar a etiqueta que estivesse livre e se existisse colocava a etiqueta nessa posição. Esta solução obrigava a que o tamanho das etiquetas fosse múltiplo de uma quadrícula da grelha para conseguir calcular quantas quadrículas seriam necessárias. Este algoritmo era pouco flexível e bastante limitativo do número de etiquetas que se podia colocar no ecrã simultaneamente, porque podia haver muito espaco livre mas que desta forma não daria para ser aproveitado. Sempre que se rodava o miradouro, as etiquetas deslocavam-se de quadrícula em quadrícula originando um movimento bastante rígido.

No caso no Miradouro Virtual®, um ponto de interesse é constituído visualmente por um ponto fixo (designado de âncora daqui por diante), uma etiqueta de comprimento variável com o seu nome e uma linha também de comprimento variável que une a etiqueta à âncora como podemos ver na figura 4.18.

Para a nova versão da interface estudaram-se diversas formas para tentar resolver o problema do posicionamento de etiquetas. As tentativas iniciais partiram do pressuposto que o posicionamento das etiquetas seria feito em tempo de calibração. Sempre que se calibrava um ponto era calculada uma posição para a nova etiqueta que não violasse nenhuma das restrições:

- As etiquetas não se podiam sobrepôr a outras etiquetas
- As etiquetas não se podiam sobrepôr às linhas
- As etiquetas não se podiam sobrepôr às âncoras.

A primeira tentativa encontrada para resolver o problema, não recorria a nenhum algoritmo, mas sim à intervenção humana. Esse papel cabia à pessoa (designada

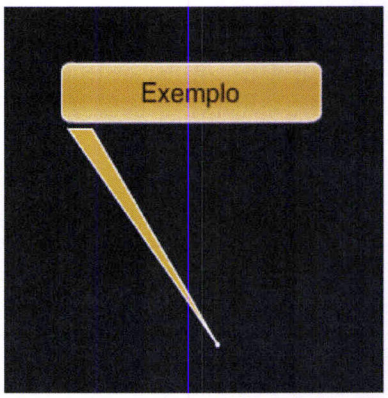

Figura 4.18: Representacão gráfica de um ponto de interesse. Composto (de cima para baixo) por uma etiqueta, uma linha em forma triangular e um ponto fixo.

de superutilizador daqui em diante) responsável por calibrar os pontos no sistema e posicionar as diversas etiquetas de forma a não violar as restrições. O superutilizador tinha a possibilidade de movimentar livremente a etiqueta pelo ecrã até encontrar um local adequado.

A segunda tentativa utilizou um algorimo de pesquisa "greedy", chamado *Simulated annealing*, para ajudar o superutilizador na calibração dos pontos. Sempre que era marcado um novo ponto, o algoritmo efectuava uma pesquisa no espaço e calculava a melhor posição para colocar a etiqueta. Se o algoritmo não conseguisse encontrar nenhuma solução no fim do número máximo de iteracões, seria novamente responsabilidade do superutilizador fazer o posicionamento manual da etiqueta.

Qualquer uma destas soluções estava longe de ser uma solucão satisfatória, obrigando a um posicionamento estático das etiquetas, o que dava pouca flexibilidade ao sistema.

As tentativas seguintes abandonaram a ideia de que o posicionamento das etiquetas tinha de ser feito em altura de calibração e partiram do princípio que o posicionamento das mesmas seria feito em tempo real.

A primeira tentativa passou por adaptar o algoritmo de *Simulated annealing* para funcionar em tempo real ao invés de em tempo de calibração. Foram realizados diversos testes ao algoritmo, mas apesar de este se comporta bem na grande maioria dos casos, em determinadas situações, o algoritmos não foi capaz de encontrar uma solução em tempo útil.

A tentativa seguinte passou por utilizar um motor de física de corpos rígidos em tempo real para detectar e resolver as colisões entre etiquetas. Analisaram-se alguns dos motores de física disponíveis para Actionscript e escolheu-se o APE. [32] Foi criada uma representacão física do POI, utilizando uma partícula circular[4] como âncora, uma mola para simular a linha e uma partícula rectângular para simular a etiqueta. A representação do ponto de interesse para o motor de física encontra-se

[4]As partículas circulares foram definidas como fixas, não estando sujeitas às forças da simulação

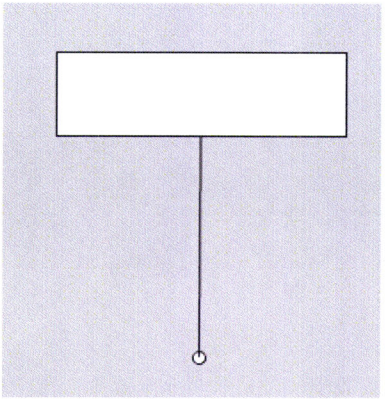

Figura 4.19: Representacão gráfica dos elementos físicos, particula rectangular, linha e particula circular, que compõem o ponto de interesse como ele é representado para o motor de fisica.

representado da figura 4.19

As partículas circulares e as partículas rectangulares foram agrupadas em dois grupos e foi definido na simulacão que os dois grupos estariam sujeitos a forças de colisão entre partículas do grupo oposto, bem como dentro do próprio grupo.

Após a configuração inicial do sistema este era capaz de detectar e resolver colisões entre as etiquetas, entre as etiquetas e as molas, e entre as etiquetas e os pontos. Esta foi a primeira solução satisfatória para o problema, que comparativamente às soluções anteriores revelou um comportamento menos restritivo e movimentos mais orgânicos. Os passos seguintes consistiram em optimizações a esta solução.

Inicialmente aplicou-se uma força ao sistema, semelhante à gravidade mas invertida, que afectava todas as etiquetas para que estas fossem atraídas para a parte superior do ecrã, como se de balões se tratassem. Essa ideia foi mais tarde abandonada, porque o aumento do número de pontos no ecrã, levava a uma grande concentração de etiquetas na parte superior, o que fazia aumentar as colisões e levava a um aumento de instabilidade do motor de física. A optimização seguinte passou por aplicar forças verticais individualmente a cada etiqueta. Se a âncora estivesse para cima do meio do ecrã, era aplicada uma força vertical na etiqueta para que esta fosse atraída para a parte superior do ecrã, e o inverso caso esta estivesse para baixo de meio do ecrã. A figura 4.20, em baixo mostra um exemplo do posicionamento das etiquetas usando esta optimização.

Durante algum tempo esta solução foi considerada satisfatória com o conjunto de pontos utilizado para testar o sistema, no entanto quando se testou com novos conjuntos de pontos, verificou-se que existia um grande número de cruzamentos entre os fios das etiquetas e o posicionamento das mesmas tornava difícil de perceber a que âncora pertencia cada etiqueta. Este comportamento agravava-se especialmente quando existiam grandes concentrações de pontos.

Foi necessário pensar numa optimização para evitar os cruzamentos das linhas

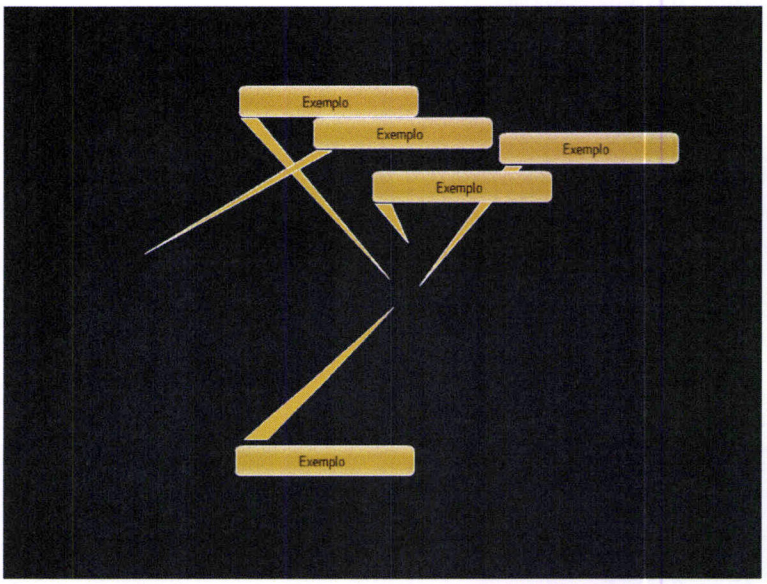

Figura 4.20: Posicionamento das etiquetas usando o motor de física e aplicando forças individuais a cada etiqueta

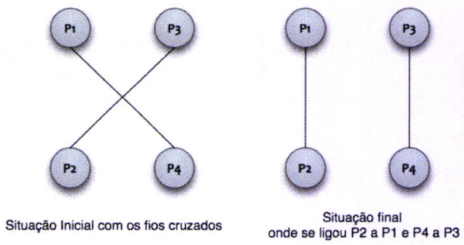

Figura 4.21: Exemplificacão do algoritmo para descruzar linhas

entre etiquetas, e melhorar a disposição visual das etiquetas.

A primeira optimização efectuada diz respeito ao cruzamento de linhas entre etiquetas. Foi implementado um algoritmo para retirar cruzamentos existentes entre etiquetas.

Este algorítmo em primeiro lugar percorre todas as linhas à procura de cruzamentos e sempre que descobre duas linhas cruzadas, descruza-as, redesenhando as linhas como é demonstrado na figura 4.21

O cálculo da intersecção entre duas linhas foi efectuado recorrendo ao algoritmo de Paul Bourke [37], também descrito no livro Graphics Gems III que para a detecção de uma intersecção consiste no cálculo de duas equações

$$u_a = \frac{(x4-x3)(y1-y3)-(y4-y3)(x1-x3)}{(y4-y3)(x2-x1)-(x4-x3)(y2-y1)}$$

$$u_b = \frac{(x2-x1)(y1-y3)-(y2-y1)(x1-x3)}{(y4-y3)(x2-x1)-(x4-x3)(y2-y1)}$$

Analisando o resultado de u_a e u_b é possivel tirar informação sobre o posicionamento das linhas.

- Se o denominador para ambas as equacões for 0 então as duas linhas são paralelas.
- Se o denominador e numerador para ambas as equacões for 0 então as duas linhas são coincidentes.
- Se o resultado de u_a e u_b estiver entre 0 e 1 então as linhas intersectam-se.

Este algoritmo mostrou-se bastante rápido a calcular as intercecções e a descruzar as linhas. A disposição visual das etiquetas depois de retirados os cruzamentos melhorou substancialmente.

A última optimização recorreu a um algoritmo de forças atracção/repulsão semelhante ao referido na secção 2.3.4 para melhorar ainda mais a disposição visual.

Um dos pontos fortes destes algoritmos, que os levou a serem utilizados no Miradouro Virtual®, tem a ver com facto de se basearem em analogias físicas, de objectos comuns que utilizamos no dia a dia, como corpos físicos ou molas. O comportamento destes algoritmos é bastante fácil de prever e perceber, porque é uma simulação do que se passa no mundo real.

No caso concreto do Miradouro Virtual® recorreu-se à implementação do algoritmo criado por Fruchterman e Reingold [29] para melhorar a disposição visual das etiquetas.

A figura 4.22 mostra uma configuração melhor para a mesma situação da figura 4.20, onde as etiquetas fora posicionadas com a versão final do algoritmo, baseada no motor de fisica APE, algoritmos para retirar cruzamentos e algoritmo de forças de atracção/repulsão.

A versão final com base no motor de física e com as optimizações para retirar cruzamentos e dispor as etiquetas de forma mais natural ainda não é perfeita, porque por vezes os dois sistemas de forças, o motor e o algoritmo de atração/repulsão entram em conflito, mas é bastante aceitável como solução para o problema do posicionamento de etiquetas.

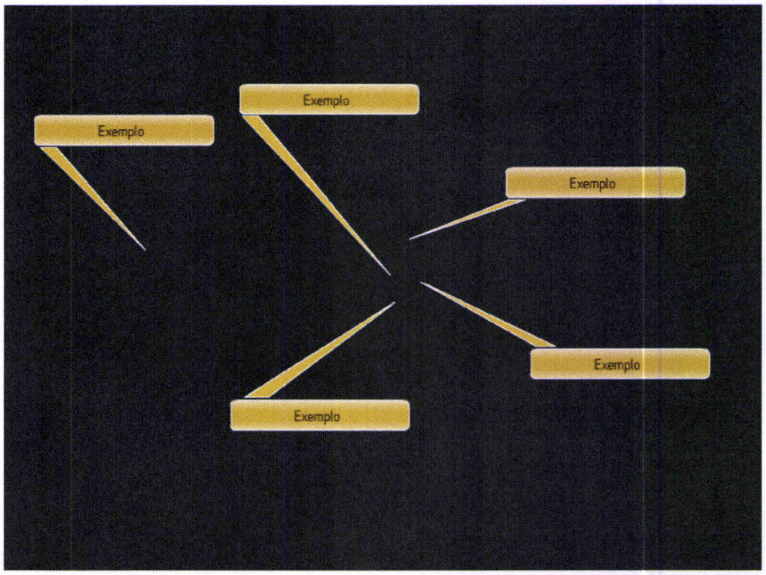

Figura 4.22: Representação gráfica da versão final do algoritmo de posicionamento de etiquetas, baseada no motor de física APE, algoritmo para retirar cruzamentos e algoritmo de forças de atracção/repulsão

Capítulo 5

Usabilidade

Neste capítulo é descrita a análise ergonómica efectuada à versão anterior e de que modo é que esta análise contribuiu para corrigir e melhorar algumas das falhas e limitações nesta nova versão. Na secção 5.2 é descrito o estudo efectuado com base em dados estatísticos recolhidos das unidades do Miradouro Virtual® e de que forma esses resultados permitem inferir conclusões acerca da nova versão.

5.1 Análise da versão anterior

A versão anterior do Miradouro Virtual® foi sujeita a uma análise ergonómica com o objectivo de avaliar a usabilidade do produto e identificar problemas, para que se pudessem desenvolver soluções e apoiar estas decisões de desenvolvimento para a nova versão. Das opiniões dos utilizadores registadas durante os testes, a que mais de destacou foi a falta da funcionalidade de Zoom.

Os resultados obtidos da análise ergonómica, revelaram que seria necessário efectuar diversas alterações à estrutura física do Miradouro Virtual®, como a dimensão,

posteriormente transferidos para um servidor central onde são realizados diversos estudos estatísticos.

Este tipo de estudo estatístico, não permite tirar conclusões acerca da usabilidade da aplicação, mas fornece informações importantes que permitem inferir certos comportamentos por parte dos utilizadores e detectar eventuais problemas.

5.2.1 Descrição dos testes

O estudo efectuado para esta tese baseou-se nos dados estatísticos recolhidos do Miradouro Virtual® da ponta do Sal, instalado em Cascais, durante o período de dia 1 a dia 24 do mês de Março de 2009.

Quando os miradouros possuem moedeiro, o tempo de uma sessão de utilização é determinado pela quantidade de moedas introduzidas. No caso do miradouro escolhido para testes, não existe moedeiro. A sessão de utilização foi definida, como consistindo no tempo que vai desde o primeiro toque efectuado pelo utilizador no *touchscreen,* até não se verificarem mais toques durante um período de 210 segundos (3,5 minutos). Este mecanismo funciona com base num temporizador. Sempre que existe uma interação com o *touchscreen* é feito reset ao contador e o contador chegar ao fim é declarado o final da sessão de utilização. O tempo mínimo para uma sessão de utilização é de 3,5 minutos, e corresponde a uma utilização onde o utilizador toca no ecrã, dando inicio à sessão e logo de seguida vai embora. Uma vez que não existe tempo máximo definido para uma utilização, o tempo da sessão de utilização pode ser bastante diverso, no entanto para este estudo, o tempo máximo para uma sessão foi fixado em 15 minutos.

As acções passiveis de serem registadas por parte do utilizador foram as seguintes:

- *POI_EXPANDED* - Registado sempre que o utilizador toca numa das etiquetas

- *LANGUAGE_BUTTON* - Registado sempre que o utilizador selecciona um idioma no ecrã de selecção de língua
- *MENU_BUTTON* - Registado sempre que o utilizador utiliza uma funcionalidade do menu
- *MENU_SEARCH* - Registado sempre que o utilizador selecciona um ponto através da pesquisa
- *ZOOM_IN* - Registado sempre que o utilizador interage com o controlo de zoom para fazer zoom in
- *ZOOM_OUT* - Registado sempre que o utilizador interage com o controlo de zoom para fazer zoom out

5.2.2 Resultados

O gráfico representado na figura 5.1 representa o número de sessões de utilização por dia do mês.

Somando as sessões de utilização para os dias definidos no período em estudo, chegamos ao valor de 473, o que corresponde a uma média de 23,65 utilizações por dia. Nos dias 5, 8 e 15, foram contabilizados valores bastante baixos, que podem ser atribuídos a diversos factores como problemas no funcionamento do Miradouro Virtual®, condições climáticas ou outros.

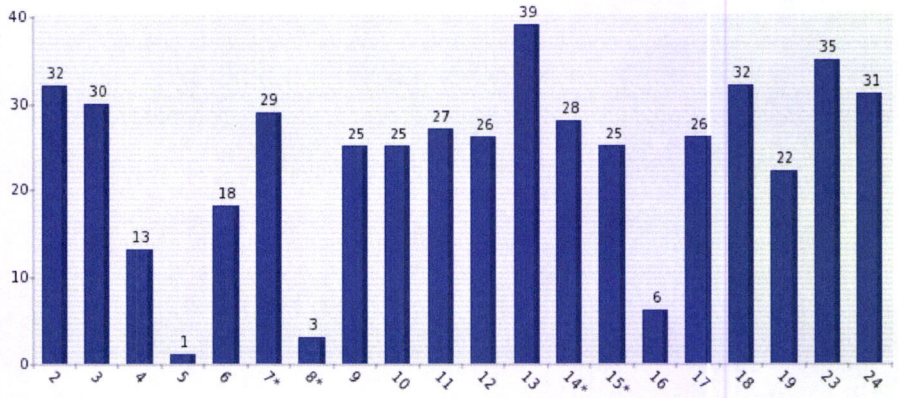

Figura 5.1: Gráfico que representa o total de sessões de utilização por dia. No eixo dos X (horizontal) estão representados os dias do mês, e no eixo dos Y (vertical) está representado o total de sessões de utilização. Os * representam os fins de semana, Sábado e Domingo.

O gráfico representado da figura 5.2 mostra o tempo médio de uma sessão de utilização no mesmo período. Olhando para o gráfico representado na figura 5.1 verificamos que dia 13, foi o dia onde existiu maior número de utilizações, no entanto o tempo médio de utilização não foi muito elevado. Comparativamente ao dia 15

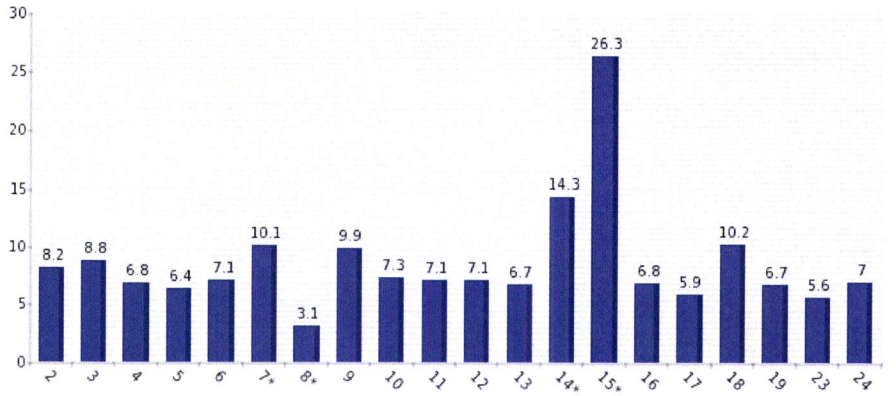

Figura 5.2: Tempo médio de utilização por dia. No eixo dos X (horizontal) estão representados os dias do mês, e no eixo dos Y (vertical) está representado o tempo das sessões de utilização. Os * representam os fins de semana, Sábado e Domingo.

onde existiram 25 utilizações, o tempo médio de utilização foi de 26,3 minutos, o maior do período de estudo.

A média dos valores médios de utilização para o período em estudo é de 8,6 minutos. Se retirarmos o tempo que o sistema necessita para finalizar a sessão que é de 3.5 minutos, então podemos dizer que o Miradouro Virtual® é utilizado em média durante 5 minutos.

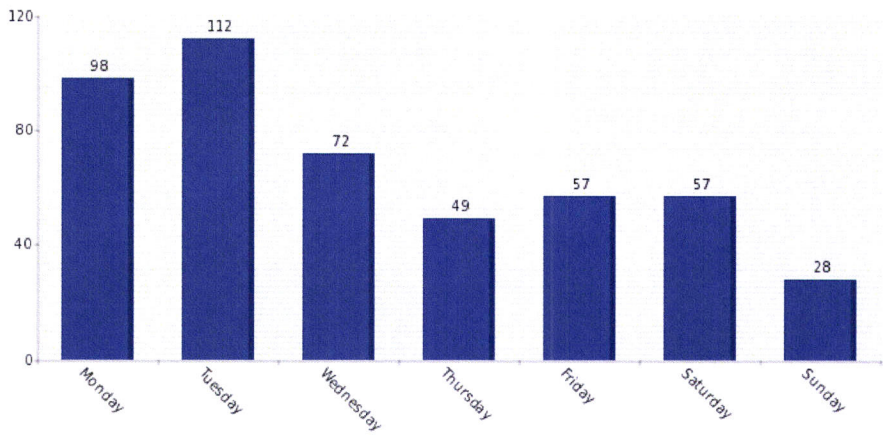

Figura 5.3: Total de sessões de utilização por dia da semana

A figura 5.3 mostra o total de utilizações por dia de semana. É curioso verificar que não é aos fins de semana que o Miradouro Virtual® é mais utilizado, mas sim às terças feiras.

O gráfico representado na figura 5.4, representa o número total de utilizações ordenadas por tempo, isto é, quantos utilizadores é que interagiram com o Miradouro

Virtual® durante quanto tempo. Como foi referido anteriormente, o tempo máximo de utilização para estes estudos foi definido em 15 minutos, tendo sido descartados os valores acima desse tempo.

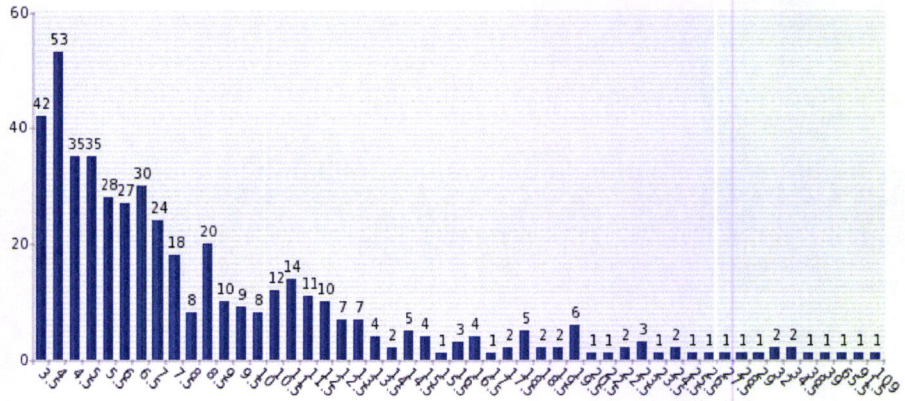

Figura 5.4: Gráfico que representa o número de utilizações ordenado por tempo de interacção. No eixo dos X (horizontal) está representado o tempo de utilização em minutos, e no eixo dos Y (vertical) está representado o total das utilizações.

A primeira coluna representa os utilizadores que abandonaram o aparelho após ter tocado uma vez no ecrã. A segunda coluna representa os utilizadores que seleccionaram um idioma no ecrã de escolha de língua, mas abandonaram o aparelho logo de seguida, interagindo com a aplicação num período inferior a 30 segundos.

Contabilizando o número de sessões de utilização até aos 15 minutos, chegamos ao valor de 424. Se tivermos em conta as duas primeiras colunas correspondentes aos utilizadores que não interagem realmente com a aplicação, temos um total de 329 utilizadores que interagiram com a aplicação entre 4 e 15 minutos, o que corresponde a 77% dos utilizadores. Os restantes 33% dos utilizadores, não chegaram realmente a interagir com a aplicação, talvez porque tiveram apenas curiosidade de ver o que acontecia se tocassem no ecrã, não tendo realmente intenção de interagir com o sistema, ou por não conseguissem utilizar a aplicação devido a problemas de idioma ou problemas de usabilidade da interface.

O gráfico representado na figura 5.5 representa as acções registadas pela interface durante a utilização do sistema por parte do utilizador. O gráfico mostra que a funcionalidade de zoom é claramente a mais requisitada pelos utilizadores do Miradouro Virtual®. O gráfico revela também que existiram 994 pedidos de visualização do menu dos pontos de interesse, i.e., que o utilizador seleccionou um ponto de interesse no ecrã. Uma vez que existiram 473 utilizações únicas, um utilizador só seleccionou em média, 2 pontos de interesse. Este valor é bastante baixo e pode indicar que os utilizadores não percebem que podem interagir com os pontos de interesse. No que diz respeito à funcionalidade de zoom, a diferença de valores pode indicar que os

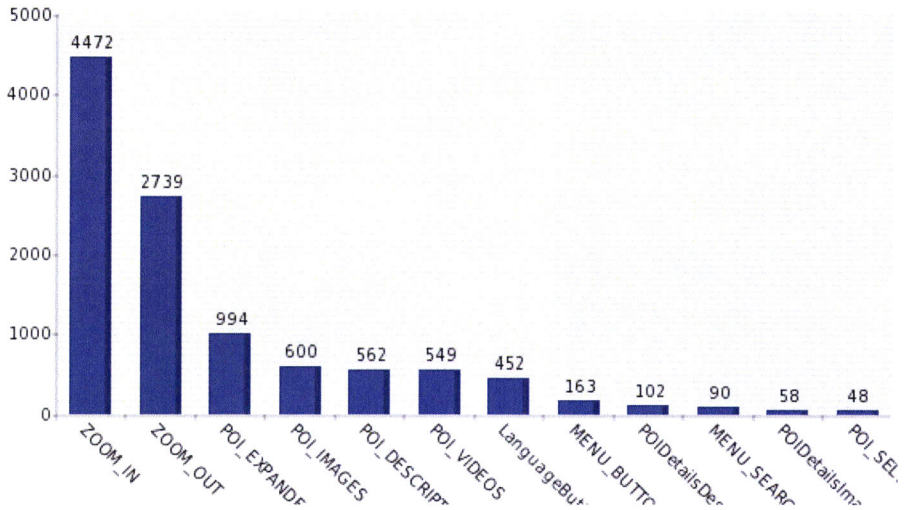

Figura 5.5: Acções realizadas pelo utilizador na interface do Miradouro Virtual®. No eixo dos X (horizontal) estão representados os dias do mês, e no eixo dos Y (vertical) está representado o tempo das sessões de utilização.

utilizadores recorrem mais ao Miradouro Virtual® para ver a paisagem, do que para ver a informação associada aos pontos de interesse.

A figura 5.6 representa os caminhos percorridos pelos utilizadores durante as suas sessões de utilização com o Miradouro Virtual®. Os círculos simbolizam as acções realizadas pelo utilizador e o seu tamanho representa o número de vezes que a acção foi realizada, quanto maior mais vezes foi realizada. A espessura da linha simboliza o número de vezes que aquele caminho foi percorrido, quando mais grossa a linha mais vezes o utilizador realizou aquela sequência de acções. A cor simboliza o tempo entre acções, quando mais vermelho maior o tempo entre acções. Claramente o tamanho dos círculos referentes ao zoom destacam-se em relação aos restantes.

A figura 5.6 oferece uma visão geral da utilização da interface permitindo analisar a forma como o utilizar interage com mesma. O caminho mais percorrido pelo utilizador depois de seleccionar o idioma é seleccionar o ponto de interesse e visualizar as suas fotografias.

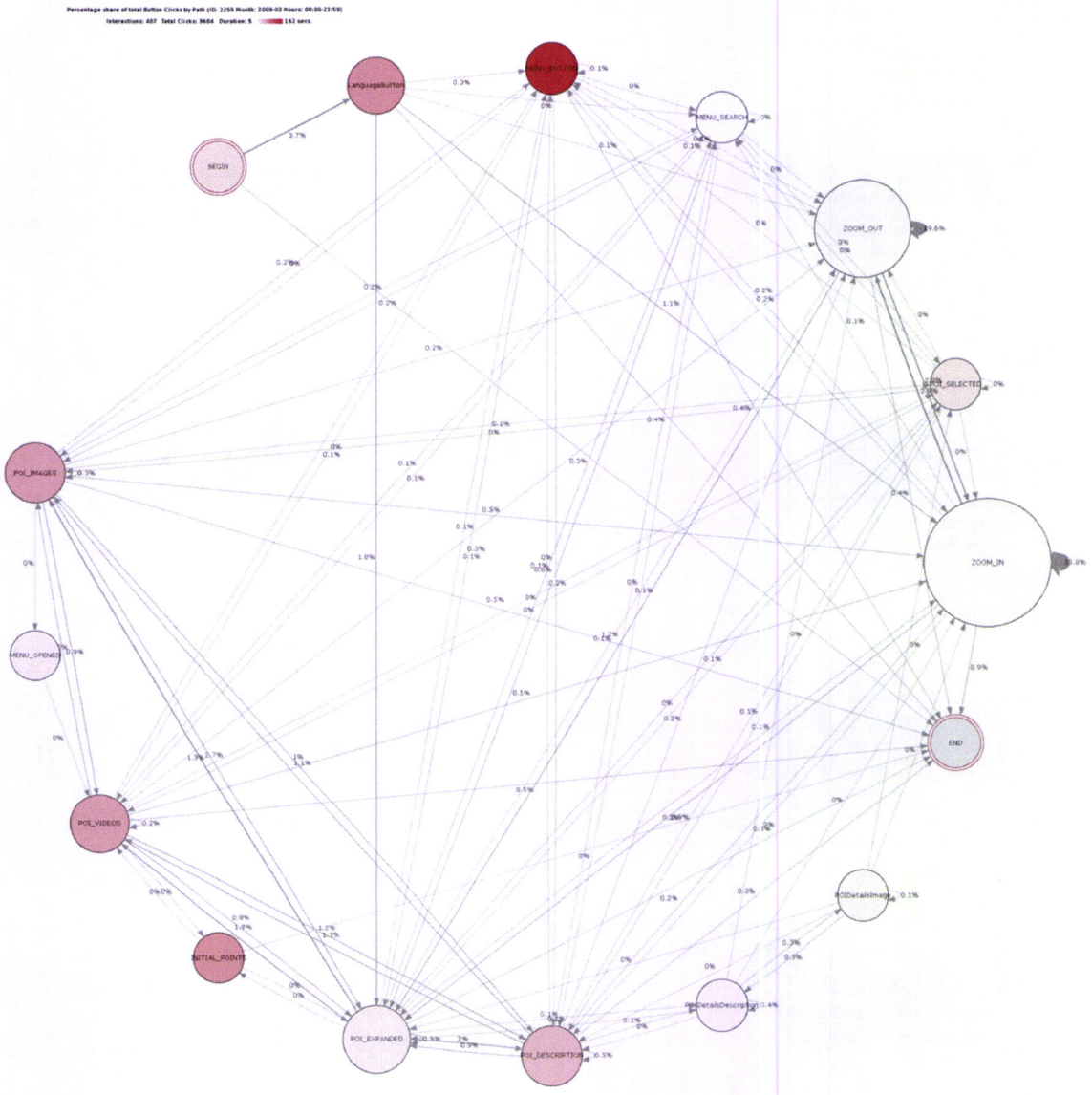

Figura 5.6: Figura representativa dos percursos efectuados pelos utilizadores durante as diversas utilizações com o Miradouro Virtual®. A versão de alta resolução pode ser encontrada no seguinte link http://tinyurl.com/actionPaths

Capítulo 6

Conclusões

Este capítulo faz um balanço sobre o trabalho apresentado ao longo da dissertação. Mais concretamente, na secção 6.1 são enumerados os objectivos alcançados, na secção 6.2 são apresentadas limitações conhecidas ao sistema desenvolvido e na secção 6.3 são sugeridos possíveis melhoramentos ao trabalho apresentado.

6.1 Objectivos Alcançados

Como foi descrito na secção 1.2, o objectivo deste trabalho incidiu no desenvolvimento de uma nova aplicação para o Miradouro Virtual®, capaz de ultrapassar as limitações e resolver alguns dos problemas da versão anterior, nomeadamente na componente da interface que foi o foco desta tese. Esta nova versão desenvolvida utilizando Flash e YVision cumpriu os objectivos criando um sistema modular e bastante versátil.

Ao nível da interface, esta nova versão veio melhorar bastante o aspecto gráfico da aplicação. Conseguiu resolver todas as limitações impostas pelo virtools a nível de design, como suporte de fontes e os elementos gráficos não rectangulares, revelando que o Flash, foi uma escolha acertada. O resultado final obtido na interface é em tudo idêntico ao concebido pela equipa de design. O algoritmo de posicionamento de etiquetas é agora mais dinâmico e os movimentos das etiquetas mais naturais e dinâmicos.

O resultado final é bastante satisfatório, tendo sido cumprido o objectivo proposto.

6.2 Limitações

Embora a solução implementada garanta o cumprimento integral dos objectivos propostos para este trabalho, existem limitações que no futuro deverão ser eliminadas de modo a tornar o sistema mais preciso e robusto.

A limitação mais notória prende-se com o componente do Engine. O facto de não estar a utilizar o motor 3D integrado no YVision inviabiliza o uso de modelos

3D georeferenciados no espaço, uma vez que seria necessário implementar de raiz os algoritmos necessários. Caso o engine já estivesse integrado com o motor 3D, o mapeamento de modelos 3D poderia ser utilizado.

Seria necessário uma configuração mais detalhada e exaustiva da câmara virtual, mas graças ao motor 3D, adicionar distorção de lente e outras correcções à imagem seria mais simples e rápido face à implementação existente, onde seria necessário implementar de raiz os complexos algoritmos.

6.3 Trabalho Futuro

Olhando para o trabalho desenvolvido, verifica-se que existem alguns melhoramentos possíveis a fazer na aplicação.

Ao nível da interface, a implementação de um modelo simplificado do motor de física evitaria ter dois sistemas de forças a competir entre eles.

Apesar do algoritmo final para o problema do posicionamento ser considerado satisfatório podem existir determinadas condições onde o motor de física e o algoritmo de atracção/repulsão entram em conflito, i.e., o motor de física ao tentar resolver uma colisão move a etiqueta para uma determinada posição e logo a seguir o algoritmo de atracção/repulsão força a etiqueta a ir para a posição original. Enquanto o utilizador não mover o miradouro e o algoritmo for obrigado a calcular novas posições, o impasse não é resolvido.

Outro melhoramento a fazer seria ao nível da comunicação entre os diversos componentes. Esta versão recorre ao XML como formato de transmissão de mensagens. Este formato é bastante verboso, aumentando o tamanho das mensagens e por conseguinte o tempo de transmissão o que leva a um aumento no desfasamento entre o movimento físico do aparelho e o respectivo movimento das etiquetas no ecrã. O melhoramento passa por trocar mensagens usando o formato binário.

Uma das funcionalidades a implementar na nova versão é o mapeamento de modelos 3D para recriação de acontecimentos históricos ou visionamento de simulações interactivas. Para tal é necessário alterar o componente do engine para utilizar o motor 3D do YVision.

Outra melhoria seria utilizar coordenadas latitude, longitude e altitude para georeferenciar os pontos, o que evitaria grande parte do processo de calibração e permitiria tirar partido das inúmeras bases de dados de pontos de interesse existentes hoje em dia.

Lista de Figuras