Instituto Politécnico de Beja

Escola Superior de Tecnologia e Gestão

Curso de Engenharia Informática

Relatório de Projecto

“Sistema de Construção Visual de Interfaces”

Cláudia Isabel Hermozilha Oliveira, N.º 3459

Cláudio Samuel Monteiro Pedro, N.º 3805

Nuno Filipe Alves Coelho, N.º 3938

2007

Instituto Politécnico de Beja

Escola Superior de Tecnologia e Gestão

Curso de Engenharia Informática

Relatório de Projecto

Sistema de Construção Visual de Interfaces

Relatório de projecto de fim de curso, apresentado na Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja, como parte dos requisitos necessários à obtenção do grau de licenciado no curso de Engenharia Informática.

Alunos

Cláudia Isabel Hermozilha Oliveira, N.º 3459

Cláudio Samuel Monteiro Pedro, N.º 3805

Nuno Filipe Alves Coelho, N.º 3938

Orientador

Eng.º José Jasnau Caeiro

Elementos do Júri

Presidente: Eng.º José Jasnau Caeiro

Arguente: Eng.º Luís Garcia

Vogal: Eng.º Rui Pais

Beja, Julho de 2007

Sumário

Este relatório foi elaborado no âmbito do projecto final de curso dos alunos, e descreve todas as fases de desenvolvimento da ferramenta Qooxdoo GUI Builder, que permite gerar interfaces gráficas de forma assistida, com base em componentes visuais da *framework* AJAX qooxdoo.

Índice

1 Introdução

1.1 Descrição do projecto

A construção de aplicações *Web* mais interactivas é possível através do modelo de desenvolvimento *Web*, AJAX[1] [1]. Um dos factores fundamentais para o aumento de interactividade é a troca constante de pequenas quantidades de dados com o servidor, em *background*. Desta forma, as páginas *Web* não precisam de ser actualizadas na sua totalidade, cada vez que é feita uma alteração.

Qooxdoo [2] é uma das mais poderosas *frameworks* AJAX *open-source* porque disponibiliza uma vasta quantidade de classes e um conjunto de *widgets* que dificilmente se distinguem dos elementos gráficos que estamos habituados a ver em aplicações nativas de sistemas operativos. É uma *framework* que permite desenvolvimento através da linguagem JavaScript, suporta comunicação cliente-servidor de alto nível e inclui um *toolkit* gráfico que apresenta características semelhantes às de outras bibliotecas gráficas existentes, como a Qt (Dalheimer, 2002) [3].

As interfaces qooxdoo são desenvolvidas através de *scripts* escritos na linguagem JavaScript, que permitem maximizar o desempenho de execução das mesmas. Integradas num ambiente *Web* do tipo cliente-servidor, estas interfaces são processadas no lado do cliente através de uma *framework* JavaScript instalada no *browser*, que implementa as propriedades gráficas da biblioteca qooxdoo. Ou seja, qualquer *browser* que suporte *JavaScript* como o Internet Explorer, Mozilla Firefox ou Opera, pode processar estas interfaces.

O desenvolvimento de interfaces gráficas em modo de texto é moroso e de difícil acerto dadas as claras dificuldades com que um programador se depara, como por exemplo, o caso da frequente compilação ou execução da codificação para visualização da interface criada. Com o objectivo de ajudar no desenvolvimento de interfaces gráficas, existem hoje em dia inúmeras ferramentas do tipo IDE[2], que permitem de forma interactiva e rápida, construir interfaces bem estruturadas e apelativas. Ao contrário do que acontece com outros *toolkits* gráficos, para o qooxdoo não existe actualmente nenhum ambiente do género, desenvolvido.

Este projecto teve como objectivo o desenvolvimento de uma ferramenta gráfica que servisse de suporte à construção de interfaces constituídas por elementos do *toolkit* gráfico qooxdoo. Pretende-se com esta ferramenta diminuir o tempo gasto pelos programadores na construção das interfaces, facilitando-lhes ainda a construção das mesmas. No desenvolvimento desta ferramenta recorreu-se ao *toolkit* gráfico Qt que apresenta duas vantagens de extrema

[1] *Asynchronous* JavaScript *and* XML
[2] *Integrated Development Environment*

relevância. Permite efectuar *drag & drop* e os seus controlos gráficos possuem *layouts* semelhantes aos de controlos com a mesma função, existentes no *toolkit* qooxdoo.

A ferramenta desenvolvida destina-se à *framework* AJAX qooxdoo, mas também poderia ter sido desenvolvida para outras *frameworks*, bibliotecas ou *toolkits* AJAX. De seguida são apresentados aspectos a ter em conta sobre alguns dos maiores concorrentes da qooxdoo. A *framework* Bindows [4], por exemplo, oferece um excelente GUI[3] e não requer conhecimentos de HTML[4] nem de CSS[5], mas tem a grande desvantagem de não ser *open-source*. Já o *toolkit* Dojo [5], é *open-source*, ajuda a optimizar o código JavaScript e permite a criação de *widgets*, no entanto, problemas com o DHTML[6], têm-no impedido de ser adoptado em massa pelos programadores. Pode-se ainda referir a biblioteca *open-source* Prototype [6], que possui um poderoso sistema de classes e é orientada a objectos, mas não apresenta *widgets*.

1.2 Metodologia

Para desenvolver software de qualidade é necessário ter em consideração um conjunto de factores como:

- O que fazer antes de começar a desenvolver?
- Quais as estruturas necessárias para um desenvolvimento adequado?
- O que deve ser feito para o software responder às necessidades estabelecidas?
- Que técnicas se devem utilizar para controlar a complexidade, prever as falhas, documentar o software e controlar as versões?

No desenvolvimento desta ferramenta adoptou-se o modelo de desenvolvimento conhecido por modelo cascata [10], por este ser recomendado para o desenvolvimento de software onde existe um conjunto de requisitos estáveis e de elevada qualidade, e em que a duração prevista para a elaboração do desenho do mesmo não é extensa.

[3] *Graphical User Interface*
[4] *HyperText Markup Language*
[5] *Cascading Style Sheet*
[6] *Dynamic HyperText Markup Language*

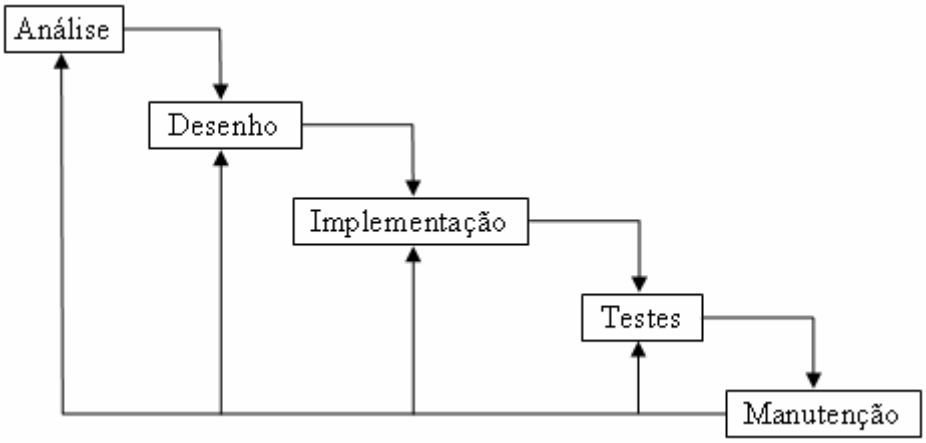

Figura 1.1 – Modelo Cascata

O desenvolvimento deste projecto dividiu-se em várias fases, de acordo com o modelo apresentado e por forma de endereçar cada um dos factores anteriormente listados, no devido momento.

Numa fase inicial, que correspondeu à fase de análise no modelo acima apresentado, foi estudado o problema de forma a identificar as características que a ferramenta a desenvolver deveria ter. Foram estudadas ferramentas análogas de forma a reter as principais características que oferecem, e a perceber as dificuldades que poderiam surgir no desenvolvimento de um projecto deste tipo.

De seguida, numa fase que correspondeu à fase de desenho do modelo cascata, foi elaborado o desenho do sistema onde foram identificados os módulos de implementação do projecto, as suas características e os tempos de desenvolvimento previstos. Foram desenhados alguns modelos visuais onde foram projectadas interfaces gráficas da aplicação. Estas foram objecto de constantes refinações até adquirirem um aspecto óptimo na óptica do utilizador final.

Durante a fase de implementação existente no modelo cascata foram implementados os módulos identificados na fase de desenho. A implementação do sistema foi efectuada pelos vários elementos do grupo de uma forma partilhada, e controlada por ferramentas de controlo de versões.

Passou-se então à fase de testes, durante a qual se procurou verificar que os requisitos detectados durante a fase de análise foram complementados.

Relativamente à última fase do modelo cascata, a da manutenção, há que referir que esta não foi tomada em conta durante a realização deste projecto. No entanto, é uma fase que se estende a partir do momento em que o software é disponibilizado, e que consiste na correcção de erros previamente não detectados e no melhoramento da funcionalidade do sistema, sempre que possível.

Resta afirmar que a documentação do código foi realizada através de uma ferramenta de geração automática de código e posteriormente publicada no sítio *Web* do projecto. Também foram criados manuais de instalação e de utilização, com o intuito de fornecer algum apoio aos utilizadores.

1.3 Estrutura deste documento

Este documento divide-se por 10 secções. Na secção 1 denominada de *Introdução*, encontra-se a descrição do projecto, a metodologia adoptada, as tecnologias utilizadas, e por fim, a estrutura deste documento. Na secção 2 denominada de *Análise de requisitos*, pode-se observar um estudo realizado sobre ferramentas semelhantes, o Diagrama de Casos de Uso da aplicação, e ainda, três possíveis cenários de utilização para a mesma. Na secção 3 denominada de *Projecto*, encontram-se o Diagrama de Classes, os Diagramas de Transição e desenhos de interfaces gráficas, acompanhados das respectivas avaliações de usabilidade. Na secção 4 denominada de *Módulos de implementação*, descrevem-se as implementações mais relevantes, nomeadamente, controlos visuais, interacção entre a interface gráfica e os controlos, interpretador e gerador de código YAML e gerador de código HTML. Na secção 5 denominada de *Geração de documentação*, expõe-se o gerador e o formato utilizado na documentação. Na secção 6 denominada de *Dificuldades sentidas*, expõem-se as necessidades com que os elementos do grupo se defrontaram, explica-se a importância de ter um repositório de dados e de utilizar um sistema de controlo de versões. Na secção 7 denominada de *Testes*, pode-se ler como foi testada a aplicação. Na secção 8 denominada de *Trabalhos futuros*, são exemplificados alguns melhoramentos que poderão, futuramente, ser realizados na aplicação. Na secção 9 denominada de *Conclusão*, descrevem-se as conclusões retiradas durante a realização deste relatório, na secção 10 denominada de *Bibliografia*, estão enunciadas as referências bibliográficas adoptadas para a realização deste documento e por fim, na secção 11 denominada de *Agradecimentos*, encontram-se os agradecimentos dos alunos.

2 Análise e especificação de requisitos

2.1 Estudo de ferramentas análogas

Com o objectivo de salientar as características mais importantes numa ferramenta deste género, foi realizado um estudo que incidiu sobre várias ferramentas de desenvolvimento de interfaces gráficas. No âmbito deste estudo foram tomadas em conta as seguintes ferramentas:

Aptana [7]

Características:

- Assistência na construção do código JavaScript, HTML e CSS;
- Comunicações através dos protocolos FTP[7] ou SFTP[8];
- *Debugger* JavaScript;
- Avisos e notificações de erros;
- Permite personalizar a própria interface;
- Suporte para cruzar plataformas.

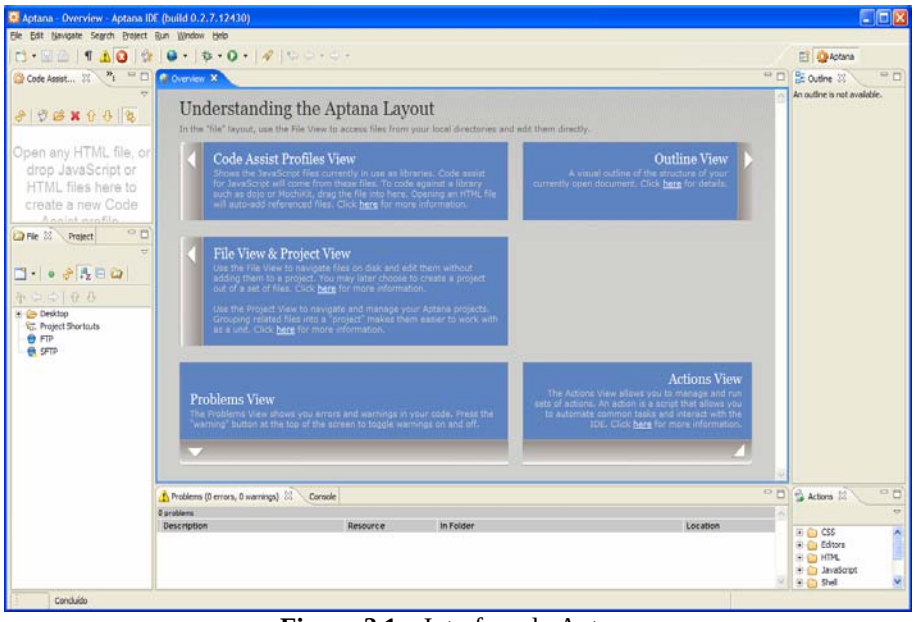

Figura 2.1 – Interface do Aptana

Macromedia Dreamweaver [8]

Características:

- Construção de sítios *Web* estáticos, constituídos por páginas *Web* HTML;

[7] *File Transfer Protocol*
[8] *Secure File Transfer Protocol*

- Criação de ficheiros de definições de apresentação para páginas HTML, através da linguagem de estilos CSS;
- Criação de páginas HTML a partir de modelos existentes;
- Armazenamento remoto de ficheiros;
- Possibilidade de adicionar à área de construção das páginas grelhas e réguas, de forma a facilitar e optimizar a construção de interfaces visuais;
- Funcionalidade *drag & drop* na colocação de controlos visuais, na área de construção das interfaces;
- Visualização estruturada de todos os ficheiros que representam o projecto em que o utilizador está a trabalhar;
- Para cada controlo visual apresentado na área de construção de interfaces, são indicadas as suas dimensões relativamente ao ecrã, o espaço ocupado em percentagem e o tamanho em *pixels;*
- Apresentação simultânea da área de desenho e do código da página *Web*;
- Criação de *records* sobre as acções do utilizador, gravando acções do mesmo, que mais tarde poderão ser repetidas sobre a área de desenho, bastando pressionar sobre a opção de repetir acção.

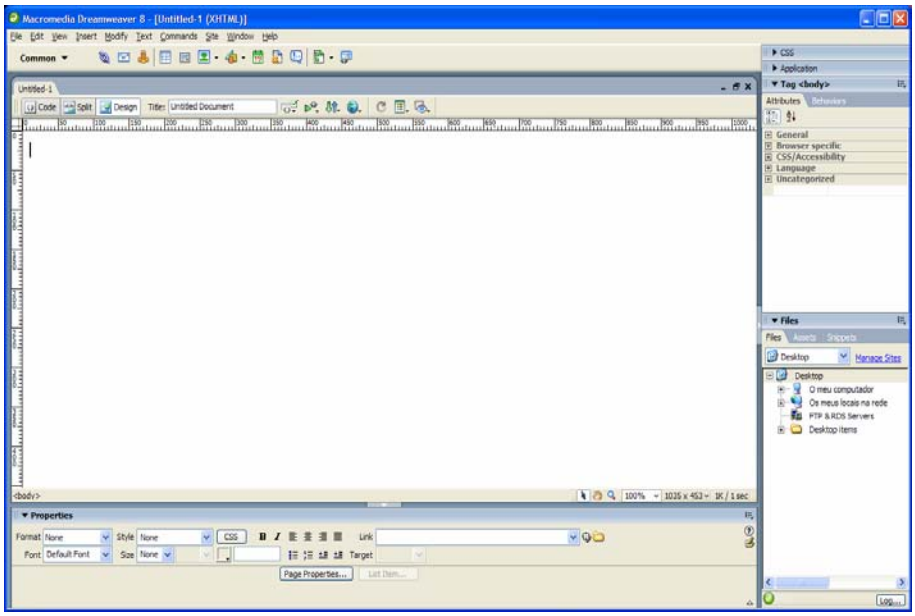

Figura 2.2 – Interface do Macromedia Dreamweaver

<u>Qt Designer</u> [9]

Características:

- *Drag & drop* na construção de interfaces gráficas;
- Modo de pré-visualização;
- Divisão dos controlos por tipo: *Input Widgets*, *Buttons*, *Spacers*, *Containers*, etc.;

- Criação de super-controlos[9];
- Criação de templates;
- Criação de janelas a partir de opções predefinidas:
  - *Dialog*;
  - *Widget*;
  - *Main Window*;
- Listagem das propriedades de um controlo;
- Listagem dos eventos associados a um controlo;
- Funcionalidades de anular, refazer, copiar, cortar e colar para executar sobre os controlos;
- Possibilidade de formatar o posicionamento dinâmico ou estático dos controlos, através do conceito de ancoragem.

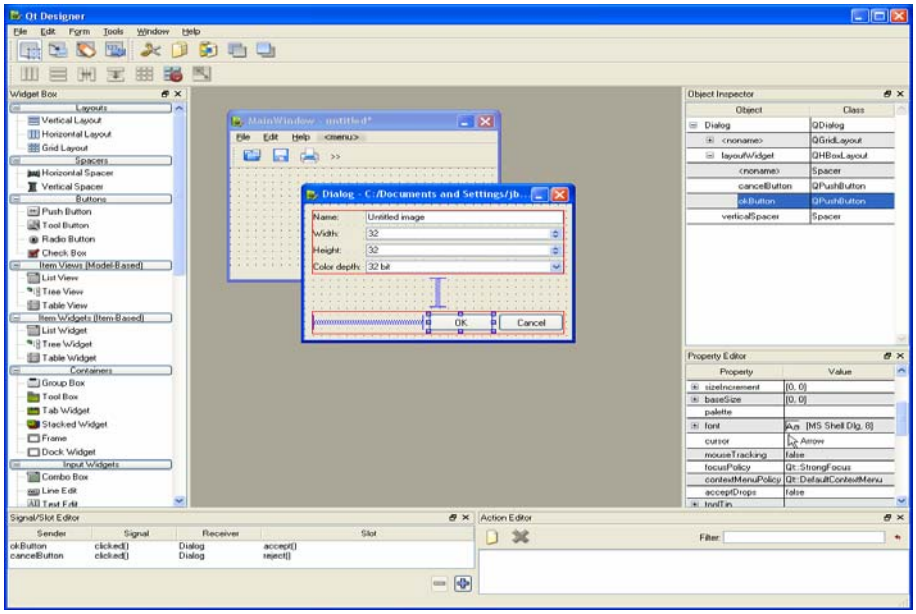

Figura 2.3 – Interface do Qt Designer

2.1.1 Características a serem implementadas

Do estudo realizado, resultou um conjunto de características que se pretendiam ver implementadas na ferramenta a desenvolver:

- Funcionalidade de *drag & drop* para a manipulação dos controlos visuais na construção das interfaces;
- Editor de propriedades para formatar os controlos visuais implementados na área de desenho;

[9] Controlo visual composto por um ou mais controlos simples.

- Armazenamento das interfaces visuais criadas num dispositivo de armazenamento seleccionado pelo utilizador;
- Armazenamento de templates com as características visuais de controlos formatados;
- Aplicação de templates de controlos visuais sobre controlos do mesmo tipo;
- Eliminação dos controlos visuais existentes na área de desenho;
- Pré-visualização da interface criada, num *browser*;
- Geração de páginas HTML à imagem das interfaces visuais criadas.

2.2 Tecnologias utilizadas

2.2.1 Python

Linguagem de programação interpretada, interactiva e orientada a objectos, que combina uma sintaxe concisa e clara com recursos poderosos existentes na sua biblioteca padrão. Por todas estas vantagens, por ser *open-source*, poder ser utilizada tanto em Windows como em Linux, e por ser uma linguagem com a qual já se tinha alguma prática, proveniente da disciplina Pesquisa e Optimização, optou-se por esta linguagem de programação, para o desenvolvimento da aplicação. Resta referir que se utilizou a versão 2.4.4.

2.2.2 Qt

Qt é um *toolkit* gráfico multiplataforma para o desenvolvimento de programas com interface gráfica. Por ser adaptável ao Python, por reunir características visuais semelhantes às da *framework* qooxdoo e por permitir o *drag & drop*, optou-se por este *toolkit* gráfico, para o desenvolvimento da interface gráfica da aplicação. Resta referir que se utilizou a versão 4.1.4.

2.2.3 YAML

Linguagem de codificação de dados inspirada em linguagens como XML[10], C, Python e Perl, assim como no formato do correio electrónico especificado pela RFC[11] 2822. Por ser legível e se adequar à necessidade de codificar interfaces gráficas e templates, optou-se por esta linguagem de codificação de dados, para esse efeito.

2.2.4 HTML

Linguagem de formatação de documentos destinados à *Web* que usa uma especificação própria para ligações hipertextuais. Nela encontra-se definida a sintaxe e a sequência de edição de determinados elementos de programação, conhecidos por *tags*, que instruem o *browser* acerca

[10] eXtensible Markup Language
[11] Request For Comments

da forma de exibição dos conteúdos do documento, não sendo esses elementos mostrados pelo mesmo. Por esta tecnologia permitir a incorporação de código JavaScript, recorreu-se à mesma, para a exibição em *browsers*, das interfaces gráficas criadas.

2.2.5 JavaScript

Linguagem de programação criada pela Netscape, inicialmente denominada por LiveScript, para atender a necessidades como validação de formulários e interacção com páginas *Web*. As principais características desta linguagem de criação de *scripts* são a tipificação dinâmica e o facto de ser interpretada, ao invés de compilada. É a linguagem que permite o desenvolvimento de interfaces gráficas qooxdoo.

2.2.6 AJAX

AJAX é um modelo que estabelece um conjunto de normas para o desenvolvimento de sistemas *Web* mais dinâmicos e criativos. Utiliza diversas tecnologias (JavaScript, XML, CSS, DOM, etc.) que ajudam a tornar as páginas de navegação mais interactivas com o utilizador, melhorando a usabilidade relativamente ao funcionamento dos sistemas *Web* implementados de acordo com o modelo clássico de comunicação, cliente-servidor.

No modelo clássico de comunicação quando o utilizador pretende visualizar um certo conteúdo ou realizar uma determinada tarefa, através do *browser*, é enviado um pedido HTTP ao servidor de dados. O pedido é processado pelo servidor, e como resposta é enviada uma página HTML com o resultado do pedido realizado. Durante o tempo da acção realizada (tempo decorrido entre o momento em que o cliente fez o pedido e a visualização da resposta do servidor) o utilizador não deverá poder interagir com o sistema *Web* e na resposta do servidor toda a página HTML será redesenhada com os novos conteúdos.

Em contraste ao modelo clássico, ver figura 2.4, o modelo AJAX tende em aproximar os sistemas *Web* das aplicações convencionais que são executadas na máquina do utilizador, incluindo uma camada intermediária entre o cliente e o servidor, convencionalmente designado como motor AJAX, que mantém as comunicações com o servidor. Assim, aquando do inicio da interacção com um sistema *Web* AJAX, o *browser* recebe com a interface visual o motor AJAX, onde sempre que são solicitados dados é estabelecida comunicação com o servidor e são actualizas as respectivas áreas de *output* na página *Web* com os dados provenientes do servidor. Desta forma as interacções do utilizador com o sistema são realizadas assincronamente, não sendo necessário que o utilizador espere por uma resposta do servidor para continuar a interagir com o sistema.

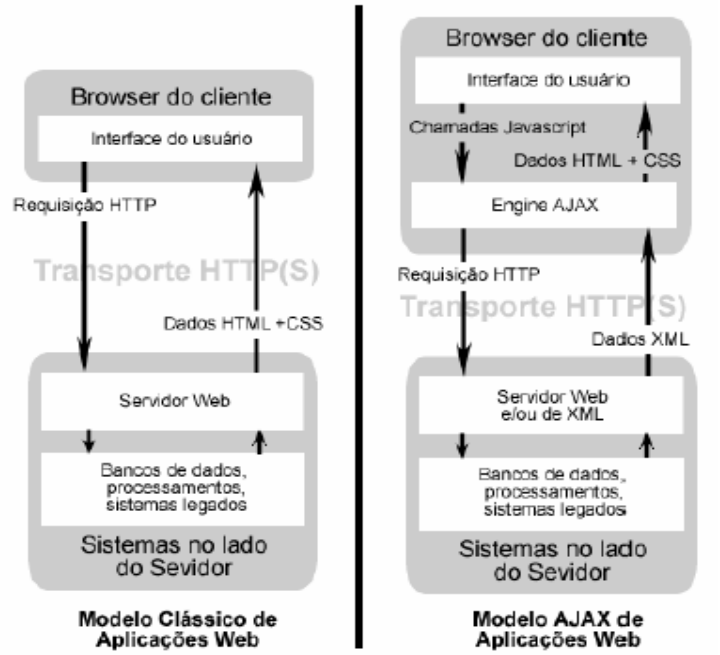

Figura 2.4 – Contraste entre os modelos, clássico e AJAX, de funcionamento de sistemas *Web*

Com o modelo AJAX os pedidos HTTP tradicionais são transformados em chamadas JavaScript ao motor AJAX. Se o pedido não necessitar de ser processado pelo servidor, o motor AJAX processa-o. Caso contrário, realiza uma requisição assíncrona ao servidor, usando normalmente XML como forma de representação dos dados. Esta tecnologia é a base do funcionamento da *framework* qooxdoo.

2.2.7 qooxdoo

Framework AJAX *open-source* que permite o desenvolvimento de interfaces *Web* recorrendo à linguagem JavaScript. Suporta comunicação cliente-servidor de alto nível e inclui um *toolkit* gráfico. É com o intuito de apoiar a criação de interfaces gráficas nesta tecnologia, que se recorre a todas as outras tecnologias.

2.3 Casos de Uso

Figura 2.5 – Diagrama de Casos de Uso da aplicação a desenvolver

2.4 Cenários de utilização

Cenário n.º 1 – Utilização de templates na formatação de controlos visuais

A Eng. Sofia, como programadora *Web* tem de projectar um sítio *Web* para um projecto sobre a marcação de férias de funcionários de uma empresa. Este sítio *Web* será constituído por várias páginas *Web*, sendo que em cada uma delas existirão vários tipos de controlos visuais.

De forma a normalizar a estética do sítio *Web*, todos os controlos visuais terão de ter uma formatação semelhante. A Eng. Sofia antes de construir a sua interface irá proceder à construção de templates de formatação para cada um dos controlos visuais que as páginas *Web* irão utilizar. Abriu a aplicação e para a área de desenho arrastou um botão e uma caixa de texto. O botão terá a função de aceitar alterações e a caixa de texto servirá para a introdução de dados. Relativamente ao botão, foi alterado o seu tamanho em largura para 60 *pixels* e em altura para 30 *pixels*. A cor de fundo escolhida para o botão foi a cinzenta e foi-lhe ainda aplicada uma imagem de fundo que reflecte metaforicamente aceitação. Relativamente à caixa de texto, foi alterado o seu tamanho em largura para 150 *pixels* e em altura para 30 *pixels*. A cor de fundo escolhida para a caixa de texto foi a amarela clara. Em ambos os controlos, o tipo de letra escolhido foi "Arial" com cor azul e tamanho 12. Para uma posterior aplicação destas formatações sobre controlos do mesmo tipo em outras interfaces, a Eng. Sofia armazenou em disco uma template para cada um destes tipos de controlos.

No dia seguinte decide então começar a construção das interfaces gráficas das páginas *Web* pela colocação dos controlos visuais necessários. A criação prévia de templates irá facilitar a formatação dos botões e das caixas de texto. A Eng. Sofia foi seleccionando um a um os botões existentes na interface e cada vez que seleccionava um, com o botão direito do rato escolhia a opção "Apply template…". Surgia-lhe então uma janela na qual seleccionava a template que queria aplicar sobre o botão, sendo então aplicada a formatação que a template armazenava. Relativamente às caixas de texto, ela seleccionou todas de uma só vez e com o botão direito do rato sobre uma delas, seleccionou a opção "Apply template…". Neste caso, escolheu a template que armazenava a formatação que a Eng. Sofia havia criado para as caixas de texto. Por fim, guardou as alterações efectuadas sobre o projecto.

Cenário n.º 2 – Pré-visualização da interface gráfica criada na aplicação e num *browser*

O Cristiano, um aluno do curso de engenharia informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja, estava encarregue do desenvolvimento de uma página *Web* pessoal para a professora da disciplina de Matemática Discreta.

Para a construção da interface da página *Web*, o Cristiano optou pela ferramenta de desenvolvimento de interfaces gráficas *Web*, Qooxdoo GUI Builder. Iniciou a aplicação, dispôs os controlos visuais que achava necessários sobre a área de desenho e formatou os mesmos a seu gosto. Recorreu à opção "Preview in application" existente no menu "Preview" da aplicação e após esta pré-visualização efectuou o derradeiro teste que passava pela visualização da interface criada, no *browser* por defeito existente no sistema. Para conseguir visualizar a interface num *browser*, recorreu à opção "Preview in browser" existente no menu "Preview" da aplicação. Satisfeito com a aparência conseguida, guardou a interface gráfica e enviou a mesma por correio electrónico para a professora.

Cenário n.º 3 – Modificação de interfaces gráficas previamente criadas

O Danilo, um rapaz empregado numa loja de informática do comércio local, decidiu nos seus tempos livres construir uma página *Web* para a loja.

Idealizou a interface gráfica que pretendia construir e iniciou a sua construção recorrendo ao menu da aplicação, onde dentro do menu "File" escolheu a opção "New interface...". Entretanto entraram clientes na loja e ele viu-se obrigado a interromper a interface na qual estava a trabalhar. Decidiu então guardá-la recorrendo à opção "Save interface…" existente no menu "File" da aplicação, para retomar mais tarde quando tivesse algum tempo livre.

Mais tarde já em casa, sem a interferência dos clientes a entrar e sair, ele decidiu retomar o seu trabalho. No menu “File” da aplicação escolheu a opção “Open interface…”. Depois seleccionou a interface que queria modificar e deu continuidade à sua construção acrescentando controlos visuais e realizando formatações sobre os mesmos.

Por fim, deu por terminada a interface em que esteve trabalhando e guardou-a recorrendo à opção “Save interface…” existente no menu “File” da aplicação. No entanto, o Danilo sabe que poderá voltar a modificá-la em qualquer momento.

3 Projecto

3.1 Diagrama de Classes

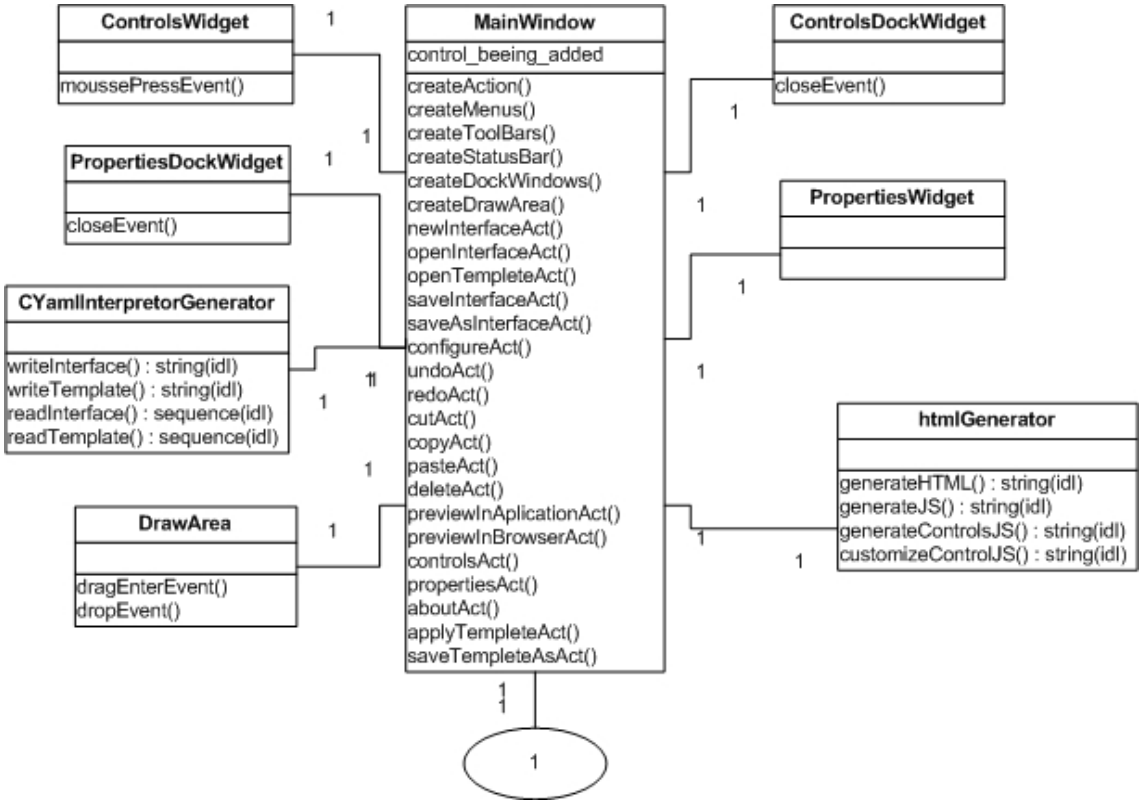

Figura 3.1 – Diagrama de Classes (Parte 1)

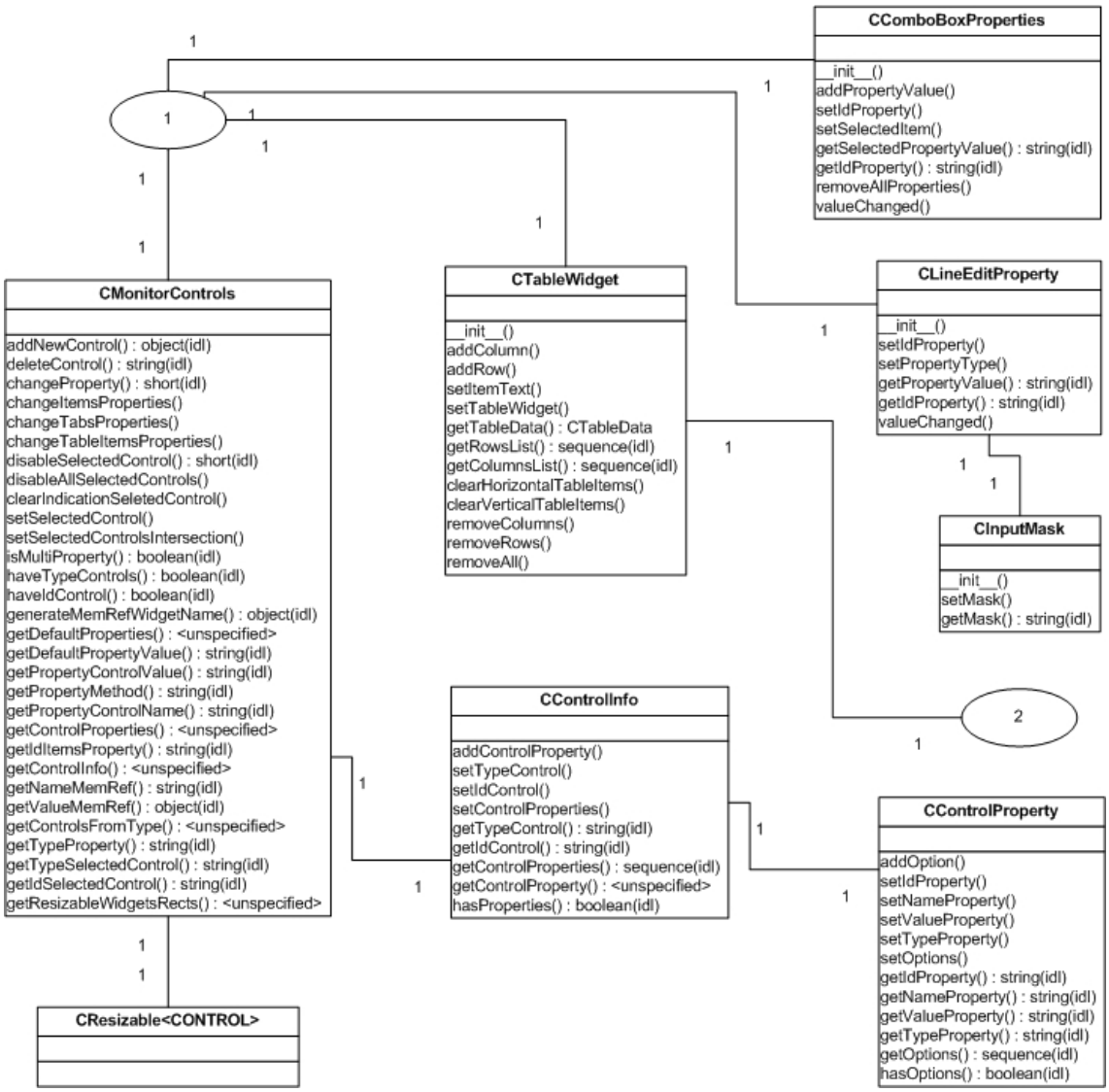

Figura 3.2 – Diagrama de Classes (Parte 2)

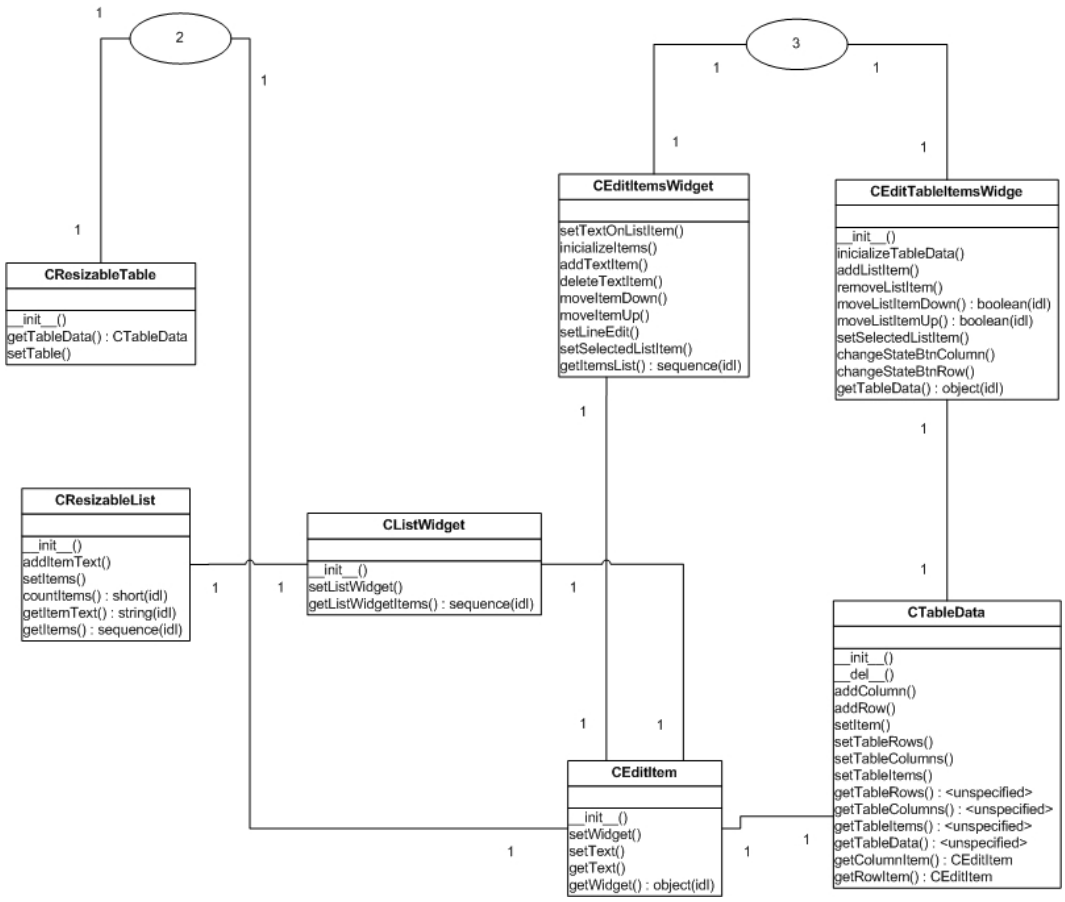

Figura 3.3 – Diagrama de Classes (Parte 3)

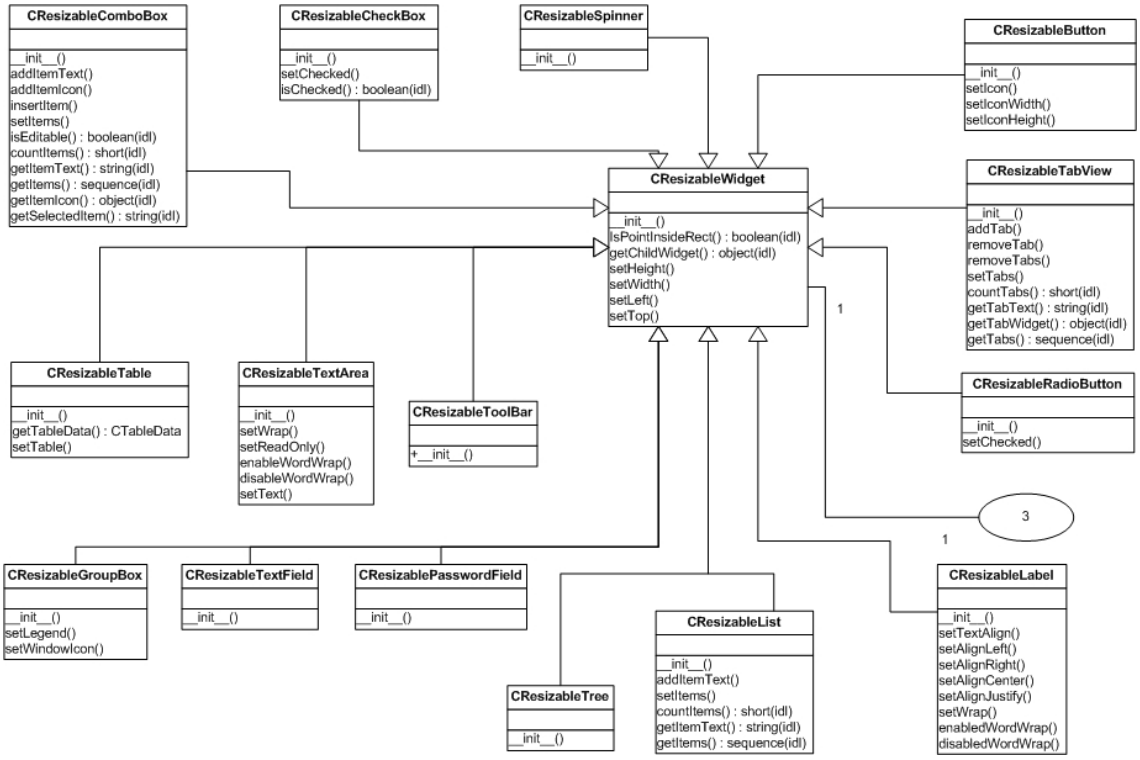

Figura 3.4 – Diagrama de Classes (Parte 4)

3.2 Diagramas de Transição

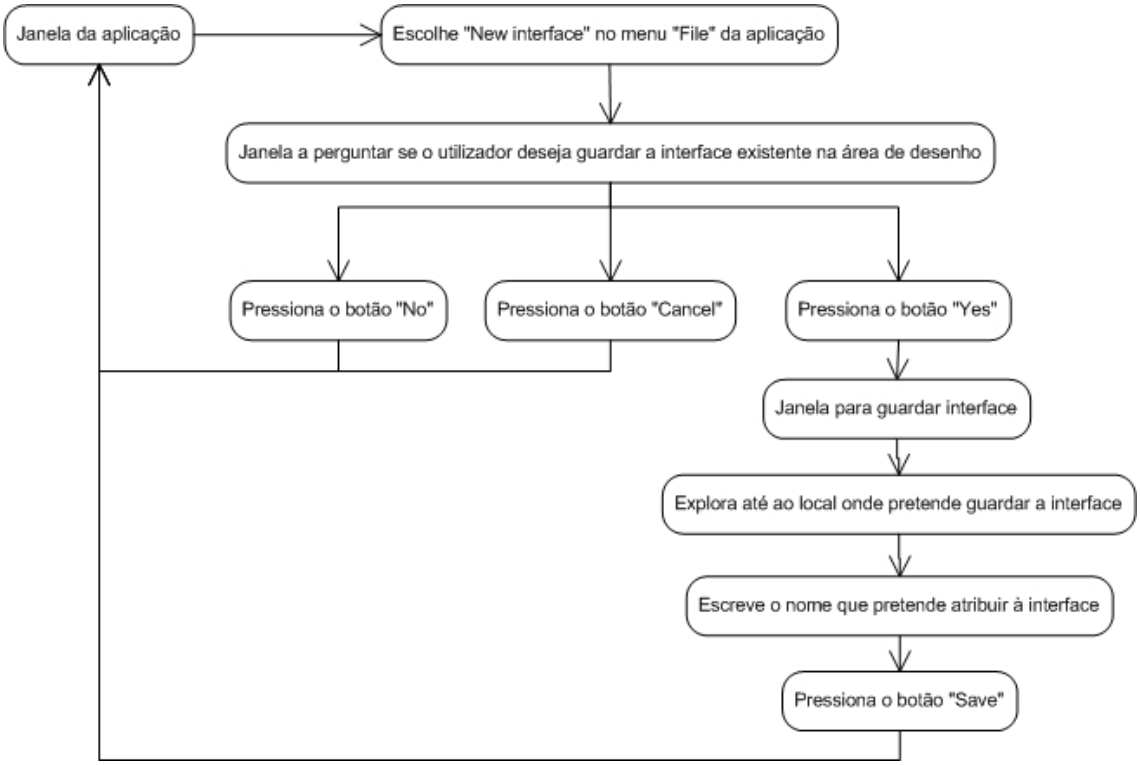

Figura 3.5 – Diagrama de Transição referente ao Caso de Uso "Criar nova interface"

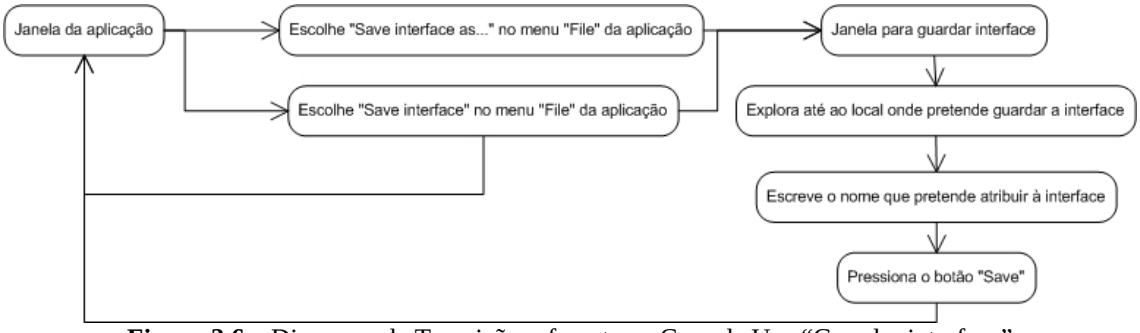

Figura 3.6 – Diagrama de Transição referente ao Caso de Uso "Guardar interface"

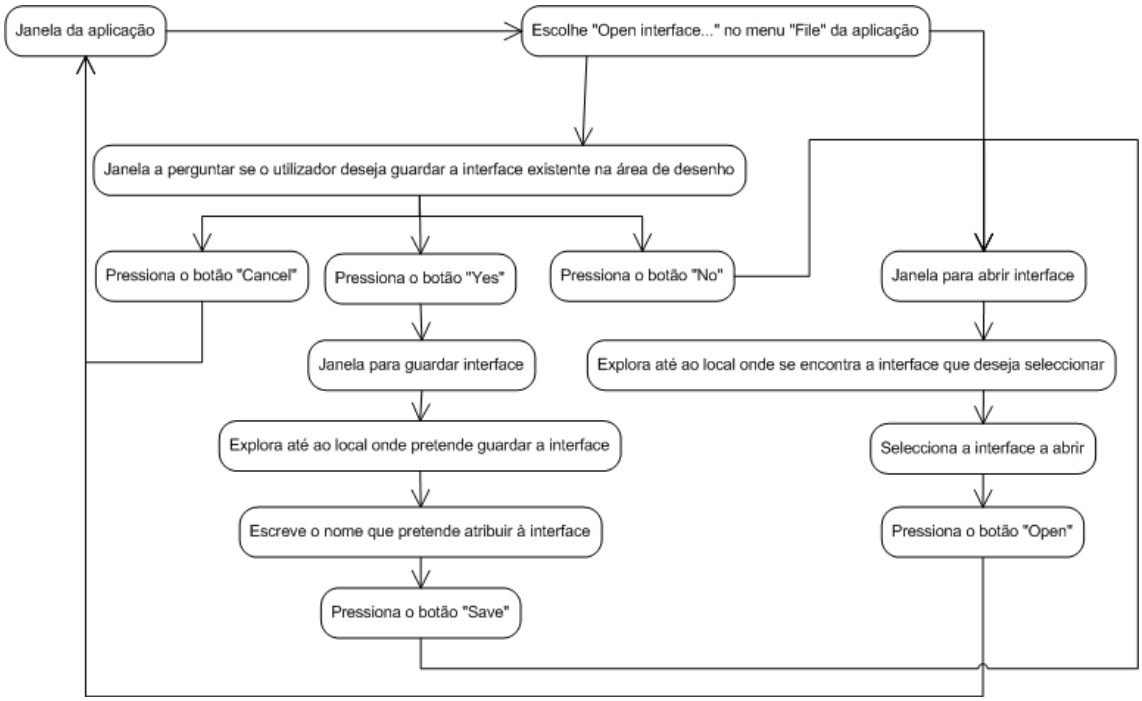

Figura 3.7 – Diagrama de Transição referente ao Caso de Uso "Abrir interface"

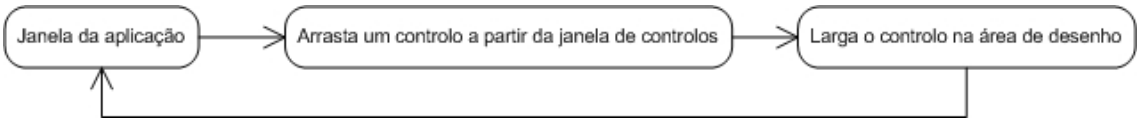

Figura 3.8 – Diagrama de Transição referente ao Caso de Uso "Inserir controlo visual"

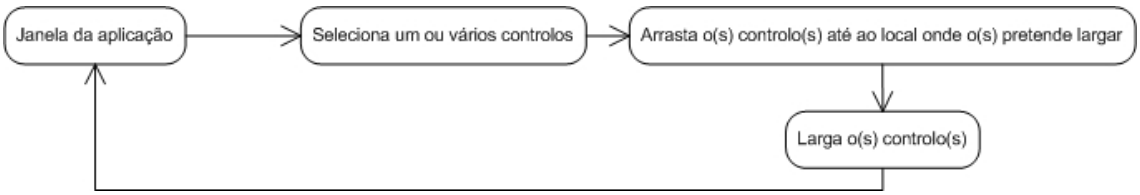

Figura 3.9 – Diagrama de Transição referente ao Caso de Uso "Mover controlo visual"

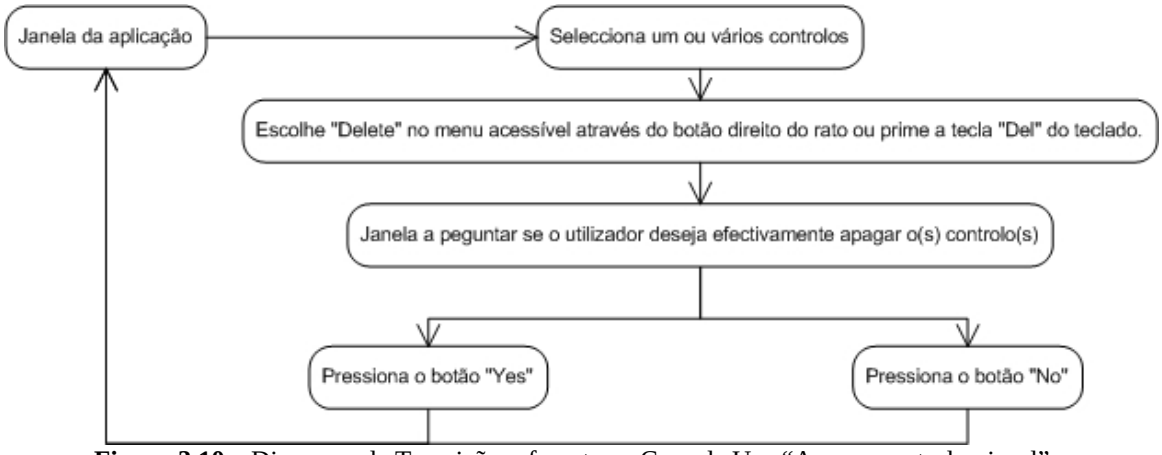

Figura 3.10 – Diagrama de Transição referente ao Caso de Uso "Apagar controlo visual"

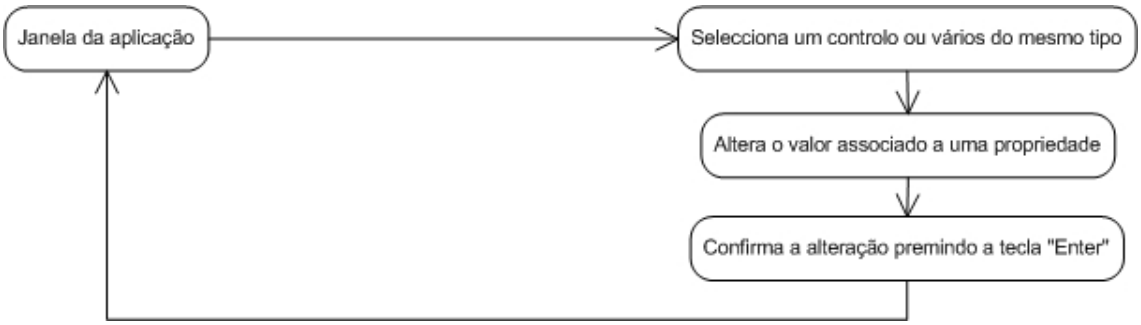

Figura 3.11 – Diagrama de Transição referente ao Caso de Uso "Formatar controlo visual"

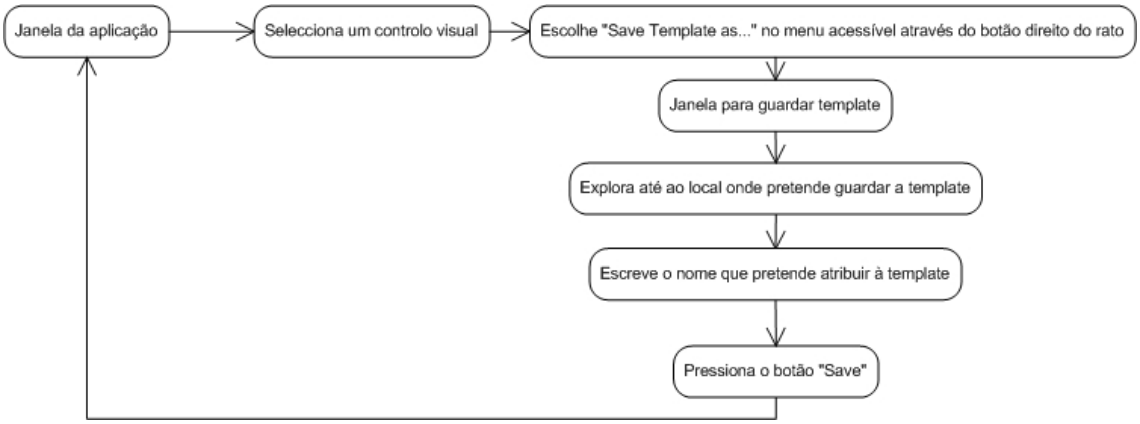

Figura 3.12 – Diagrama de Transição referente ao Caso de Uso "Guardar template de controlo visual"

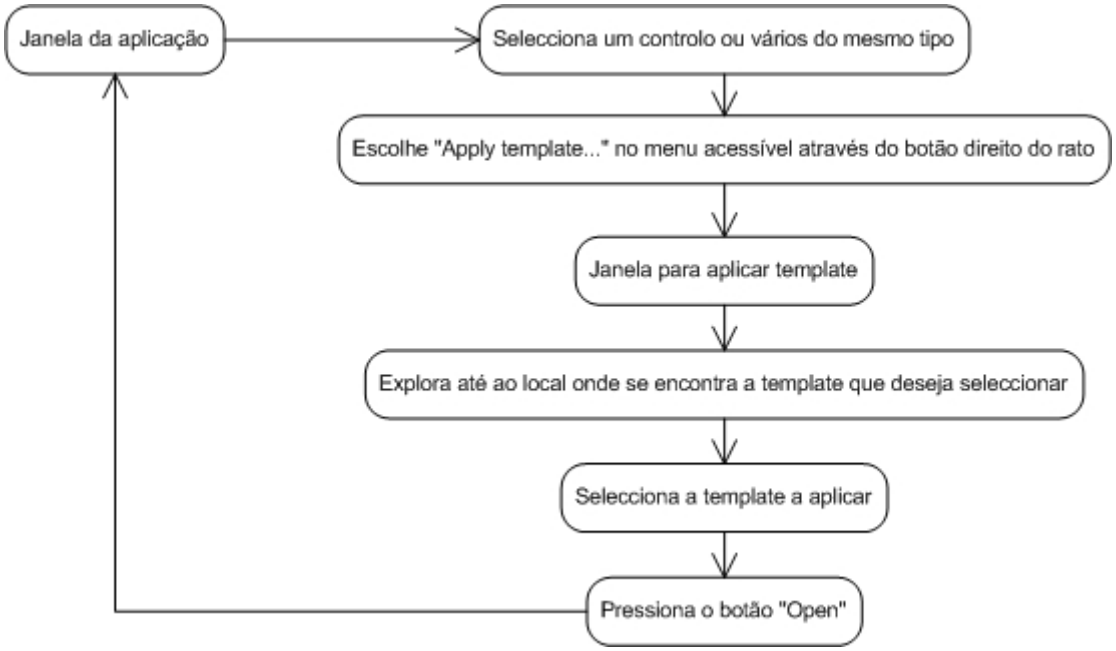

Figura 3.13 – Diagrama de Transição referente ao Caso de Uso "Aplicar template sobre controlo visual"

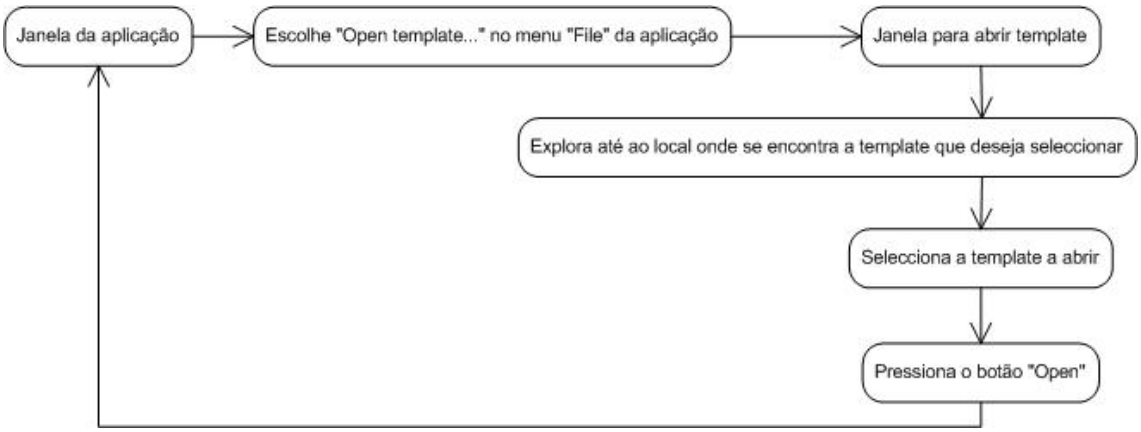

Figura 3.14 – Diagrama de Transição referente ao Caso de Uso "Abrir template de controlo visual"

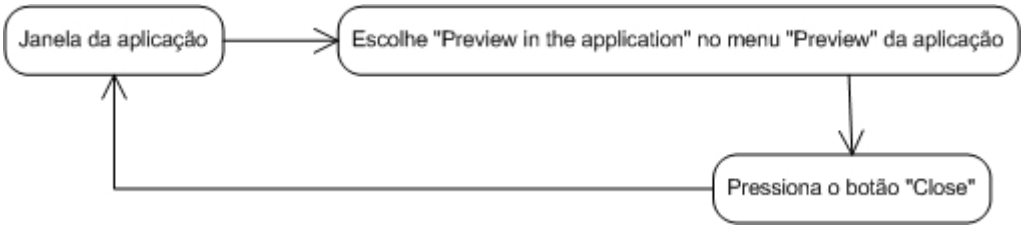

Figura 3.15 – Diagrama de Transição referente ao Caso de Uso "Pré-visualizar interface na aplicação"

Figura 3.16 – Diagrama de Transição referente ao Caso de Uso "Pré-visualizar interface num *browser*"

3.3 Desenho e avaliação de interfaces

3.3.1 Protótipos de baixa fidelidade

Para maximizar a futura interacção entre o utilizador e aplicação foi desenvolvido um conjunto de protótipos de baixa fidelidade. Os protótipos de baixa fidelidade são constituídos por esboços, criados a lápis e papel, das interfaces gráficas da aplicação que ilustram as suas principais funcionalidades, fornecendo uma visão geral do funcionamento do componente a desenvolver. Neste processo é possível verificar uma clara evolução no desenho das interfaces, de acordo com todo um processo de refinação que foi efectuado aos vários protótipos criados. Foram então criadas as seguintes versões de protótipos de baixa fidelidade:

Versão inicial

Figura 3.17 – Primeiro esboço da interface gráfica da janela da aplicação

Versão refinada

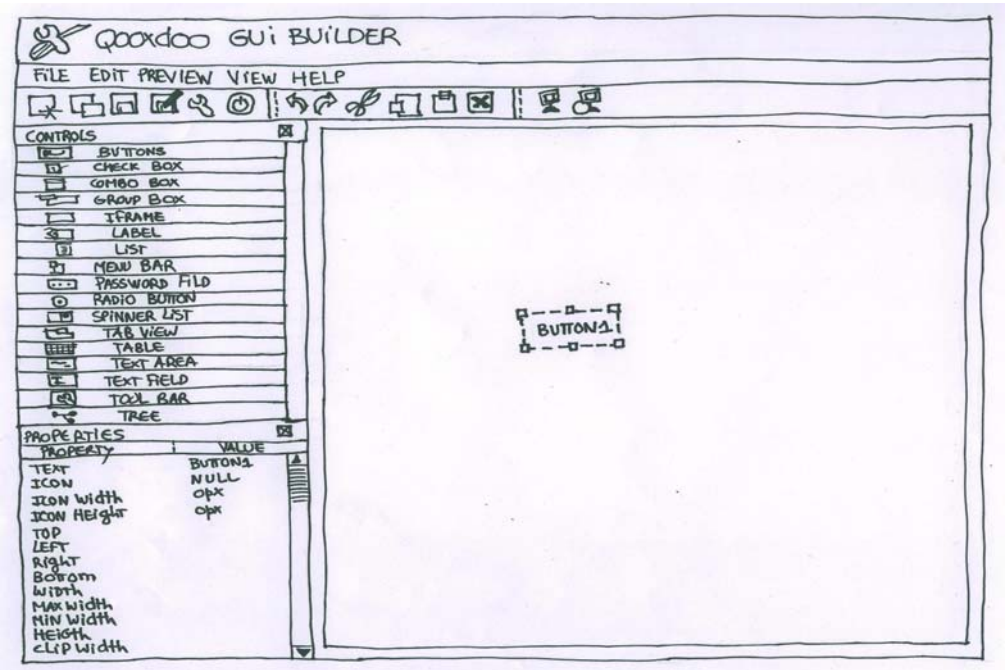

Figura 3.18 – Esboço refinado da interface gráfica da janela da aplicação

Esboços das restantes interface gráficas que constituem a aplicação:

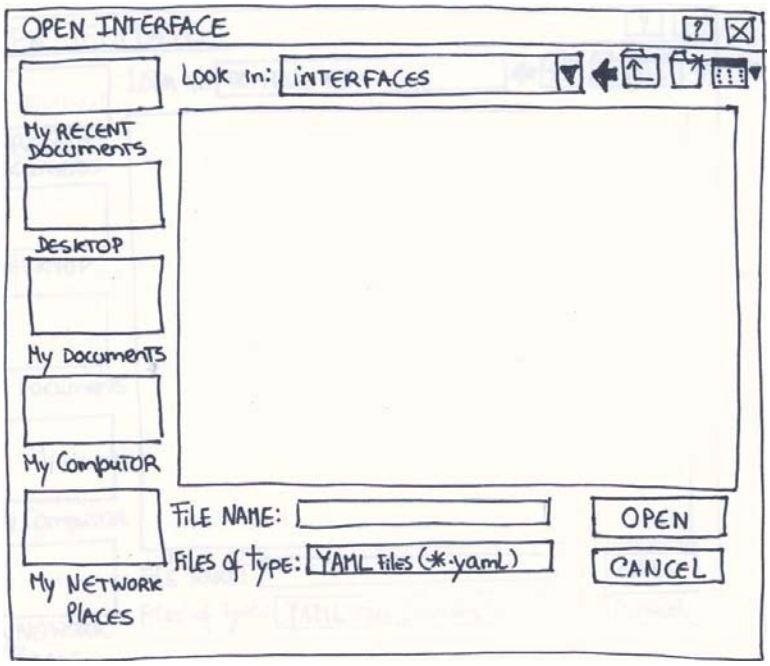

Figura 3.19 – Janela para abrir interface

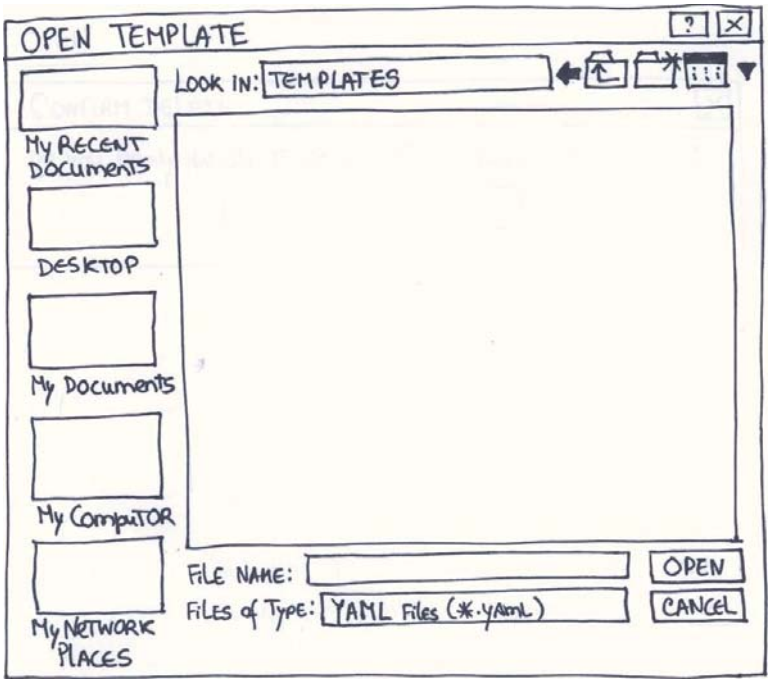

Figura 3.20 – Janela para abrir template

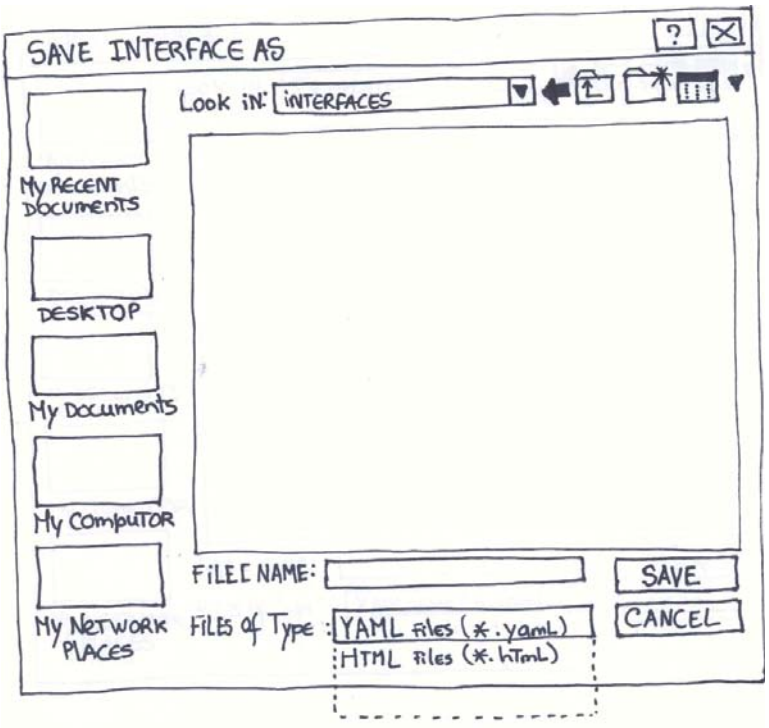

Figura 3.21 – Janela para guardar interface

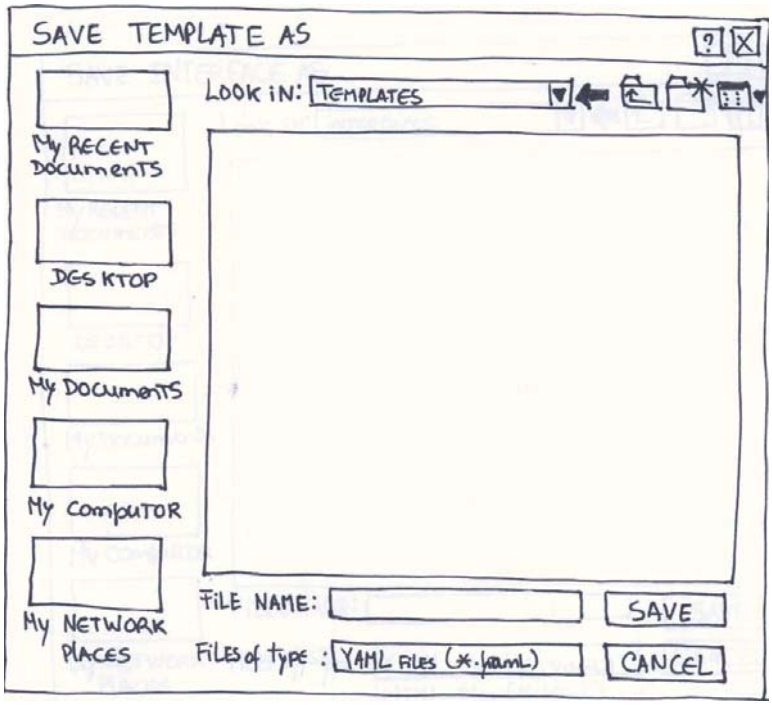

Figura 3.22 – Janela para guardar template

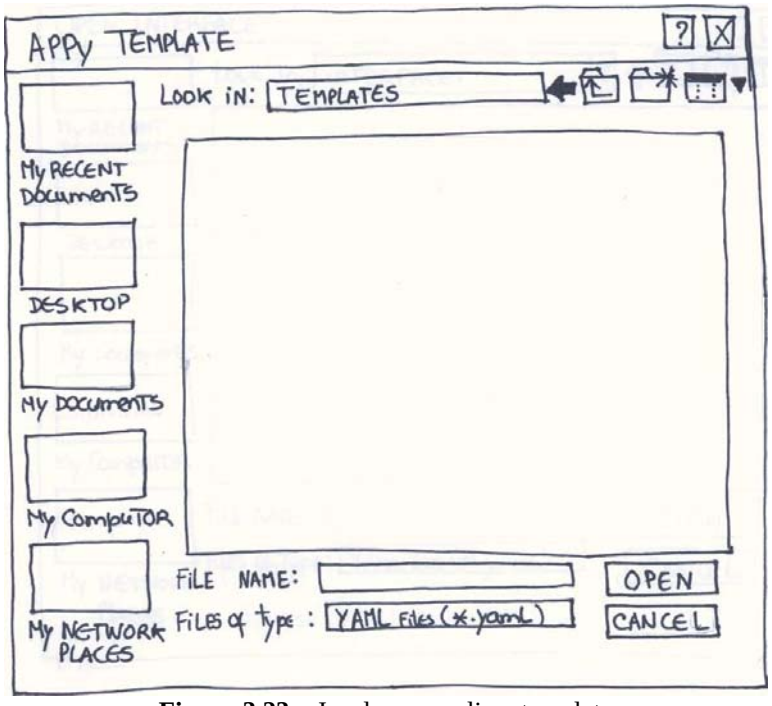

Figura 3.23 – Janela para aplicar template

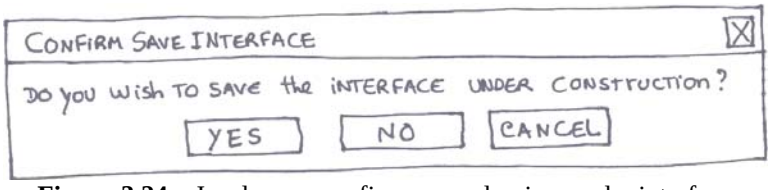

Figura 3.24 – Janela para confirmar que deseja guardar interface

Figura 3.25 – Janela para confirmar que deseja realmente apagar o(s) controlo(s) seleccionado(s)

3.3.2 Avaliação de usabilidade das interfaces

A usabilidade de uma interface é determinada através de um conjunto de práticas de análise sistemática sobre a interacção que o homem tem com uma interface gráfica, na sua componente comunicacional. Este conjunto de práticas leva à criação de interfaces gráficas com maior usabilidade onde todo um processo de refinação terá de ser processado pelo elemento que constrói as interfaces gráficas. Este processo de refinação é complementado por avaliações que terão de ser realizadas sobre as interfaces. Existem vários tipos de avaliações, todas elas com o objectivo de estimar o estado de usabilidade das interfaces, confrontando-as com diversas regras e heurísticas, do que resulta uma classificação baseada no cumprimento dessas regras. A avaliação realizada baseou-se nas dez heurísticas de Nielsen [11].

O primeiro dos dois processos de avaliação que se seguem foi realizado sobre a **versão inicial** da interface gráfica da janela da aplicação**,** que aparece representada na figura 3.17. O outro foi realizado posteriormente, sobre a **versão refinada** da mesma interface gráfica**,** que aparece representada na figura 3.18. A **versão refinada** foi esboçada após o processo de avaliação sobre a **versão inicial.**

Tabela 3.1 – Avaliação sobre a versão inicial

Viola	Grau	Problema	Solução
Visibilidade do estado do sistema	4	Nem todos os controlos visuais, para construção das interfaces, estão visíveis no estado inicial da interface tendo que o utilizador para isso deslocar a *scrollbar* horizontal.	Dispor os possíveis controlos visuais para construção das interfaces numa posição vertical, de forma a ser mais fácil o seu acesso.
Ligação entre o sistema e o mundo real	5	Os controlos visuais, para construção das interfaces, não contêm qualquer tipo de imagem metafórica associada que permita uma mais fácil percepção do utilizador sobre o tipo de controlo representado.	Dispor para cada tipo de controlo uma imagem associada de forma que seja facilmente identificado o tipo de controlo a utilizar.
Liberdade e controlo por parte do utilizador	5	A estrutura de navegação não é muito concisa na sua forma de interacção, dada a disposição das várias secções da interface.	• Posicionar a área de desenho no lado direito da interface; • Posicionar os tipos de controlos e secção das suas propriedades do lado esquerdo da interface.
Consistência e standards	4	Os acessos ás várias opções de são realizados através dos menus “File”, “Edit”, “Preview” e “Help”, não sendo prevista qualquer tipo de atalho que ajude na consistência do manuseamento.	Dispor na interface um conjunto de atalhos, através de uma barra de menus, onde seja possível realizar várias operações.
Prevenção de erros	5	Não existe sobre qualquer forma visual a possibilidade de utilizador recuperar erros (ex: apagar um controlo, alterar a sua posição, etc.).	Prever botões de Desfazer (*Undo*) e Refazer (*Redo*), para a prevenção de erros associados à utilização da aplicação.
Reconhecimento em vez de o lembrar	4	Não existindo atalhos para as várias funcionalidades da aplicação, o reconhecimento não é realizado facilmente onde provavelmente o utilizador terá de navegar pela aplicação para encontrar o ponto de acesso às funcionalidades.	• Dispor na interface um conjunto de atalhos, através de uma barra de menus, onde seja possível realizar várias operações; • Dispor a interface de forma a que as

			acções do utilizador no acesso às funcionalidades fluíam instintivamente.
Flexibilidade e eficiência de utilização	-	Esta heurística não é violada porque a aplicação possui atalhos através de combinações de teclas, que permitem aos utilizadores mais avançados uma maior eficiência na utilização da aplicação.	-
Desenho minimalista e estético	-	Esta heurística não é violada pois todos os diálogos possuem a informação necessária ao bom funcionamento do programa e uma estética consistente.	-
Ajuda no reconhecimento, diagnóstico e recuperação de erros	3	Não estão previstas mensagens de erros associadas às acções do utilizador.	Devem ser esboçadas mensagens de erro associadas aos principais tipos de funcionalidades da aplicação.
Ajuda e documentação	-	No que se refere a esta heurística, devido ao facto de não possuirmos informação nos protótipos referentes a esta área, não podemos avaliar a sua validade.	-

Os esboços do protótipo de baixa fidelidade da **versão refinada** e das restantes interface gráficas que constituem a aplicação foram realizados de acordo com avaliação feita sobre a **versão inicial.** Contudo foi realizada uma nova avaliação para comprovar a usabilidade das mesmas confrontando as interfaces com as heurísticas.

Tabela 3.2 – Avaliação sobre a versão refinada

Viola	Grau	Problema	Solução
Visibilidade do estado do sistema	-	Esta heurística não é violada dado que todas as informações relativas ao estado da utilização da aplicação estão identificadas.	-
Ligação entre o sistema e o mundo real	-	Esta heurística não é violada, pois o vocabulário utilizado se adequa ao tipo de utilizadores finais e todos os controlos de interface são acompanhados de um ícone metafórico sugestivo, o que a torna bastante perceptível as suas funcionalidades.	-
Liberdade e controlo por parte do	-	Esta heurística não é violada porque pelos protótipos apresentados, o utilizador parece	-

utilizador		ter total controlo sobre o programa, uma vez que nunca chega a sair da janela da aplicação.	
Consistência e standards	-	Esta heurística não é violada porque a aplicação foi idealizada com base em estudos realizados a partir de aplicações do mesmo tipo.	-
Prevenção de erros	-	Esta heurística não é violada uma vez que existem na aplicação diversos mecanismos que permitem prevenir erros.	-
Reconhecimento em vez de o lembrar	-	Esta heurística não é violada, porque praticamente todas as funcionalidades da aplicação se encontram centradas numa janela da aplicação.	-
Flexibilidade e eficiência de utilização		Esta heurística não é violada porque a aplicação possui atalhos através de combinações de teclas, que permitem aos utilizadores mais avançados uma maior eficiência na utilização da aplicação.	
Desenho minimalista e estético	-	Esta heurística não é violada pois todos os diálogos possuem a informação necessária ao bom funcionamento do programa e uma estética consistente.	-
Ajuda no reconhecimento, diagnóstico e recuperação de erros	-	A partir da informação transmitida pelos protótipos, apenas podemos afirmar que esta heurística não é violada no que diz respeito a recuperação de erros.	-
Ajuda e documentação	-	No que se refere a esta heurística, devido ao facto de não possuirmos informação nos protótipos referentes a esta área, não podemos avaliar a sua validade.	-

4 Módulos de implementação

Após a especificação dos requisitos e a modelação do sistema, seguiu-se a fase de implementação, que consistiu na codificação de vários módulos de implementação, utilizando as tecnologias anteriormente apresentadas.

De uma forma geral, a implementação deste sistema dividiu-se em quatro módulos de implementação:

- Recolha de informação sobre os controlos visuais a disponibilizar ao utilizador, para a construção das interfaces;
- Construção da interface gráfica da aplicação e implementação de mecanismos para interacção com os controlos visuais disponíveis;
- Construção de um interpretador e gerador de código YAML, que serve de base ao armazenamento e carregamento das interfaces e templates criadas pelo utilizador;
- Construção de um gerador de código HTML que incluirá código JavaScript onde estará codificada a interface qooxdoo.

De seguida, apresenta-se de uma maneira geral, a forma como os módulos foram implementados.

4.1 Controlos visuais

4.1.1 Selecção dos controlos visuais

Foi necessário efectuar um estudo para decidir quais os controlos que iriam ser disponibilizados na aplicação para a construção de interfaces por parte do utilizador. Como a interface da aplicação foi desenvolvida recorrendo ao *toolkit* gráfico Qt, foi necessário estabelecer termos de comparação entre os controlos visuais disponíveis em qooxdoo e em Qt, tendo para isto sido consultadas as documentações de ambas as tecnologias.

De entre os controlos disponíveis em qooxdoo que tinham controlos similares em Qt, foi seleccionado o seguinte conjunto de controlos visuais para a construção das interfaces e atribuída uma identificação única:

Tabela 4.1 – Controlos visuais disponíveis e respectivos IDs

Controlo	ID	Controlo	ID	Controlo	ID
Button	BTN	Check Box	CKB	Combo Box	CMB
Group Box	GRB	IFrame	IFR	Label	LBL

List	LST	Menu Bar	MBR	Password Field	PWF
Rádio Button	RDB	Spinner	SPR	TabView	TBV
Table	TBL	Text Area	TXV	Text Field	TXF
Tool Bar	TLB	Tree	TRE		

Para que o utilizador tenha a possibilidade de formatar estes controlos visuais, foi necessário recolher as propriedades que estão associadas a cada um dos controlos, de acordo com a documentação qooxdoo, estabelecendo sempre que possível, relação com as propriedades associadas em Qt. Foi então recolhido um conjunto de propriedades gerais, partilhadas entre todos os controlos escolhidos, e um conjunto de propriedades específicas, para cada um dos controlos visuais.

4.1.2 Estruturação da informação

Toda a informação recolhida relativamente aos controlos visuais está estruturada em ficheiros de texto com extensão ".dat", de acordo com a seguinte estrutura:

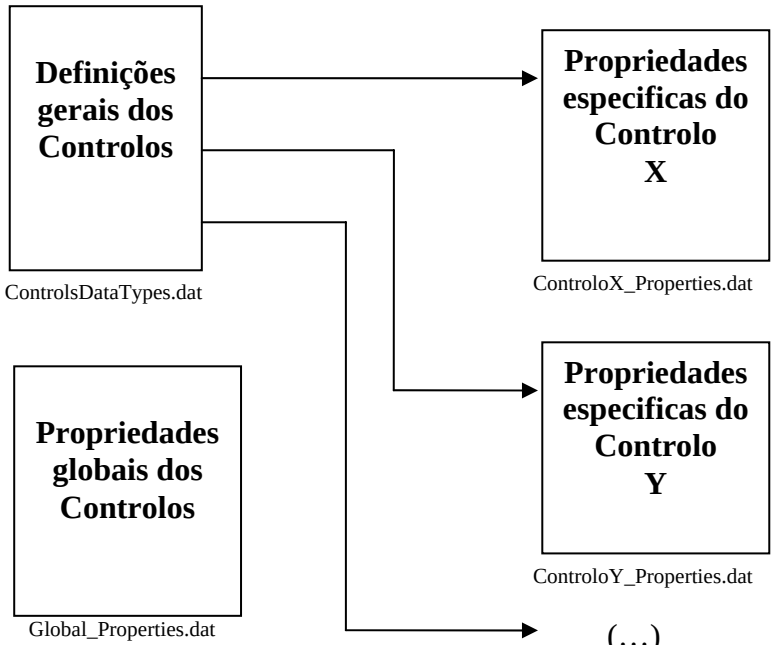

Figura 4.1 – Estrutura física de dados que guarda informação relativa a controlos visuais

- **Definições gerais dos controlos** – ControlsDataTypes.dat

Neste ficheiro são referenciados os controlos que a aplicação vai disponibilizar ao utilizador. Através de um conjunto de palavras reservadas, que identificam cada um dos controlos, são armazenadas as seguintes informações de acordo com o seguinte formato:

```
<ID do Controlo>:<Ficheiro com as propriedades específicas>:<Construtor da classe que representa o controlo visual na aplicação>
```

Exemplo da estruturação da informação para os controlos TextField e ComboBox:

```
TXF:TextField_Properties.dat:ResizableTextField(typeControl,id,parent)
CMB:ComboBox_Properties.dat:ResizableComboBox(typeControl,id,parent)
```

- **Propriedades especificas de um controlo** – ControloX_Properties.dat

Para cada um dos controlos visuais, existe um conjunto de propriedades específicas, as quais são referenciadas num ficheiro de dados, identificado com o nome do controlo visual. Tem-se então o seguinte formato:

```
<ID da propriedade>:<Designação da propriedade>:<Valores que a propriedade pode tomar>:<Método a executar na classe que representa o controlo para a alteração visual da propriedade>:<Tipo de dados da propriedade>
```

Tabela 4.2 – Tipos de dados e sua descrição

Tipos de dados	**Descrição**
TInt	Tipo Inteiro
TString	Tipo Cadeia de caracteres
TBoolean	Tipo Booleano
TItems	Representa os *items* de um controlo com esta propriedade. (Ex: *List*, *ComboBox*)
TTabs	Representa as *tabs* do controlo *TabView*.
TTableItems	Representa os *items* do controlo *Table.*

Exemplo da estruturação da informação para o controlo TextField (Ficheiro: TextField_Properties.dat):

```
01:Text:Text Field:setText(param):TString
02:MaxLength:100:-:TInt
03:ReadOnly:false/true:-:TBoolean
```

- **Propriedades gerais dos controlos** – Global_Properties.dat

Neste ficheiro é referenciado todo um conjunto de propriedades que são “globais” a todos os controlos. O formato deste ficheiro é semelhante ao formato dos ficheiros com as propriedades específicas dos controlos, onde como distinção, a identificação das propriedades é iniciada com a letra “G”, que se refere a Global. Tem-se então o seguinte exemplo:

```
G01:Top:0:setTop(param):TInt
G02:Left:0:setLeft(param):TInt
G03:Right:10:-:TInt
(…)
```

De notar, que todas as constantes que fazem parte dos ficheiros são identificadas pela aplicação através de um ficheiro de constantes implementado, designado “const.py”.

Os valores das propriedades que constam nestes ficheiros, serão atribuídos às propriedades dos controlos visuais, quando estes forem criados.

Este conjunto de informação é processado e colocado em memória pela aplicação, a partir de uma instância da classe CMonitorControls, a qual gere e mantém toda a informação relativa aos controlos visuais, funcionando como ponte de interacção entre a interface da aplicação e a gestão dos controlos visuais.

De referir que qualquer actualização de informação sobre os controlos poderá ser realizada sobre estes ficheiros, as quais terão efeito na interacção com os controlos visuais durante a execução da aplicação, tornado o sistema flexível a qualquer tipo de actualização sobre as características dos controlos que possam ser realizadas no futuro.

4.1.3 Representação dos controlos visuais

Cada controlo visual é representado por uma classe específica, que permite a interacção com a aplicação, na construção visual das interfaces e a alteração das propriedades visuais.

Sempre que o utilizador adiciona um controlo visual à área de desenho, é gerada uma instância destas classes e aquando da ocorrência de operações visuais sobre os controlos, serão executados métodos para que as alterações a nível visual sobre os controlos ocorram.

Como modelado na fase de desenho, foram implementadas as seguintes classes que representam os controlos visuais:

- CResizableTextField
- CResizableComboBo
- CResizableList
- CResizableTable
- CResizableButton
- CResizableCheckBox
- CResizableGroupBox
- CResizableLabel
- CResizablePasswordField
- CResizableRadioButton
- CResizableSpinner
- CResizableTabView
- CResizableTextArea
- CResizableToolBar
- CResizableTree

Cada uma destas classes implementa uma interface visual específica para um controlo visual e suas propriedades.

Foi ainda construída a classe CResizableWidget, que implementa um conjunto de funcionalidades globais a todos os controlos visuais. Por este facto, todas as classes acima referidas, herdam as propriedades da classe CResizableWidget.

4.2 Interacção entre a interface gráfica e os controlos

A interface gráfica da aplicação foi desenvolvida com base nos protótipos de baixa fidelidade desenhados durante a fase de modelação do sistema.

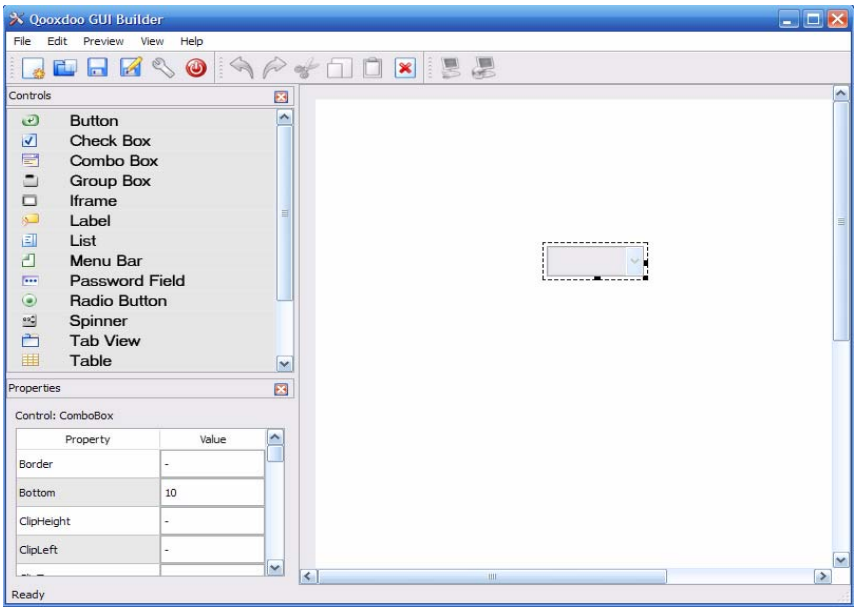

Figura 4.2 – Interface gráfica da aplicação

Uma das funcionalidades desta interface é permitir ao utilizador construir uma interface visual a partir de um conjunto de controlos visuais disponíveis, com a possibilidade de efectuar um conjunto de formatações sobre esses mesmos controlos.

4.2.1 Plataforma de gestão dos controlos visuais

Toda a interacção possível entre o utilizador e os controlos visuais é controlada por uma plataforma intermediária que faz a gestão dos controlos visuais. Esta plataforma está implementada na classe CMonitorControls, onde é disponibilizado um conjunto de métodos que possibilita a criação e eliminação de controlos visuais, e a leitura e alteração de valores das propriedades, entre outros.

Quando a interface gráfica é iniciada, é carregada a informação sobre controlos visuais e suas propriedades para memória, sob a forma de estruturas de dados definidas internamente, a partir dos ficheiros de propriedades referidos acima.

Adição de controlos

Quando o utilizador adiciona um controlo visual à área de desenho, através da instância da classe CMonitorControls, é criada uma instância de uma classe Resizable que representa o controlo visual. Esta instância é então retornada à MainWindow, que posiciona na área de desenho a representação do controlo visual. A informação do controlo adicionado é armazenada numa estrutura de dados interna da classe CMonitorControls, para que possa ser mantida durante a existência do controlo na área de desenho.

Eliminação de controlos

Quando o utilizador elimina um controlo da área de desenho, a informação armazenada sobre o controlo visual será eliminada da estrutura de dados e por conseguinte a referência será eliminada o que fará com que o controlo seja eliminado da interface gráfica.

Alteração das propriedades dos controlos

Inicialmente os controlos visuais são formatados com as propriedades por defeito que constam nos ficheiros que armazenam as propriedades dos mesmos. Sempre que o utilizador altera uma propriedade de um controlo visual, esta é armazenada na estrutura de dados, através da instância da classe CMonitorControls. Caso a propriedade alterada tenha um efeito visual associado para ser apresentado, será executado um método da instância que representa o controlo, de forma a concretizar-se esse efeito.

4.2.2 Mecanismos de comunicação

A interacção entre o utilizador e os controlos visuais, tem que ser processada de forma a ser apresentado e actualizado todo um conjunto de informação.

Temos então vários tipos de acções possíveis sobre um controlo, das quais tem de resultar um efeito:

Tabela 4.3 – Acções sobre controlos e seus efeitos

Acção	Efeito resultante
Pressionar controlo visual	• Apresentar as propriedades do controlo para que o utilizador as possa alterar; • Apresentar uma zona sobre o controlo que permita identificar que o controlo foi seleccionado e redimensionar.
Alterar propriedade visual	• Armazenar a informação da propriedade alterada; • Actualizar visualmente a informação da propriedade alterada; • Actualizar visualmente o controlo de acordo com a propriedade alterada.
Salvar controlo como template. (“Save as	• Terá de ser disponibilizado uma interface ao utilizador para escolher o directório de armazenamento da template do

template…”)	controlo visual; • Terá de ser criado o *output* para ficheiro, da template do controlo visual.
Aplicar template sobre controlo. (“Apply template…”)	• Terá de ser disponibilizada uma interface ao utilizador para escolher o ficheiro de template; • Terá de ser interpretado o conteúdo da template e sobre o controlo visual aplicadas as formatações contidas na template.

Todas estas acções podem ter origem em diferentes instâncias de classes criadas e são processadas em outras instâncias. Para que fosse possível processar estes eventos foi necessário utilizar o mecanismo de sinais implementado pelo *toolkit* Qt - signal, funcionando como ponte de ligação entre as várias instâncias das classes que intervêm nestes pares acção-efeito.

Envio de Sinais – Acção

Por exemplo, ao pressionar um controlo visual é enviado um sinal da seguinte forma:

```
self.emit(QtCore.SIGNAL(SIGNAL_RESIZABLE_CLICKED), typeControl, idControl)
```

Onde:

```
SIGNAL_RESIZABLE_CLICKED = "Resizable_Clicked(const QString &, const QString &)"
```

Assim, sempre que um controlo é pressionado, é enviado um sinal com duas propriedades associadas, tipo de controlo e ID do controlo, que identificam o controlo que foi seleccionado.

Recepção e processamento de Sinais – Efeito resultante

Para que a acção concretizada seja processada é necessário identificar o sinal que é emitido, na classe onde o queremos processar, com a indicação da SLOT que tratará o sinal.

Neste exemplo, de pressionar um controlo, a acção é processada na MainWindow, onde é necessário apresentar as propriedades do controlo seleccionado. Temos então o seguinte código para a identificação do sinal e indicação da SLOT de processamento:

```
QtCore.QObject.connect(controlWidget,          QtCore.SIGNAL(SIGNAL_RESIZABLE_CLICKED),
self.SignalProcess_resizableClicked)
```

Onde a SLOT SignalProcess_resizableClicked está definida da seguinte forma:

```
def SignalProcess_resizableReleased(self, typeControl, idControl):
(…)
```

Podemos verificar que esta SLOT recebe por parâmetro duas referências para o tipo de dados *String*, as quais identificam o controlo seleccionado. A SLOT é então processada, de acordo com o código associado, e as propriedades são apresentadas. Desta mesma forma, são processados os restantes sinais emitidos.

4.3 Interpretador e gerador de código YAML

Uma das necessidades mais recorrentes dos utilizadores é a possibilidade de armazenar o seu trabalho, para posteriormente ser consultado ou alterado.

Na aplicação desenvolvida o utilizador poderá criar as suas interfaces visuais e armazenar estas numa unidade de armazenamento à sua escolha, e posteriormente poderá visualizá-las e aplicar-lhes alterações.

Da mesma forma poderão ser criadas templates a partir de controlos visuais, onde ficam armazenados os valores associados às propriedades dos mesmos, para posteriormente serem utilizadas na formatação de controlos visuais do mesmo tipo, de acordo com a template.

4.3.1 Codificação YAML

O armazenamento em suporte das interfaces visuais e das templates é realizado através de uma codificação estruturada de informação, que representa interfaces ou templates, através da linguagem de codificação YAML.

Para a realização desta codificação foi necessário encontrar um *package*[12], para a linguagem de programação Python, que implementasse a leitura e escrita de código YAML. Da pesquisa realizada foi escolhido o *package* PyYAML, versão 3.04, o qual preenche todos os requisitos funcionais para a implementação desta funcionalidade, de interpretação e geração de código YAML.

Foi implementada a classe CYamlInterpretorGenerator que implementa um conjunto de métodos cuja função é interpretar e gerar código YAML, servindo de ponde de ligação entre a aplicação e o *package* PyYAML, na construção formatada do código YAML.

4.3.2 Formato dos ficheiros YAML

Sendo a linguagem YAML flexível na construção de estruturas de codificação, foi necessário estudar qual o formato mais indicado para a codificação das informações que compõem as interfaces visuais e as templates dos controlos visuais.

Interfaces visuais

Para as interfaces visuais foram identificadas quais as informações que deveriam ser codificadas na linguagem YAML. Tem-se então os tipos de controlos existentes na interface, os quais são identificados por um ID relacionado com cada um dos tipos controlos, e tem-se as propriedades de cada um destes controlos, onde é necessário armazenar as suas identificações e valores associados.

[12] Mecanismo de propósito geral para a organização de elementos em grupos.

O formato do ficheiro YAML é o seguinte:

```
- <ID Controlo Visual X> :
        {
                ‘<ID Propriedade A>‘ : ‘<Valor Propriedade> ‘,
                ‘<ID Propriedade B>‘ : ‘<Valor Propriedade> ‘,
                (…)
        }
- <ID Controlo Visual Y> :
        {
                ‘<ID Propriedade A>’ : ‘<Valor Propriedade>‘,
                ‘<ID Propriedade B>’ : ‘<Valor Propriedade>‘,
                (…)
        }
(…)
```

O excerto seguinte ilustra o formato do ficheiro YAML, de uma interface visual armazenada em memória, que contém as informações de um controlo do tipo Lista e de um controlo do tipo Botão:

```
- LST: {
                '01': 'true',
                '02': 'false',
                (…)
        }
- BTN: {
                '01': Button,
                (…)
       }
```

A extensão deste tipo de ficheiro em disco é **“.ymli”.**

Para a apresentação da interface visual armazenada neste ficheiro é necessário utilizar um método da classe CYamlInterpretorGenerator, que retornará um conjunto de informações que serão posteriormente interpretadas na instância da classe CMonitorControls, a qual criará uma interface de acordo com a informação interpretada a partir da área de desenho.

Para o armazenamento num ficheiro, é executado um método da classe CMonitorControls, que retorna a informação dos controlos que compõem a interface visual, e a partir desta será gerado o código YAML, através de um método da classe CYamlInterpretorGenerator.

Templates dos controlos visuais

Para as templates dos controlos visuais as informações que são necessárias armazenar são somente a identificação do controlo visual e as suas propriedades. Assim o formato YAML é semelhante ao formato das interfaces visuais diferindo unicamente no facto de que um ficheiro de uma template irá armazenar apenas informação de um controlo visual e suas propriedades.

Temos então o excerto seguinte de um ficheiro de template, do tipo de controlo botão:

```
BTN: {
                '01': Button,
                '02': '-',
                G01: '165',
                G02: '145',
                (…)
    }
```

A extensão deste tipo de ficheiro em disco é **“.ymlt”.**

4.4 Gerador de código HTML

O gerador de código HTML é o “motor” que proporciona ao utilizador a visualização da interface, num *browser*. O código HTML gerado vai incluir uma porção de código JavaScript, onde, através de comandos qooxdoo, se encontrará descrita a interface criada.

Este gerador é composto por quatro funções fundamentais, que trabalham entre si de forma hierárquica e conjunta de forma a gerar todo o código necessário. Essas funções são:

- generateHTML:
  - Função por onde se inicia o processo e que retorna o produto final sob a forma de uma *string*, que irá ser o conteúdo de um ficheiro HTML;
  - Gera o código HTML que “rodeia” a porção JavaScript/qooxdoo do documento;
  - A parte JavaScript/qooxdoo provém da função generateJS;
  - Retorna o que gera e o que obtém, para uma instância sua, criada dentro da classe MainWindow;
- generateJS:
  - Função que retorna a porção JavaScript/qooxdoo do código HTML gerado, embora vá buscar grande parte desse código à função generateControlsJS;
  - Retorna o que gera e o que obtém, para a função generateHTML;
- generateControlsJS:
  - É nesta função que são tratados os controlos que fazem parte da interface existente na área de desenho;
  - Após identificar o tipo de controlo que está a tratar, adiciona à *string* que irá retornar, o código JavaScript/qooxdoo que gera o mesmo, e recorre à função customizeControlJS que lhe retorna o código necessário para formatar o controlo;

- É ainda nesta função que os controlos são adicionados ao documento qooxdoo, através de linhas de comando como a seguinte: “button1.addToDocument();”;
- Retorna o que gera e o que obtém, para a função generateJS;

- customizeControlJS:
  - É nesta função que se gera o código que formata um determinado controlo de acordo com a apresentação do mesmo na área de desenho;
  - Retorna o que gera para a função generateControlsJS.

De seguida, exemplifica-se a produção deste gerador.

```
<html>
	<head>
		<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
		<title>Exemplo</title>
		<script type="text/javascript" src="script/qx.js"></script>
		<script type="text/javascript">
			qx.OO.defineClass("Exemplo",	qx.component.AbstractApplication,
function(){qx.component.AbstractApplication.call(this);});
			qx.Proto.main = function(e)
			{
				var btn0 = new qx.ui.form.Button("Pressione-me");
				btn0.setRight(10);
				btn0.setBottom(10);
				btn0.setTop(10);
				btn0.setLeft(10);
				btn0.addToDocument();
			};
		</script>
	</head>
	<body>
		<script type="text/javascript">
			qx.core.Init.getInstance().setApplication(Exemplo);
		</script>
	</body>
</html>
```

5 Geração de documentação

Num projecto de desenvolvimento de software é fundamental que o código fique documentado, para o tornar mais compreensível, e para que mais tarde, o mesmo possa ser reutilizado ou alterado mais facilmente, tanto pelo próprio criador do código, como por qualquer outro programador.

A não documentação do código após a sua implementação, poderá levar a que não se compreenda o que foi anteriormente implementado. Numa situação em que se pretenda fazer alterações ou reutilização de código, por exemplo, o programador pode não identificar a funcionalidade de determinados procedimentos ou de determinados métodos de uma determinada classe, por não existir qualquer tipo de informação. Se ele for incapaz de reutilizar e tiver que implementar de novo, irá perder tempo, que poderia poupar caso existisse uma base documental com informação sobre o código implementado.

5.1 Gerador de documentação

Para uma mais fácil leitura da documentação é conveniente que esta seja elaborada em ficheiros multimédia, tais como documentos PDF, HTML, Microsoft Compiled HTML Help ou ainda, MAN para sistemas Unix.

Doxygen é um sistema gerador de documentação para projectos de software desenvolvidos em C++, C, Java e Python, por exemplo, que possibilita a geração de ficheiros multimédia com documentação a partir de ficheiros com código documentados.

A geração da documentação, recorrendo a esta aplicação, é realizada com base num ficheiro de configuração que contém um conjunto de opções relacionadas com os ficheiros de código origem e com o formato da documentação que irá ser gerada. Contém ainda, informação sobre o projecto, linguagem de programação utilizada, localização dos ficheiros de código, por exemplo.

No caso particular deste projecto, recorreu-se à ferramenta Doxywizard, disponibilizada pelo Doxygen, para a geração do ficheiro de configuração. As opções anteriormente referidas são tomadas durante a execução do Doxywizard. Posteriormente à geração do ficheiro, foi utilizada a mesma ferramenta, para gerar a documentação do código implementado neste projecto, no formato HTML.

5.2 Formato utilizado na documentação

Para que seja possível gerar a documentação através desta aplicação é necessário que os comentários dos ficheiros de código estejam num formato específico, que se encontra descrito na documentação do *doxygen.*

Tomemos os seguintes exemplos do formato utilizado para documentação das classes:

- Formato sobre a identificação e descrição de uma classe:

```
## Documentation for <Nome da Classe>
#
# <Descrição da funcionalidade da Classe>
```

- Formato sobre da descrição do *constructor* de uma classe:

```
## The constructor.
#
# <Descrição do Contructor>
#
# @Param <Nome Parâmetro> <Tipo de Dados>
# @Param (…)
```

- Formato da descrição do *destructor* de uma classe:

```
## The destructor.
#
# <Descrição do Destructor>
```

- Formato da descrição de um método de uma classe:

```
##
# <Descrição do método>
#
# @Param <Nome Parâmetro> <Tipo de Dados>
#
# @return <Tipo de Dados>
```

- Quando se pretende fazer referência a um tipo de dados já referenciado, usa-se o seguinte formato de comentário:

```
# @see <Tipo de Dados referenciado>
```

- Exemplo utilizado na documentação da classe *CMonitorControls*:

```
## Documentation for CMonitorControls.
#
#Monitorize, manage and maintain the interactions with the resizableWidgets,
#where is possible to create (@see addNewControl) and delete (@see deleteControl)
#resizable Widgets, change properties (@see changeProperties) and some more actions
#related with.
class CMonitorControls(QtCore.QObject):
        (…)

## The constructor.
#
def __init__(self):
        (…)

##
```

```
# Returns the related information from Control identified with the given typeControl and
idControl.
#
# @Param typeControl string
# @Param idControl string
#
# @return CControlInfo
# @see CControlInfo
def getControlInfo(self, typeControl, idControl):
        (…)
```

6 Dificuldades sentidas

6.1 Necessidade

O desenvolvimento deste projecto foi realizado por três elementos onde nem todos estavam geograficamente próximos. Antes de iniciar o desenvolvimento deste projecto persistiram uma série de questões fundamentais:

- Como iremos trabalhar em conjunto, não estando fisicamente próximos?
- Como iremos armazenar o trabalho desenvolvido para o projecto e disponibilizá-lo?
- Como iremos controlar o que cada elemento do grupo de trabalho desenvolve?

Para além dos vários meios de comunicação disponíveis para conversação, foi necessário encontrar um sistema que permitisse armazenar e gerir o trabalho desenvolvido pelos elementos do grupo, e que estivesse disponível via Internet, que seria o único meio de acesso possível para todos os elementos.

6.2 Repositório de dados

Através do *Google Code* foi utilizado um serviço denominado de *Project Hosting*, o qual oferece uma plataforma de controlo de versões e de identificação de *bugs* sobre projectos de cariz *open-source*. Esta plataforma de alojamento disponibiliza várias facilidades, em ambiente de desenvolvimento orientado a grupos de trabalho, entre as quais temos:

- Espaço de armazenamento virtual para alojamento de ficheiros associados ao projecto (repositório de dados);
- Controlo de acessos;
- Controlo de versões sobre os ficheiros existentes no repositório de dados do projecto;
- Envio de relatórios para a *mailing list* do projecto, a informar sobre actualizações feitas aos ficheiros existentes no repositório de dados.

Assim através deste serviço foi criado um projecto ao qual está associado um repositório de dados que pode ser mantido através do sistema de controlo de versões [11] Subversion [12].

Este repositório de dados consiste num espaço virtual que permite o alojamento de ficheiros relacionados com o projecto por qualquer membro associado ao mesmo. É possível a criação de pastas para a organização do repositório.

Associadas ao repositório existem as seguintes pastas de armazenamento:

- **/branches –** Contém cópias das várias versões do repositório;

- **/trunk –** Contém a actual versão actual do repositório;
- **/tags –** Contém a documentação associada ao projecto.

O acesso ao repositório é controlado pela plataforma através de um nome de utilizador e de uma palavra-chave.

O projecto foi criado com a seguinte configuração:

Tabela 6.1 – Configuração do projecto no Google Code

Nome do projecto:	"qooxdooguibuilder" – Qooxdoo Gui Builder
URL do repositório:	http://qooxdooguibuilder.googlecode.com/svn/trunk/
Licença:	GNU General Public License 2.0
Project Owners:	jjcaeiro
Project Members:	claudia.i.h.oliveira, claudio.pedro, nuno.a.coelho

6.3 Sistema de controlo de versões

O sistema de controlo de versões utilizado para gerir o repositório de dados é o sistema *open-source* Subversion. Este software permite controlar diferentes versões de um ficheiro, mantendo o histórico e o desenvolvimento corrente sob forma de versões.

6.3.1 Vantagens

A utilização deste sistema foi muito útil durante o desenvolvimento do projecto pelas seguintes razões:

- Minimizou os conflitos sobre o trabalho desenvolvido pelos vários elementos do grupo;
- Possibilitou o retrocesso para versões anteriores quando houve necessidade;
- Registou as alterações efectuadas, numa determinada data, por um elemento do grupo, sob a forma de *logs*.

6.3.2 Gestão do repositório de dados

O acesso ao repositório foi feito através da aplicação cliente TortoiseSVN. Esta aplicação permite ao utilizador do repositório interagir com o sistema Subversion através de uma interface gráfica simples, integrada no sistema operativo. O utilizador poderá gerir o repositório de dados facilmente e utilizar todas as potencialidades que o sistema Subversion oferece.

No desenvolvimento do nosso projecto, foram utilizadas essencialmente 4 funcionalidades fundamentais da aplicação TortoiseSVN:

- ***Checkout*:** Permite ao utilizador obter uma cópia da versão actual do repositório de dados, na sua estação de trabalho. Para esse efeito, tem que indicar o URL do mesmo, e a directoria onde pretende que seja guardada a cópia.

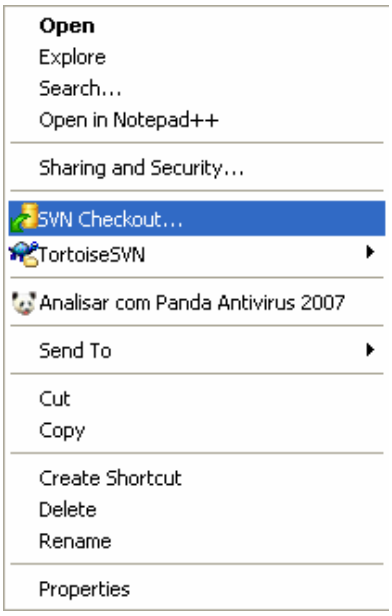

Figura 6.1 – Menu de contexto com a opção para iniciar o *checkout* do TortoiseSVN seleccionada

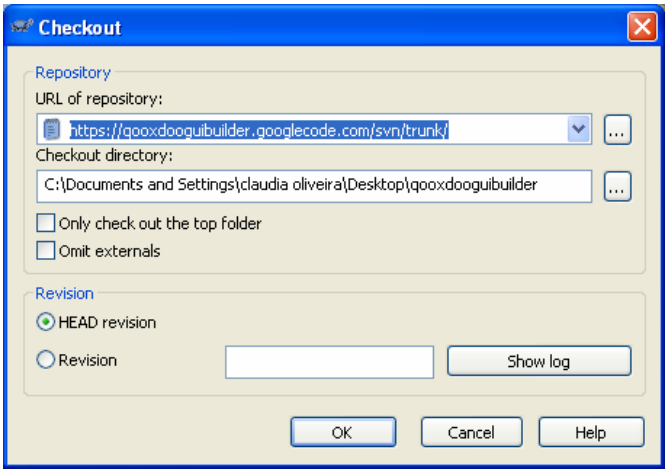

Figura 6.2 – Janela onde se configura o *checkout* do TortoiseSVN

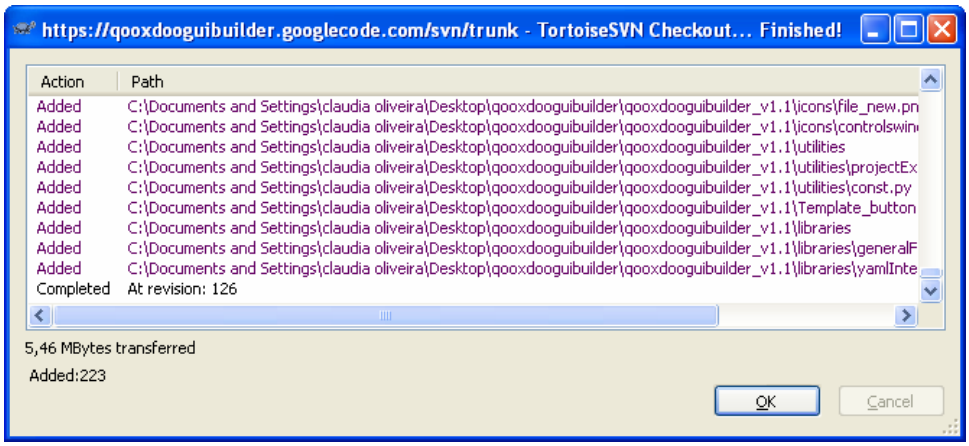

Figura 6.3 – Janela de processamento do *checkout* do TortoiseSVN

- ***Add:*** Permite ao utilizador marcar pastas e ficheiros, como itens a serem adicionados ao repositório de dados, durante o próximo *commit*.

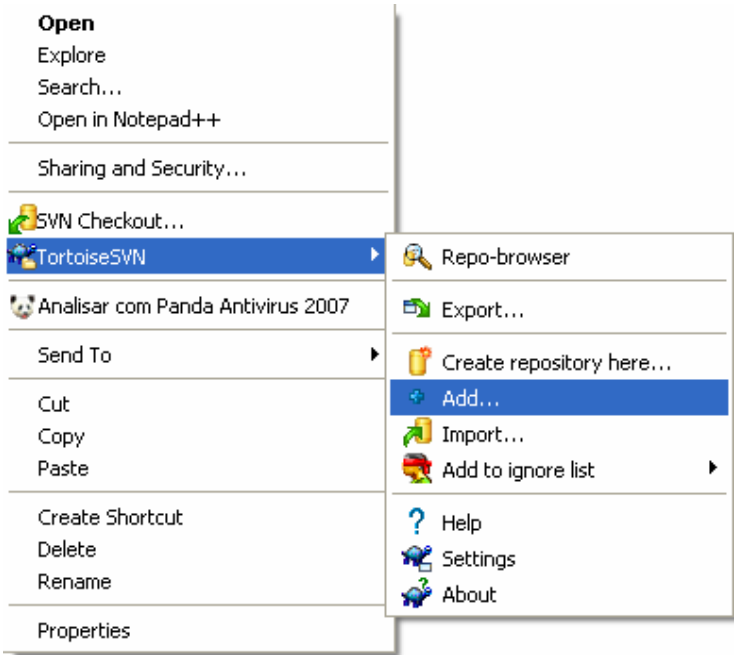

Figura 6.4 – Menu de contexto com a opção para iniciar o *add* do TortoiseSVN seleccionada

Figura 6.5 – Janela onde se configura o *add* do TortoiseSVN

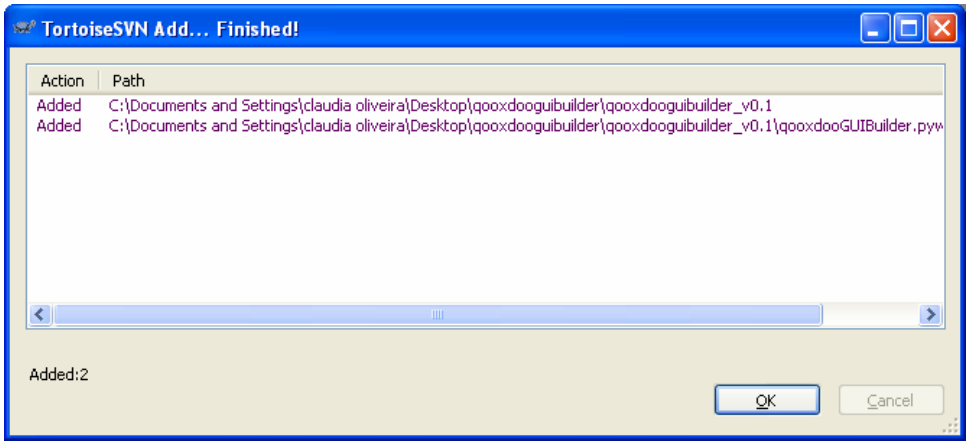

Figura 6.6 – Janela de processamento do *add* do TortoiseSVN

- ***Commit*:** Permite ao utilizador submeter para o repositório de dados, alterações efectuadas na cópia do repositório, existente na sua estação de trabalho. O utilizador poderá ainda associar ao *commit*, uma mensagem que informe acerca das alterações realizadas. Esta mensagem facilitará a percepção das alterações aos outros membros do projecto e poderá ser útil em posteriores consultas. Podemos ainda referir que ao conjunto de alterações é dada a designação de *revision*, e que a cada *commit* é atribuído um índice numérico crescente de *revision*.

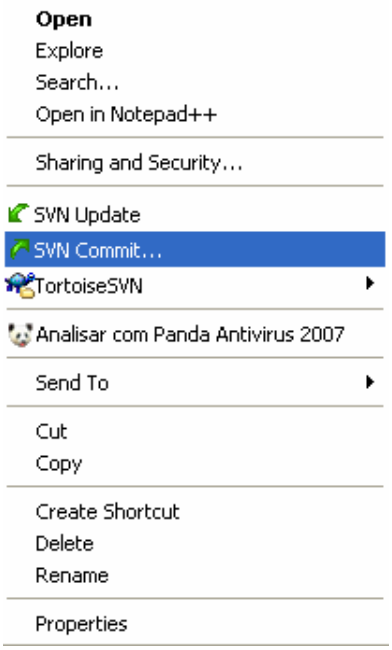

Figura 6.7 – Menu de contexto com a opção para iniciar o *commit* do TortoiseSVN seleccionada

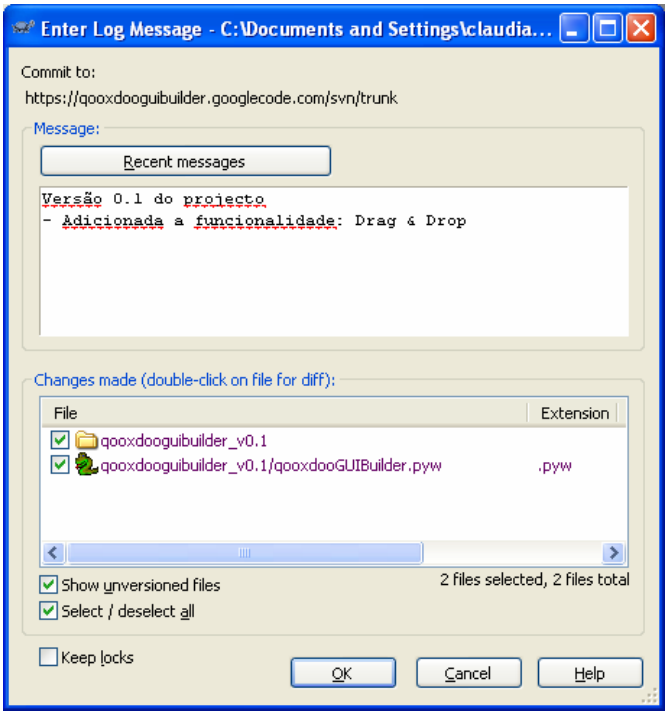

Figura 6.8 – Janela onde se configura o *commit* do TortoiseSVN

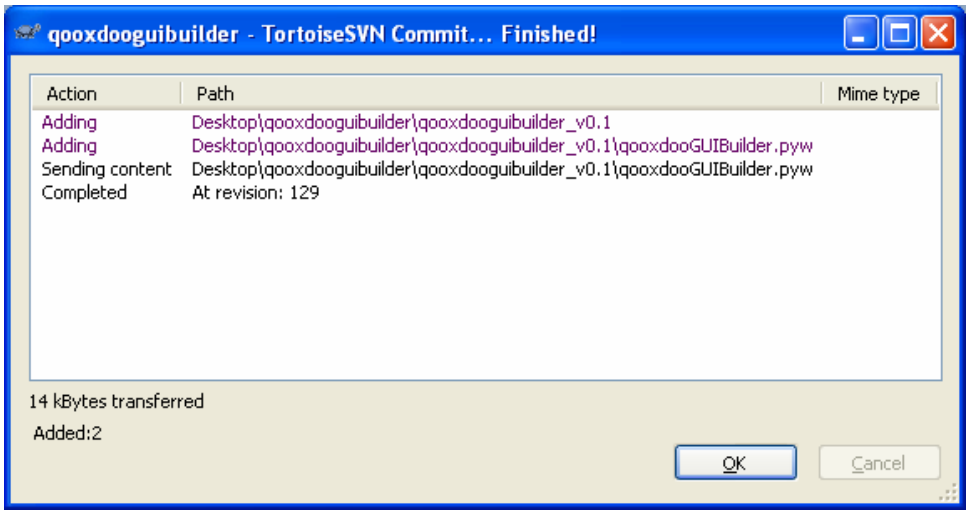

Figura 6.9 – Janela de processamento do *commit* do TortoiseSVN

- ***Update*:** Permite ao utilizador actualizar a cópia do repositório, existente na sua estação de trabalho. Sempre que um elemento do grupo de trabalho efectua um *commit* sobre o repositório de dados, é expressamente necessário que os restantes membros actualizem a cópia do repositório na qual trabalham, de forma a evitar conflitos no desenvolvimento. A notificação dos *commits* é feita via e-mail, para a *mailing list* associada ao projecto.

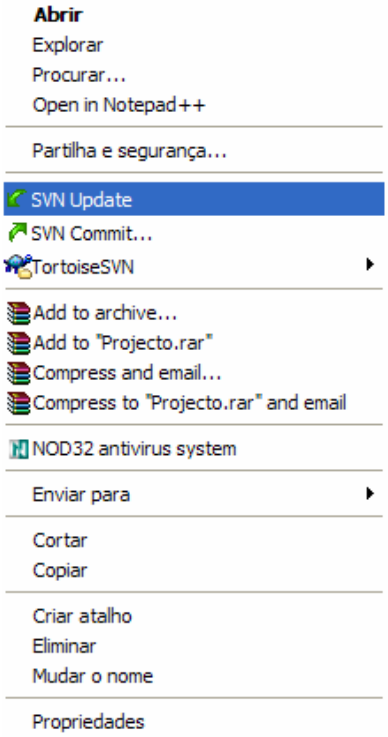

Figura 6.10 – Menu de contexto com a opção para iniciar o *update* do TortoiseSVN seleccionada

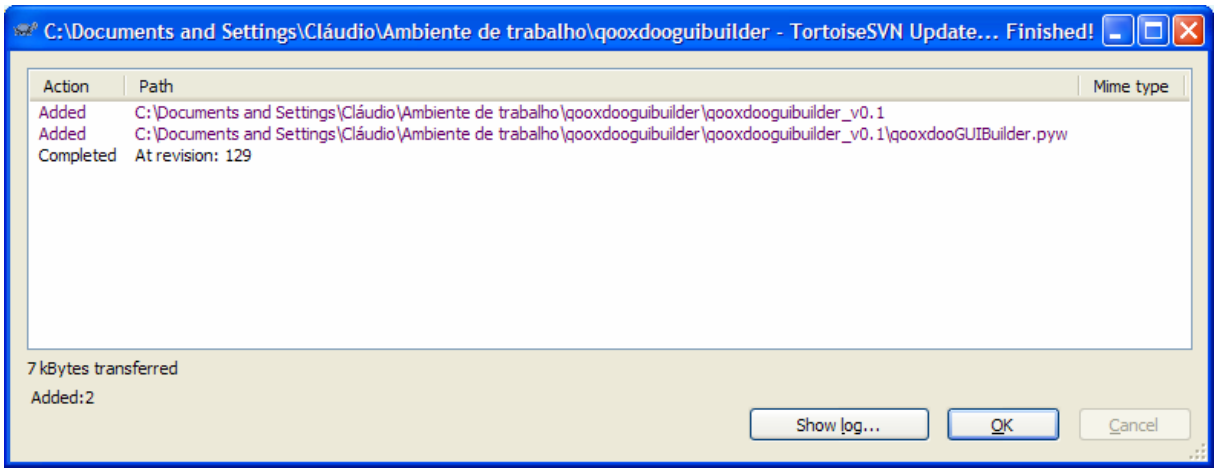

Figura 6.11 – Janela de processamento do *update* do TortoiseSVN

7 Testes

Durante a fase de implementação do sistema foram sendo, sistematicamente, efectuados testes com o objectivo de cumprir com todos os requisitos inicialmente identificados. Ainda assim, com a implementação terminada, foram efectuados novamente testes de validação ao sistema.

Foram requisitados três utilizadores que nunca haviam interagido com a aplicação, que testaram as funcionalidades e o desempenho da mesma. Todos concluíram que a aplicação estava pronta para começar a ser utilizada, embora se tenham queixado da impossibilidade de executar determinadas acções, que já estão previstas na aplicação, mas não implementadas.

8 Trabalhos futuros

Undo

O desenvolvimento desta funcionalidade permitirá ao utilizador anular acções por si realizadas, durante a construção de interfaces. Desta forma, será possível voltar a um estado anterior na criação de uma interface, caso ele pretenda.

Redo

O desenvolvimento desta funcionalidade permitirá ao utilizador refazer acções por si anuladas, durante a construção de interfaces. Desta forma, será possível avançar para um estado posterior na criação de uma interface, caso ele pretenda.

Cut

O desenvolvimento desta funcionalidade permitirá ao utilizador "cortar" um ou mais controlos durante a construção de interfaces. Desta forma, será possível dar início ao processo de translação de um ou mais controlos para uma localização diferente.

Copy

O desenvolvimento desta funcionalidade permitirá ao utilizador copiar um ou mais controlos durante a construção de interfaces. Desta forma, será possível dar início ao processo de multiplicação de um ou mais controlos, na área de desenho.

Paste

O desenvolvimento desta funcionalidade permitirá ao utilizador "colar" um ou mais controlos na área de desenho, sobre os quais tenha sido executada a acção *cut* ou *copy*, durante a construção de interfaces. Desta forma, será possível terminar os processos de translação e de multiplicação providenciados pelas funcionalidades *cut* e *copy*, respectivamente.

Aplicação auxiliar gestora dos controlos existentes na aplicação

O desenvolvimento desta aplicação auxiliar permitirá adicionar e remover controlos da janela que contem os controlos disponíveis para a criação de interfaces. Permitirá ainda, adicionar e remover propriedades específicas associadas a um determinado controlo e propriedades gerais associadas a todos os controlos.

Adicionar eventos aos controlos

A possibilidade de adicionar eventos aos controlos existentes nas interfaces aumentaria o nível de usabilidade das mesmas e permitiria ao utilizador, por exemplo, fazer um botão executar uma determinada acção.

Criar várias interfaces em simultâneo

A possibilidade de criar várias interfaces em simultâneo, utilizando separadores na área de desenho, permitiria ao utilizador, por exemplo, copiar controlos de uma área de desenho para outra.

Grelha discreta sobre a área de desenho

Uma grelha discreta sobre a área de desenho permitiria ao utilizador da aplicação orientar-se melhor na construção de interfaces.

9 Conclusão

Este projecto permitiu acima de tudo aprofundar conhecimentos ao nível das aplicações que ajudam os utilizadores a desenvolver mais facilmente interfaces. Permitiu ainda, aplicar conhecimentos adquiridos em diversas disciplinas leccionadas durante o curso de engenharia de informática, como são os casos de Engenharia de Software, Pesquisa e Optimização, Multimédia e Sistemas Interactivos e Interacção Pessoa-Computador.

Através do estudo realizado sobre ferramentas análogas, tiraram-se conclusões quanto às características cuja implementação na ferramenta a desenvolver, é fundamental. De entre as várias características destacam-se o *drag & drop* na manipulação de controlos visuais e a criação e aplicação de templates.

Relativamente ao desenvolvimento do sistema, a fase de análise, foi aquela que recebeu maior atenção pois a mesma iria definir todo o restante trabalho. A fase de desenho, por sua vez foi a que definiu o aspecto e o funcionamento para o qual a aplicação iria convergir. A fase mais prolongada foi claramente, a fase de implementação. A fase mais curta em termos de duração foi a fase de testes, porque ao mesmo tempo que iam sendo efectuadas implementações, iam sendo efectuados testes sobre as mesmas. Mesmo assim a aplicação acabou por ser testada por três utilizadores completamente estranhos à aplicação.

Em suma, este projecto foi extremamente aliciante e interessante devido à sua abrangência de conhecimentos diversificada, sendo que será igualmente interessante seguir o processo evolutivo da aplicação desenvolvida, bem como, o aumento das suas potencialidades e funcionalidades.

10 Bibliografia

[1] *Ajax (programação)*. (2007, Janeiro 14). [Online]. Disponível em http://pt.wikipedia.org/wiki/AJAX_%28programa%C3%A7%C3%A3o%29

[2] *qooxdoo*. (2006, Dezembro 22). [Online]. Disponível em http://qooxdoo.org

[3] Matthias Kalle Dalheimer, *Programming with Qt, 2nd Edition*, O'Reilly, Janeiro 2002, capítulo 1.

[4] *Bindows*. (2007, Janeiro 29). [Online]. Disponível em http://www.bindows.net

[5] *Dojo*. (2006). [Online]. Disponível em http://dojotoolkit.org

[6] *Prototype*. (2007). [Online]. Disponível em http://www.prototypejs.org

[7] *Aptana*. (2006, Dezembro 20). [Online]. Disponível em http://www.aptana.com

[8] *Macromedia Dreamweaver*. (2006, Junho 21). [Online]. Disponível em http://www.adobe.com/products/dreamweaver

[9] *Qt Designer*. (2007). [Online]. Disponível em http://www.trolltech.com/products/qt/features/designer

[10] *Waterfall model*. (2007, Janeiro 28). [Online]. Disponível em http://en.wikipedia.org/wiki/Waterfall_development

[11] *Heuristic Evaluation*. (2005). [Online]. Disponível em http://www.useit.com/papers/heuristic

[12] *Sistema de controle de versão* (2007, Junho 5). [Online]. Disponível em http://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%A3o

[13] *Subversion (software)*. (2007, Junho 15). [Online]. Disponível em http://en.wikipedia.org/wiki/Subversion_(software)

11 Agradecimentos

Gostaríamos de agradecer ao Eng. José Jasnau Caeiro por todo o apoio prestado, pelas condições proporcionadas, pelo conhecimento transmitido e pela disponibilidade oferecida para a resolução de vários problemas com que nos deparámos ao longo do projecto.