



Universidade do Algarve  
Faculdade de Ciências e  
Tecnologias

# **Package Java para Criação Dinâmica de Interfaces Gráficas Definidas em XML**

Relatório de Projecto

Ano Lectivo: 2002/2003

Curso: Licenciatura em Ensino De Informática

Orientador: Prof. Doutor João M. P. Cardoso

Realizado por:

Nome: Gil Moreira

N.º 17325

Faro, 4 de Julho de 2003

## Resumo

Este projecto teve como objectivo principal a simplificação da construção de interfaces gráficas. Para a realização dos objectivos foram escolhidas duas tecnologias recentes, Java e XML. Também para este fim foi definida uma linguagem de formatação de interfaces utilizando XML, e foi criada uma *package* em Java que interpreta essa linguagem e constrói a interface respectiva. Esta *package* adiciona ainda uma camada de abstracção entre o programador e o Swing que é o *kit* nativo de construção de interfaces em Java. A linguagem de formatação permite associar acções de vários tipos aos componentes da interface gráfica. Acções que podem ser utilizadas sem necessidade de adicionar código extra e acções que requerem adição de código pelo programador. Efectivamente, é provado que é viável a utilização de XML para definir a estrutura de interfaces gráficas e que a interligação das duas tecnologias, referidas anteriormente, permite realmente simplificar a sua construção.

## Agradecimentos

O autor deseja agradecer aos colegas de projecto pela companhia, apoio e incentivos, e espera sinceramente que as noites passadas em frente a um monitor, naquela maravilhosa sala carinhosamente apelidada de *bunker*, não tenham sido em vão. Um agradecimento especial é dirigido ao orientador deste projecto, pois sem ele, de certeza que, nada teria sido concretizado.

# Índice

<b>Figuras .....</b>	<b>iv</b>
<b>Exemplos.....</b>	<b>iv</b>
<b>Tabelas.....</b>	<b>iv</b>
<b>Acrónimos .....</b>	<b>v</b>
<b>1. Introdução .....</b>	<b>1</b>
<b>2. Tecnologias Utilizadas.....</b>	<b>5</b>
2.1 XML.....	5
2.2 Java .....	6
2.2.1 Swing .....	7
2.2.2 Reflection.....	10
2.3 Java & XML .....	11
2.3.1 Xerces.....	12
SAX .....	13
DOM .....	14
<b>3. Trabalho Realizado .....</b>	<b>17</b>
3.1 Especificação XML .....	18
3.1.1 Acções .....	20
3.2 Métodos .....	22
3.3 Classes desenvolvidas .....	23
3.4 Processamento dos componentes.....	24
<b>4. Programas Exemplo .....</b>	<b>26</b>
4.1 Aplicação 1 .....	26
4.2 Aplicação 2 .....	26
4.3 Aplicação 3 .....	30
<b>5. Conclusões e Trabalho Futuro.....</b>	<b>32</b>
<b>Referências .....</b>	<b>33</b>
<b>Apêndices .....</b>	<b>34</b>
Apêndice 1 – Código que define uma interface em Swing.....	35
Apêndice 2 – Código que adiciona uma acção a um botão .....	36
Apêndice 3 – Tabela de componentes e atributos .....	37
Apêndice 4 – Exemplo de uma interface definida em XML.....	38
Apêndice 5 – Diagrama de classes.....	39
Apêndice 6 – Código e definições das aplicações exemplo .....	40
Aplicação 1 .....	40
Aplicação 2 .....	43
Aplicação 3.....	48

## Figuras

Figura 1 – GUI do <i>System 1.1</i> do Macintosh. ....	1
Figura 2 – Primeira hipótese para um gerador de GUIs. ....	3
Figura 3 – Segunda hipótese considerada. ....	3
Figura 4 – Demonstração dos componentes Swing. ....	8
Figura 5 – Exemplo de uma interface em Swing. ....	8
Figura 6 – Interface com um botão funcional. ....	9
Figura 7 – Estrutura de um ficheiro XML. ....	12
Figura 8 – Árvore gerada que descreve a estrutura do documento. ....	15
Figura 9 – Esquema das fases do modelo de prototipagem. ....	17
Figura 10 – Diagrama de classes simples. ....	23
Figura 11 – Interface da aplicação exemplo 1. ....	26
Figura 12 – Interface da aplicação exemplo 2. ....	27
Figura 13 – Menu Control. ....	27
Figura 14 – Menu <i>Source File</i> . ....	28
Figura 15 – Escolha de ficheiros. ....	28
Figura 16 – Menu Help. ....	29
Figura 17 – Janela <i>AboutWindow</i> . ....	29
Figura 18 – Execução de um <i>Compile</i> . ....	30
Figura 19 – Aplicação Exemplo 3. ....	31

## Exemplos

Exemplo 1 – Programa em Java que escreve no ecrã “Olá Mundo”. ....	7
Exemplo 2 – Linhas de código que ilustram o mecanismo de acções. ....	9
Exemplo 3 – Criação de um objecto usando o <i>reflection</i> . ....	10
Exemplo 4 – Invocação de um método com parâmetros. ....	11
Exemplo 5 – Código Java que percorre uma árvore DOM. ....	15

## Tabelas

Tabela 1 – <i>Tags</i> de componentes. ....	18
Tabela 2 – <i>Tags</i> de controlo. ....	19
Tabela 3 – Atributos. ....	19
Tabela 4 – Atributos para atribuir acções. ....	20
Tabela 5 – Valores do atributo ACTION que não necessitam de adição de código Java. ....	20
Tabela 6 – Valores do ACTION que necessitam de métodos cliente. ....	21
Tabela 7 – Métodos de interacção com componentes. ....	22

## Acrónimos

API  
*Application Programming Interface*

DOM  
*Document Object Model*

DTD  
*Document Type Definition*

GUI  
*Graphical User Interface*

HTML  
*Hypertext Markup Language*

IDE  
*Integrated Development Environment*

J2SDK  
*Java 2 Software Development Kit*

SAX  
*Simple API for XML*

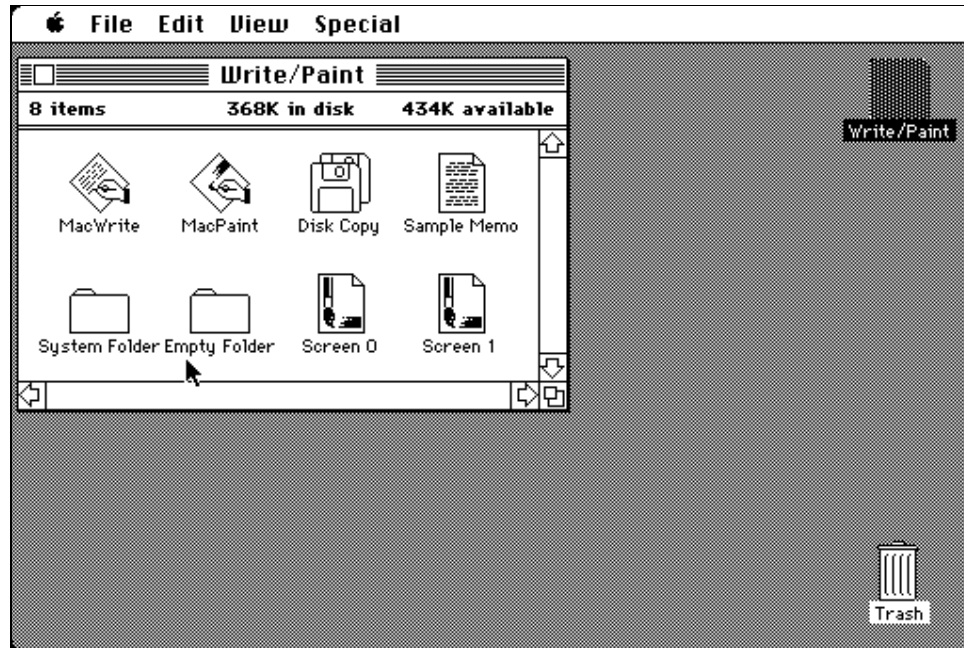
W3C  
*World Wide Web Consortium*

XML  
*eXtensible Markup Language*

## 1. Introdução

Desde o aparecimento e comercialização do *Macintosh*<sup>1</sup> da *Apple* que as interfaces gráficas se foram tornando cada vez mais comuns, até se tornarem o meio preferido da grande maioria dos utilizadores de computadores para interagir com estes.

Também chamadas de GUIs<sup>2</sup>, estas interfaces iniciais introduziram novos conceitos, que, mais do que formas de representar conceitos abstractos de forma pictórica e intuitiva (os ícones são um bom exemplo) tornaram-se essenciais para a utilização de um computador. Pode-se ver na *Figura 1* uma das primeiras interfaces gráficas.



**Figura 1 – GUI do *System 1.1* do Macintosh.**

Esta revolução deveu-se também ao aparecimento do rato, um dispositivo de controlo completamente diferente do teclado. O rato funciona como ponteiro, permitindo aos utilizadores deslocar no ecrã, tipicamente, uma seta e escolher as acções a serem executadas através de um simples pressionar de botão. De facto este dispositivo tornou-se tão importante como corriqueiro, e até foi inventada uma

---

<sup>1</sup> Um dos primeiros computadores pessoais com interface gráfica.

<sup>2</sup> Do termo inglês Graphical User Interfaces.

palavra para o acto de pressionar e largar o botão, o clique<sup>3</sup>, que se refere ao som que o botão do rato faz quando pressionado.

Foi portanto este casamento, entre o rato e as interfaces gráficas, o responsável pela cimentação desta nova forma de trabalhar com um computador. Este casamento é tão forte que, apesar de possível, é difícil e laborioso interagir com uma interface gráfica através de um teclado. No entanto este ainda é essencial para tarefas de introdução de dados.

Em conjunto com outros factores, esta mudança no modo de interacção contribuiu imenso para a relativa facilidade com que hoje se trabalha com um computador. Actualmente, já se vai tornando raro encontrar, fora do ramo da informática, quem saiba trabalhar com uma linha de comandos, o rato é hoje algo de indispensável em ambientes gráficos. Qualquer sistema operativo possui uma interface gráfica, passando-se o mesmo com as aplicações.

O desenvolvimento de aplicações é um processo de tal forma complexo que existem inúmeras metodologias diferentes para lidar com ele. Quando juntamos o desenho e programação de uma interface gráfica para a aplicação, o desenvolvimento torna-se, previsivelmente, mais complicado. A interface necessita de código para ser gerada, o que vai aumentar a complexidade do software. É muitas das vezes um processo moroso pois é necessário dominar as APIs (*Application Programming Interface*).

Procurou-se com este projecto, estudar uma forma de simplificar este processo de criação de interfaces gráficas. O objectivo principal é a interligação de tecnologias, nomeadamente o Java e o XML (*eXtensible Markup Language*), de forma a tornar mais acessível a construção da interface gráfica de uma aplicação. Procura-se desta forma retirar complexidade ao desenvolvimento de software acelerando ao mesmo tempo a sua construção.

Foram inicialmente idealizadas duas possibilidades de resolver esta questão. Na *Figura 2* é ilustrada a primeira hipótese considerada. E com a excepção da introdução do XML é a abordagem utilizada pelos editores de GUIs actuais, por exemplo o editor integrado do *JBuilder*<sup>4</sup>. Nesta abordagem a definição de uma interface seria feita em XML. Esta definição seria processada por um gerador de GUIs que criaria o código do GUI usando os procedimentos da API utilizada. O

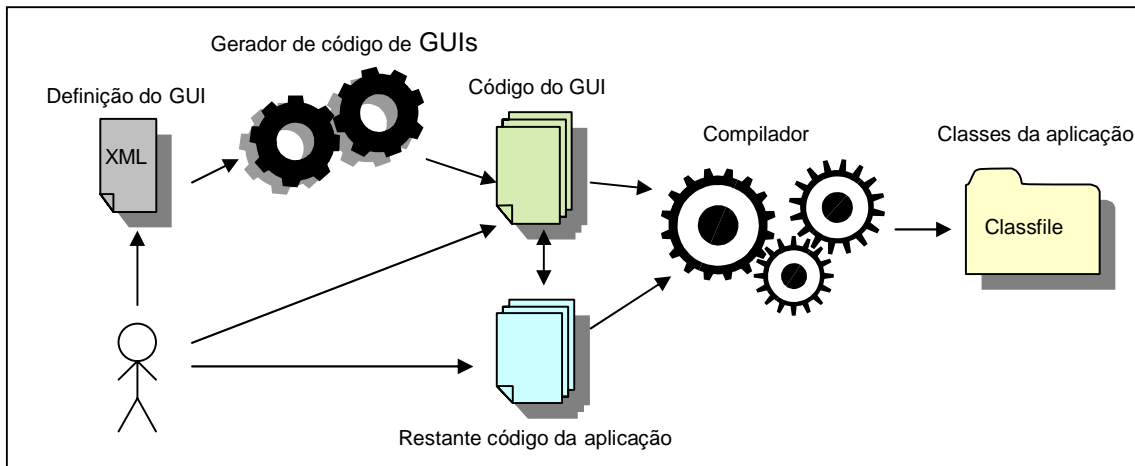
---

<sup>3</sup> Do termo inglês *Click*.

<sup>4</sup> [www.borland.com](http://www.borland.com)

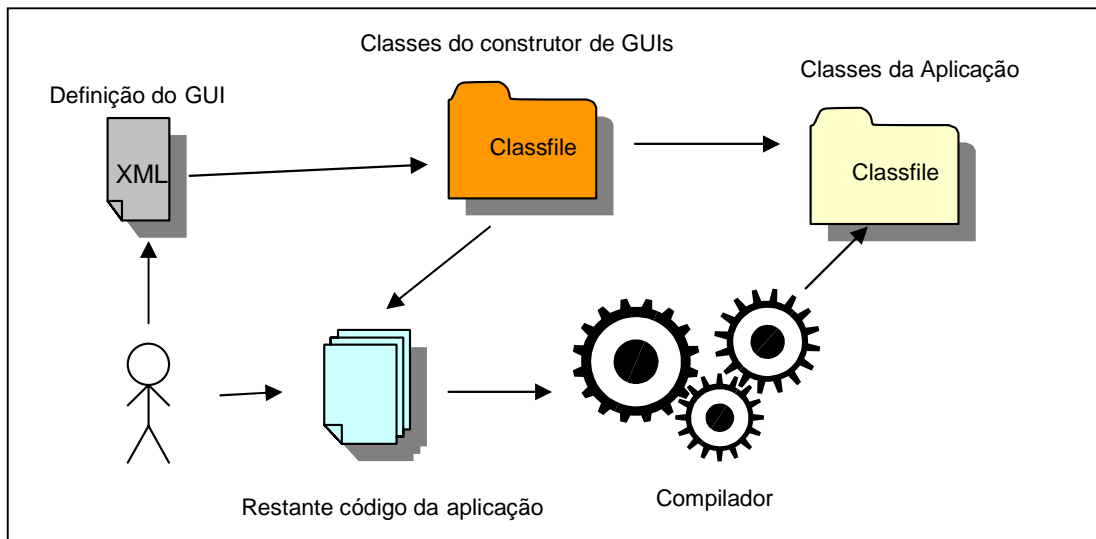


utilizador teria depois de juntar código (para tratar os eventos despoletados pelos componentes, etc) e compilar o programa final.



**Figura 2 – Primeira hipótese para um gerador de GUIs.**

Na segunda hipótese, ilustrada na *Figura 3*, a interface gráfica seria criada automaticamente. Para isto seria construído um conjunto de classes que interpretariam a definição em XML, tendo o utilizador de somente invocar a interpretação e construção no código da aplicação.



**Figura 3 – Segunda hipótese considerada.**

Foi decidido seguir a abordagem da segunda hipótese porque torna o processo de integração do código mais simples. Ao contrário da primeira hipótese,

implica que o utilizador aprenda a utilizar os métodos fornecidos, mas evita a aprendizagem mais demorada da API nativa para gerar GUIs, e fornece por isso um grau de abstracção mais elevado.

Sendo o objectivo principal do projecto a interligação de Java e XML para a criação de GUIs, é analisada a possibilidade de utilizar as capacidades do XML para definir uma estrutura que corresponda a uma interface gráfica. Posteriormente é explicada a construção da interface, automaticamente, utilizando Java e Swing. Como nem Java nem XML fazem parte do conteúdo curricular da licenciatura, os objectivos secundários, fundamentais para a realização do projecto, são a aprendizagem de alguns conceitos de XML e Java. Incluindo-se na aprendizagem da tecnologia Java a aprendizagem das APIs necessárias: SAX, DOM e o *kit* de construção de interfaces gráficas Swing.

No capítulo 2 deste relatório será feita uma introdução às tecnologias utilizadas com as justificações dos motivos pelas quais foram escolhidas. No capítulo 3 será apresentado o trabalho que foi realizado, de uma forma resumida. No capítulo 4 serão apresentados alguns exemplos práticos do trabalho desenvolvido. E finalmente no capítulo 5 serão feitas as considerações finais do projecto e apresentadas algumas possibilidades de trabalho futuro.

## 2. Tecnologias Utilizadas

Neste capítulo serão descritas as tecnologias utilizadas, tal como os seus pontos fortes, *desvantagens* e descritos os motivos pelos quais foram escolhidas. Um dos pontos em comum entre estas tecnologias é que foram examinadas na perspectiva da realização do projecto, ou seja, as características que foram examinadas mais em pormenor foram as relevantes para a realização do projecto. Quer isto dizer que não foi aprendido tudo o que há para aprender sobre cada uma delas.

### 2.1 XML

O XML [1], *eXtensible Markup Language*, é uma tecnologia muito completa que permite todo o tipo de operações de tratamento de informação, desde formatar documentos até filtrar dados. Apesar do nome, o XML não é em si uma linguagem de *markup* (formatação), mas antes um conjunto de regras pelas quais podemos definir linguagens desse tipo<sup>5</sup>.

Pode-se ver neste exemplo, uma formatação válida em XML de um documento:

```
<?xml version="1.0"?>

<mensagem tipo="informação">
  <paragrafo> Este é um documento em XML.</paragrafo>
  <paragrafo> O XML é muito útil.</paragrafo>
</mensagem>
```

À partida vêm-se algumas semelhanças com o *HTML*, tem-se palavras-chave ou *tags*, entre “< >” que são utilizadas para definir o que se chamam de *elementos*. As *tags* que possuem “/” antes do nome do elemento, indicam que estamos a fechar aquele elemento, por exemplo </mensagem>. Quando o elemento não contém texto é possível fechá-lo desta forma <mensagem />.

O cabeçalho, <?xml ?> presente na primeira linha, é onde se podem definir algumas características do documento, por exemplo, a versão de XML, *version="1.0"* ou o *encoding*, que indica em que formato está codificado o ficheiro de texto, são

---

<sup>5</sup> Designa-se *freeform XML*.

algumas. É possível ainda que as *tags* possuam valores, chamados atributos, no exemplo a *tag mensagem* possui um atributo *tipo*, que neste caso poderá indicar o tipo da mensagem.

O XML permite definir *tags* livremente, pode-se definir uma linguagem de formatação com estas *tags*, a isso chama-se um *document type* ou *aplicação XML*. Também é possível utilizar um *document type definition (DTD)*, que contém uma definição de todas as *tags* permitidas na linguagem, para validar<sup>6</sup> um documento. Ao programa que se encarrega de ler o documento XML do ficheiro chama-se *parser XML*, este lê o ficheiro interpreta os dados e passa-os a outro programa, ou rotina.

O standard XML é gerido pelo W3C<sup>7</sup> (*World Wide Web Consortium*) a mesma entidade responsável pela maior parte dos standards utilizados na Internet. Presentemente a versão de XML é a 1.0.

## 2.2 Java

A linguagem de programação utilizada neste projecto foi o Java. Esta linguagem foi desenvolvida pela *Sun Microsystems*, com o objectivo principal de ser independente da plataforma, isto é, que um programa feito em Java funcione sem necessidade de alterações, incluindo recompilação, em qualquer tipo de computador ou sistema. Isto é concretizado através da chamada, *máquina virtual*. Todos os sistemas que desejem correr programas Java necessitam de implementar esta máquina. O programa é executado nesse computador virtual, que é obrigado a fornecer as mesmas funcionalidades em qualquer plataforma. A única variação de sistema para sistema, é a forma como a máquina está implementada, no entanto para o programador Java esses detalhes são transparentes.

Tudo isto significa que um sistema para correr um programa feito em Java necessita apenas de ter instalada a *máquina virtual*. É possível instalar esta máquina virtual em muitos sistemas operativos diferentes, por exemplo, *Windows* e *Unix*. Esta característica juntamente com a possibilidade de integrar aplicações Java em HTML<sup>8</sup>, as chamadas *applets* Java, que concedem mais interactividade a uma página Web e são executadas no lado do cliente, tornou esta linguagem muito popular. Destaca-se ainda que a linguagem Java é orientada a objectos possuindo

---

<sup>6</sup> Verificar se um documento está de acordo com as regras lexicais e sintácticas definidas no DTD.

<sup>7</sup> [www.w3c.com](http://www.w3c.com)

<sup>8</sup> HyperText Markup Language, linguagem de formatação utilizada para construir páginas de Internet.

uma sintaxe similar ao C++. A linguagem Java apresenta propriedades que permitem a obtenção de maior produtividade relativamente a outras linguagens.

Actualmente esta linguagem encontra-se na segunda versão (Java 2) e algumas ferramentas encontram-se disponíveis no site da *Sun* gratuitamente. Pode-se encontrar lá o *Java2 Runtime Environment*<sup>9</sup> que permite executar aplicações Java, e o *Java2 SDK*<sup>10</sup> que possibilita a criação de aplicações Java. Pode-se ver no *Exemplo 1* um pequeno programa em Java que imprime “Olá Mundo”.

```
public class ola
{
    public static void main(String args[])
    {
        System.out.println("Olá Mundo");
    }
}
```

**Exemplo 1 – Programa em Java que escreve no ecrã “Olá Mundo”.**

Foi utilizado para este projecto a versão 1.4.1 do *J2SDK*, que providencia as ferramentas básicas para a programação em Java, e o *IDE*<sup>11</sup> da *Borland*, *JBuilder 8 Personal Edition*, que é disponibilizado gratuitamente, mediante um registo, no site<sup>12</sup> da empresa.

### 2.2.1 Swing

O Swing [2] faz parte integrante da linguagem de programação Java, do que é conhecido por *Java Foundation Classes* (JFC) que são um conjunto de classes, disponíveis para os programadores Java, que servem para facilitar o processo de criação de aplicações com interfaces gráficas. Mais especificamente, o Swing é um “kit” para desenvolver aplicações com interfaces gráficas, possibilita a personalização do aspecto dos componentes, suporta os conceitos principais

---

<sup>9</sup> Também conhecido como J2RE

<sup>10</sup> Também conhecido como J2SDK.

<sup>11</sup> *Integrated Development Environment*

<sup>12</sup> <http://www.borland.com/products/downloads>

utilizados nas interfaces, como por exemplo o *drag and drop*<sup>13</sup>, e os componentes que providencia são independentes da plataforma em que o software corre. A *Figura 4* apresenta alguns dos componentes Swing disponíveis.

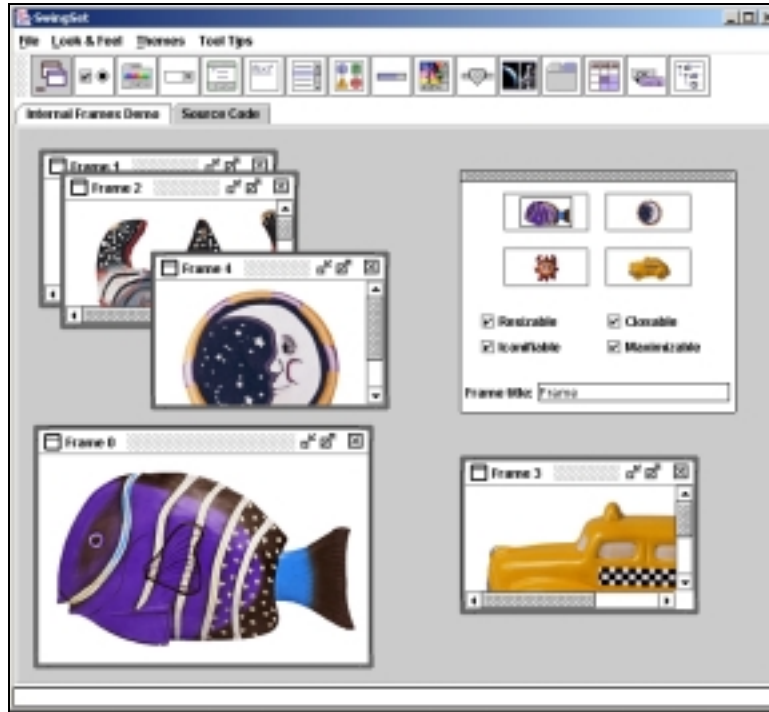


Figura 4 – Demonstração dos componentes Swing.

O Swing foi um dos motivos que levou à utilização do Java como linguagem neste projecto. Pode-se ver na *Figura 5* uma pequena interface criada em Swing.

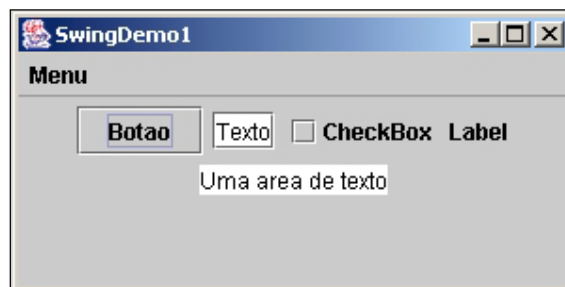
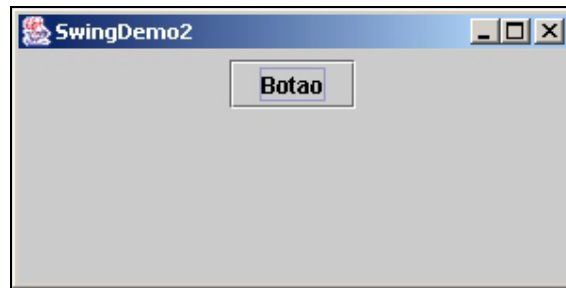


Figura 5 – Exemplo de uma interface em Swing.

O código necessário para definir esta interface encontra-se no **apêndice 1**. Neste caso, apesar de se tratar de uma simples interface, são necessárias para

<sup>13</sup> Termo utilizado para a acção de arrastar um componente gráfico de um sitio para outro, despoletando uma acção na aplicação.

definir a estrutura bastantes linhas de código Java. A atribuição das acções aos componentes requer o desenvolvimento e integração de mais linhas de código. No exemplo seguinte vê-se como é possível atribuir acções a um botão. O botão presente na interface da *Figura 6* escreve a mensagem “Carregou no botão! Obrigado” no ecrã.



**Figura 6 – Interface com um botão funcional.**

O código responsável pela criação da interface e atribuição da acção ao botão encontra-se na no **apêndice 2**. As linhas de código essenciais à compreensão deste mecanismo são:

```
(1) public class SwingDemo2 implements ActionListener  
  
(2) botao.addActionListener(new SwingDemo2());  
  
(3) public void actionPerformed(ActionEvent e)  
    {  
        System.out.println("Carregou no botão! Obrigado");  
    }
```

**Exemplo 2 – Linhas de código que ilustram o mecanismo de acções.**

Este exemplo ilustra a atribuição de uma acção simples a um botão. Para tal foi necessário atribuir ao botão um *ActionListener*, linha (2).

A classe *ActionListener* é uma interface de Java, por isso será implementada pela nova classe através da palavra *implements*, linha (1).

O método *actionPerformed* é o método que a nova classe tem de implementar da interface *ActionListener* que é chamado quando o botão é pressionado, linha (3).

Salienta-se que num caso em que seja necessário ter botões, itens de menu, e outros componentes terá de ser adicionado mais código.

### 2.2.2 Reflection

O *reflection* é uma funcionalidade de Java que permite criar objectos de uma classe existente e invocar métodos dessa classe conhecendo o nome da classe em tempo de execução [3]. O exemplo 3 mostra a instanciação de um objecto e invocação de um método para uma classe com nome identificado por *nomedaclasse*.

```
(1) Class cls = Class.forName(nomedaclasse);  
(2) Object objecto = cls.newInstance();  
(3) Method meth = cls.getMethod(nomedometodo, null);  
(4) meth.invoke(objecto, null);
```

**Exemplo 3 – Criação de um objecto usando o *reflection*.**

A numeração entre () identifica o número da linha.

De seguida são explicadas as linhas de código do exemplo:

- Na linha (1) é declarado um objecto do tipo *Class* e utilizando o método *forName(String)* é obtida a referência à classe que se pretende instanciar.
- Na linha (2) é instanciado um novo objecto da classe pretendida. Através do método *newInstance()*.
- Na linha (3) é criado um objecto do tipo *Method* que irá conter a referência ao nome do método a invocar. O segundo argumento do método *getMethod* especifica os parâmetros do método que se quer obter. Neste caso utilizando *null* indica-se que não tem parâmetros.
- Na linha (4) é invocado o método através do *invoke*. O segundo argumento representa os parâmetros do método que se invoca. Neste caso não possui logo coloca-se *null*.

Este exemplo demonstra a criação de um objecto através do nome da classe, que pode ser apenas conhecido durante a execução da aplicação, e a invocação de



um método, sem parâmetros, desse objecto. No entanto é também possível invocar métodos com parâmetros. O exemplo 5 demonstra isso.

```
(1) Class cls = Class.forName(nomedaclasse);  
(2) Object objecto = cls.newInstance();  
(3) Class[] parametros = new Class[1];  
(4) parametros[0] = Class.forName("java.lang.String");  
(5) Method meth = cls.getMethod(nomedometodo,parametros);  
(6) Object[] valores = new Object[1];  
(7) valores[0] = "valor";  
(8) meth.invoke(objecto, valores);
```

**Exemplo 4 – Invocação de um método com parâmetros.**

As linhas (1) e (2) são semelhantes às linhas correspondentes do exemplo 3. Na linha (3) é criado um array de objectos *Class* que irá conter em cada posição a classe correspondente ao parâmetro do método que se quer invocar. Neste exemplo, o método a invocar só possui um parâmetro, do tipo string. Por este motivo na linha (4) é definido o elemento 0 do array. A linha (5) é igual ao exemplo anterior, com a diferença que agora colocamos como parâmetro adicional do *getMethod* o array de parâmetros do método em vez de *null*. Também é necessário criar um array de *Object* que irá conter os valores para os parâmetros do método, linhas (6) e (7). No final é invocado o método com os valores para os parâmetros que atribuímos no passo anterior, linha (8). No capítulo 3 é explicado onde foi utilizada esta capacidade da linguagem Java.

## 2.3 Java & XML

O processamento de ficheiros XML em Java é relativamente simples. O suporte a XML em Java é assegurado pela existência de vários *parsers* de XML. É

com relativa facilidade que se combinam componentes de Swing com XML [4], como se vê na seguinte figura.

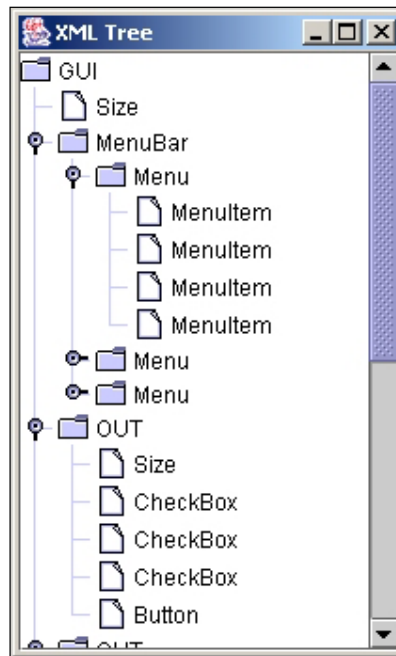


Figura 7 – Estrutura de um ficheiro XML.

Na *Figura 7* pode-se ver a estrutura de um ficheiro XML, que descreve um GUI de acordo com a especificação descrita neste relatório. De seguida é apresentado o *parser* utilizado neste trabalho.

### 2.3.1 Xerces

O Xerces é o *parser* de XML utilizado neste projecto. É um dos vários *parsers* XML disponíveis para Java. É reconhecido como sendo o *parser* XML com a melhor conformidade em relação ao *standard* XML1.0, e suporte a várias APIs, incluindo DOM e SAX, sendo também configurável e adequado para qualquer tipo de processamento XML [4].

Este *parser* é mantido pelo *Apache XML Project*. Este grupo pertence à *Apache Software Foundation*<sup>14</sup>, que é responsável pelo famoso *Apache HTTP Server* utilizado pela maioria dos sites na Internet. Existe também numa versão para

---

<sup>14</sup> <http://apache.org/>

C++ no site<sup>15</sup> do grupo. A versão de Java aparece referenciada como *Xerces2 Java*, e a versão C++ é chamada *Xerces-C++*. Sempre que aparecer o termo *Xerces* ao longo deste documento, está a ser referida a versão de Java. Foi utilizada a versão 2.4.0 deste *software*, a mais recente à data do projecto.

O Xerces suporta várias APIs para o *parsing* de XML em Java. Destas foram avaliadas duas, SAX e DOM e optou-se pela utilização da última. Os motivos pelos quais essa decisão foi tomada são explicados nos pontos seguintes.

## SAX

Este API para *parsing* de XML em Java, cujo nome são as iniciais de *Simple API for XML*. Se o *parser* implementar todas as opções do SAX providenciará um suporte completo à actual versão de XML. Esta API funciona de forma a disponibilizar os dados do documento XML à medida que realiza a análise ao ficheiro XML. Isto significa que cabe ao programador desenvolver estruturas para armazenar os dados do XML. Por esta razão a SAX é muito rápida e eficiente, no entanto torna os programas mais difíceis de conceber quando é necessário representar as estruturas definidas em XML em estruturas intermédias para manipulação [4].

Basicamente os passos para utilizar esta API são os seguintes:

- (1) `public class MinhaClasse extends DefaultHandler`
  - (2) `XMLReader parser = XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");`
  - (3) `ContentHandler handler = new MinhaClasse();`
  - (4) `parser.setContentHandler(handler);`
- Na Linha (1), é criada uma classe, por exemplo *MinhaClasse*, que estende a classe *DefaultHandler*. Esta classe terá de implementar os métodos que irão processar o documento. Destacam-se de entre esses métodos os *startElement* e *endElement* que são activados quando, respectivamente, um elemento novo é processado e quando esse elemento acabou.

---

<sup>15</sup> [Http://xml.apache.org/xerces2-j/](http://xml.apache.org/xerces2-j/)

- Na Linha (2), é criado um objecto *XMLReader* onde é inicializado o *parser* que vamos utilizar:
- Na Linha (3), de seguida cria-se um novo *ContentHandler* a partir da *MinhaClasse* e atribui-se esse handler ao *parser*.
- Na Linha (4), executa-se depois o método *parse* do *XMLReader* e os métodos que foram implementados na *MinhaClasse* serão chamados à medida que o documento é processado.

## DOM

O DOM, *Document Object Model*, é como o nome indica um modelo de representação de um documento XML. Ao contrário do SAX, que permite controlo passo a passo, o DOM processa o ficheiro XML de uma só vez. O conteúdo desse ficheiro é representado em forma de árvore, e é através dessa árvore que o programador determina o que o ficheiro contém. Esta abordagem ocupa mais memória pois é necessário armazenar a árvore. A árvore gerada reflecte a estrutura definida no documento XML. O nó raiz representa a *tag* do topo e os filhos as *tags* aninhadas. Por exemplo, um ficheiro com a seguinte definição:

```
<GUI>
  <Size/>
  <MenuBar>
    <Menu>
      <MenuItem/>
      <MenuItem/>
    <Menu/>
    <Menu>
      <MenuItem/>
    </Menu>
  </MenuBar>
  <Button/>
</GUI>
```

É representado pela árvore apresentada na *Figura 8*.

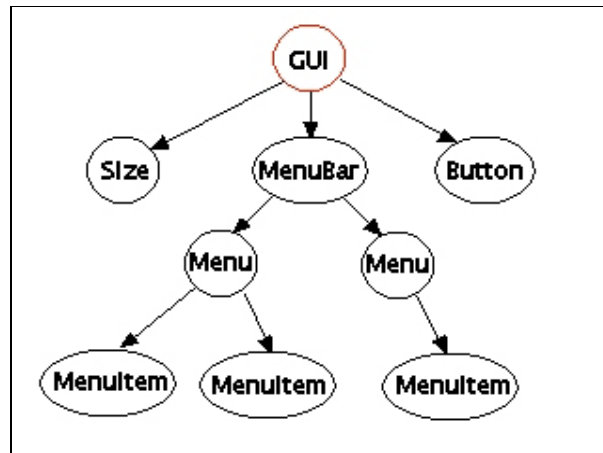


Figura 8 – Árvore gerada que descreve a estrutura do documento.

Esta estrutura em árvore é providenciada pelo DOM, que facilita também os métodos para percorrer a árvore. Pode-se ver no *Exemplo 5* a utilização de alguns desses métodos. Este código permite imprimir no ecrã os nomes dos nós da árvore, que correspondem às *tags* XML, por níveis.

```

private void percorren(Node n)
{
    System.out.println(n.getNodeName()+" "+n.getLocalName());

    if (n.hasChildNodes())
    {
        q.addItem(n.getFirstChild());
    }

    Node nn = n.getNextSibling();

    if (nn!=null)
        percorren(nn);

    if(!q.isEmpty())
        percorren((Node) q.removeHead());
}

```

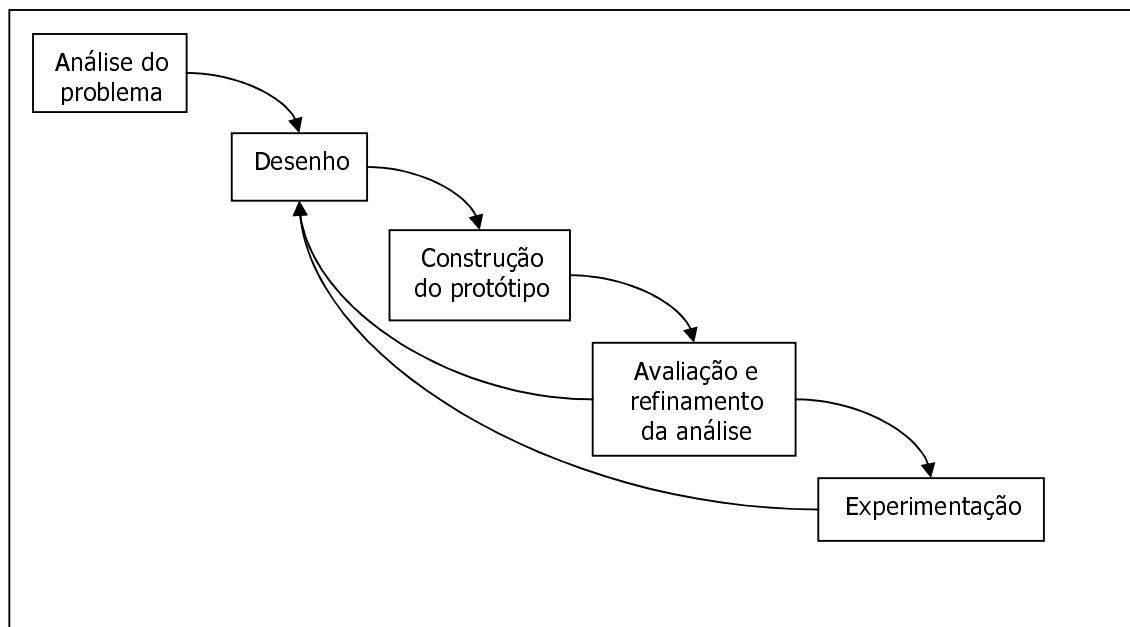
Exemplo 5 – Código Java que percorre uma árvore DOM.

O DOM é um standard do W3C. A versão actual é a chamada de *DOM Level 3*. Esta versão não é ainda completamente suportada pelo Xerces, mas este factor não é limitativo neste projecto. Torna-se mais simples utilizar esta estrutura em árvore para processar os dados do XML do que criar uma estrutura personalizada

usando a SAX. Foi principalmente por este motivo que foi escolhida esta API para a realização deste projecto.

### 3. Trabalho Realizado

O objectivo essencial deste trabalho é simplificar a construção de interfaces gráficas. Para este efeito utilizou-se XML para definir uma linguagem de formatação para GUIs. A interpretação do documento XML e construção da interface são da responsabilidade da *package* Java elaborada. Para o desenvolvimento deste projecto, foi utilizado o *modelo de prototipagem*, cujas fases são apresentadas na *Figura 9*.



**Figura 9 – Esquema das fases do modelo de prototipagem.**

Na fase de análise do problema são recolhidos os dados iniciais necessários ao desenvolvimento do desenho inicial. No projecto em questão, nesta fase, foi feita a análise inicial do gerador de interfaces definidas em XML. Durante a fase de desenho foi feito um esboço inicial, muitas vezes incompleto, que será melhorado ao longo de todo o processo. De seguida foi construído o protótipo inicial e procede-se à sua avaliação. Após esta avaliação existe, na maioria dos casos, a necessidade de refinar a análise, e voltar à fase de desenho. Durante este projecto por várias vezes foi necessário voltar à fase de desenho. Por exemplo quando se julgou necessário trocar a API utilizada para o processamento dos documentos XML. Por último o protótipo foi testado. Caso não sejam cumpridos os requisitos, volta-se para a fase de desenho.

### 3.1 Especificação XML

Para se poder utilizar XML para definir uma interface gráfica foi necessário criar uma especificação de uma linguagem de formatação. Esta linguagem define através de *tags* todos os componentes de um GUI e as acções associadas. Os componentes do GUI são todos os objectos que ele possui tais como: menus, botões, ou caixas de texto. A ordem pela qual os componentes estão definidos no ficheiro XML corresponde à ordem pela qual são posicionados na interface.

A tabela 1 mostra as *tags* definidas e os componentes a que correspondem.



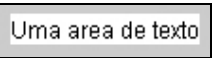

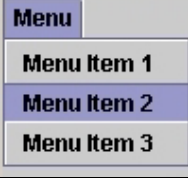
Tags	Componente	Imagem
<b>Controlo</b>		
Button	Botão	
CheckBox	Caixa de Verificação	
<b>Visualização</b>		
ImageLabel & TextLabel	Etiquetas	
TextArea	Área de texto	
TextField	Caixa de texto	
<b>Menus</b>		
MenuBar	Barra de Menus	
Menu	Menu	
MenuItem	Itens de Menu	

Tabela 1 – Tags de componentes.

Foram ainda definidas outras *tags* de controlo, mostradas na tabela 2, que não correspondem a componentes individuais. A *tag GUI* é a que marca o início e o fim da definição da interface. Deve ser inserida no início do documento e fechada após todos os componentes terem sido definidos. Isto significa que todos os componentes são definidos dentro deste elemento. A *OUT* é semelhante à *GUI* pois permite a definição de agrupamentos de componentes, embora a sua utilização seja para definir agrupamentos que dependem de acções tratadas na interface principal.



Tags	Função
GUI	Marca o início da definição da interface.
OUT	Define um componente exterior.
Size	Configura o tamanho da interface.

Tabela 2 – Tags de controlo.

Em conjunto com as *tags* foram definidos atributos. Alguns atributos são gerais, como o *NAME*, mas outros são específicos de determinados componentes. A tabela 3 mostra os atributos e a sua função.

Atributos	Função	Valores Possíveis
<b>Valor</b>		
<b>NAME</b>	Nome do componente	String
<b>LABEL</b>	Etiqueta do componente	
<b>TEXT</b>	Texto que o componente possui	
<b>IMAGE</b>	Caminho para uma imagem (ficheiro jpeg,gif,etc)	
<b>SELECT</b>	Define se o componente está seleccionado	"true" ou "false"
<b>Configuração física</b>		
<b>HEIGHT</b>	Controla a altura da interface	Inteiro
<b>WIDTH</b>	Controla a largura da interface	
<b>COLS</b>	Indica o número de colunas	String
<b>ROWS</b>	Indica as linhas do componente	
<b>ALIGNH</b>	Controla o alinhamento horizontal	
<b>ALIGNV</b>	Controla o alinhamento vertical	"true" ou "false"
<b>BORDER</b>	Indica se existe contorno no componente	
<b>RESIZE</b>	Configura se é possível redimensionar a interface	
<b>VISIBLE</b>	Define a visibilidade de um componente	
<b>ENABLED</b>	Define se um componente está activo.	

Tabela 3 – Atributos.

Encontra-se no **apêndice 1** uma tabela que indica a que componentes correspondem os atributos. Um exemplo de uma definição de um componente com atributos é:

```
<CheckBox NAME="check1" LABEL="Check box" />
```

Neste caso define-se uma caixa de verificação atribuindo-lhe o nome *check1* e a etiqueta *Check box*. Para uma explicação detalhada sobre como utilizar as *tags* e atributos é necessário consultar o manual de utilização [5].

### 3.1.1 Acções

Um GUI requer que aos componentes sejam associadas acções. Algumas acções limitam-se a despoletar funcionalidades da aplicação e outras à abertura de outros componentes. As associações permitidas pela ferramenta desenvolvida são:

- (1) Acções para abertura de outros componentes;
- (2) Acções *standard* como: sair da aplicação, enviar uma mensagem, etc.
- (3) Acções a serem tratadas pelo resto da aplicação como: abrir um diálogo de escolha de ficheiros, accionamento de um botão, etc.

Os atributos associados à definição de acções são apresentados na *Tabela 4*.

Atribuição de acções		
<b>ACTION</b>	Atribui uma acção a um componente	String
<b>CLASS</b>	Define a classe onde se encontra o método definido pelo atributo <i>METH</i>	
<b>METH</b>	Método que realiza a acção associada ao componente	
<b>MSG</b>	Define o texto de uma caixa de mensagens	
<b>TITLE</b>	Define o título de uma caixa de mensagens	"confirm" ou "input"
<b>TYPE</b>	Define o tipo de caixa que aparece	

Tabela 4 – Atributos para atribuir acções.

O atributo *ACTION* permite atribuir acções pré-definidas a certos componentes, como o botão, sem necessidade de introdução de código extra por parte do programador. Podem-se ver de seguida quais os valores do atributo que correspondem a estas acções.

Valor do ACTION	Resultado da acção	Tipo
<b>msg</b>	Envia uma mensagem ao utilizador. O conteúdo e título dessa mensagem são controlados pelos atributos <i>MSG</i> e <i>TITLE</i>	(2)
<b>exit</b>	Termina a aplicação sem pedir confirmação	(2)
<b>Nome de um componente definido pela tag OUT</b>	Mostra um componente dependente	(1)

Tabela 5 – Valores do atributo ACTION que não necessitam de adição de código Java.

Por exemplo, um botão que ao ser pressionado termina a execução da aplicação define-se da forma seguinte:

```
<Button NAME="botao" LABEL="Sair" ACTION="exit" />
```

A possibilidade de mostrar um componente dependente foi adicionada para situações em que é necessário, por exemplo, mostrar um painel de configuração ao utilizador. Pode-se por exemplo atribuir a um item de menu a função de abrir um painel desse tipo. Este mecanismo foi criado para facilitar a programação da aplicação. Apesar de não existirem muitas acções suportadas, o código que as controla é facilmente extensível.

Para além destas acções independentes do código cliente (resto da aplicação que utiliza o GUI), existem outras que necessitam de métodos cliente para comunicação de dados. Na *tabela 6* são indicados os valores do *ACTION* que necessitam de métodos no lado do cliente.

Valor do ACTION	Resultado da acção	Tipo
<b>open</b>	Abre uma caixa de escolha de ficheiros	(3)
<b>saveas box</b>	Abre uma caixa de introdução de dados ou uma caixa de confirmação	(3)

**Tabela 6 – Valores do ACTION que necessitam de métodos cliente.**

Este mecanismo foi implementado para lidar com situações em que a aplicação necessita de receber valores do utilizador através de componentes gráficos. Ao definir um botão para abrir, por exemplo, uma caixa de escolha de ficheiro é também necessário definir através dos atributos *CLASS* e *METH* a classe e respectivo método que irá receber o valor de confirmação. O método cliente deve possuir parâmetros de acordo com o tipo da acção. Neste caso o método deveria ter como parâmetro de entrada uma variável do tipo *File*.

Para controlar acções mais simples em que não se necessita de dados do utilizador, é possível utilizar apenas os atributos *CLASS* e *METH*. Neste caso, o método a utilizar não deve possuir parâmetros de entrada. Esta funcionalidade é útil quando se quer atribuir directamente a um botão, por exemplo, uma acção sempre

que este é pressionado. Encontra-se no **apêndice 2** um exemplo de uma definição de uma interface utilizando esta especificação.

### 3.2 Métodos

Depois de definida a interface, usando a especificação desenvolvida em XML, para fazer a ligação com o código é necessário utilizar a package desenvolvida. Esta package fornece os métodos necessários para construir a interface partindo do documento XML. Depois de ter sido incluído o *import* da package o procedimento é o seguinte:

- ```
(1) BuildGUI gui=new BuildGUI(nome_do_ficheiro);  
(2) gui.build();
```

Na linha (1) é criado um objecto da classe *BuildGUI* cujo construtor recebe o nome do ficheiro XML com a definição de uma interface. Na linha (2) é invocado o método *build()* que processa os componentes lidos do documento XML e desenha a interface. Depois da interface gráfica ter sido construída é necessária uma forma de interagir com os componentes, por exemplo para verificar ou modificar valores. Foram então construídos métodos para fornecer essas funcionalidades. A tabela seguinte lista os métodos divididos por categorias.

| Método                        | Função<br>(em relação a um componente)  |
|-------------------------------|-----------------------------------------|
| <b>Verificação de estados</b> |                                         |
| isComponentEnabled            | Verifica se está activado <sup>16</sup> |
| isComponentValid              | Verifica se é válido ou seja se existe  |
| isVisible                     | Indica se está visível                  |
| <b>Alteração de estados</b>   |                                         |
| setComponentEnabled           | Activa ou desactiva                     |
| setVisible                    | Esconde ou torna visível                |
| <b>Obtenção de valores</b>    |                                         |
| getComponentValue             | Obtém o valor                           |
| <b>Modificação de valores</b> |                                         |
| setComponentValue             | Muda o valor                            |
| addComponentText              | Adiciona texto                          |

Tabela 7 – Métodos de interacção com componentes.

---

<sup>16</sup> Considera-se um componente activado quando é possível ao utilizador interagir com ele.

Depois de alguma experimentação concluiu-se que também era necessário uma forma de interagir com o utilizador que não dependesse directamente de um componente. Para esse efeito foi criado o método *msgBox* que abre uma caixa de mensagem, de introdução de dados, ou de confirmação.

Procurou-se que estes métodos fossem genéricos, para que não fosse necessário ao programador utilizar métodos diferentes para cada componente. No entanto existem métodos que não suportam todos os componentes, por exemplo o *addComponentText*, que só suporta componentes de texto multi-linha, para mais pormenores acerca do suporte é necessário consultar o manual de utilização [5].

### 3.3 Classes desenvolvidas

A aplicação foi desenvolvida na linguagem Java. Pode-se ver na *Figura 10* o diagrama de classes do *package*. As classes nativas de Java não estão representadas. Este conjunto de classes resulta, principalmente, de opções tomadas durante o desenvolvimento e não de um desenho prévio.

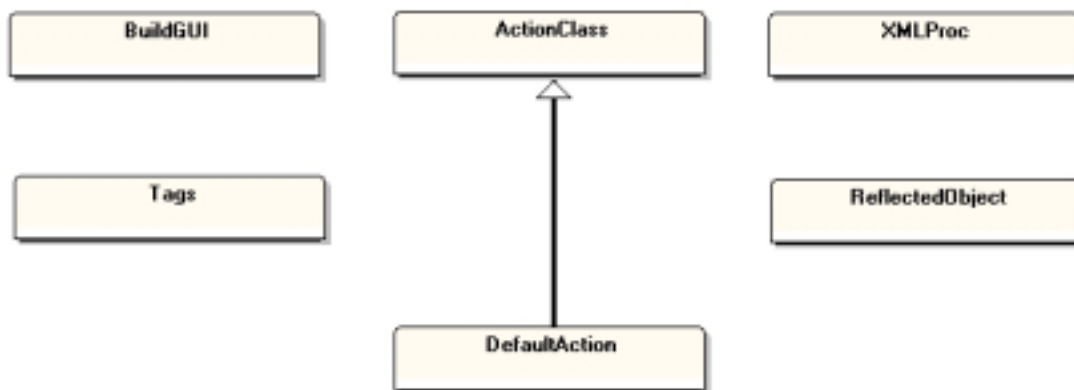


Figura 10 – Diagrama de classes simples.

A classe principal é a classe *BuildGUI* que se encarrega do processo de construção da interface, e possui os métodos para a interacção com a interface. As classes *ActionClass* e *DefaultAction* encarregam-se de atribuir as acções aos componentes. A classe *ReflectedObject* implementa os métodos necessários para a utilização genérica do *Reflection*. A classe *XMLProc* é responsável pela ligação ao *parser* XML para processar o documento. A classe *Tags* define as *tags* e valores de

atributos suportados. Encontra-se no **apêndice 3** um diagrama de classes mais detalhado no qual são mostrados os métodos e atributos<sup>17</sup> das classes.

### 3.4 Processamento dos componentes

O processamento dos componentes é feito dentro da classe *BuildGUI*. Inicialmente o documento XML é processado pelo *parser*, utilizando a API DOM [4]. É produzido um *Document*, que é uma interface Java representativa do documento XML na forma de árvore. Em seguida essa árvore é percorrida de forma a expôr e verificar os componentes e respectivos atributos, construindo-os e adicionando-os à interface. Este processo é feito da seguinte forma:

1. Partindo do nó principal é verificado se define sub-componentes (casos dos elementos *OUT* e GUI).
2. Caso defina é construído um vector (classe *Vector* em Java) com os componentes imediatamente abaixo desse nó.
3. O vector é percorrido e os elementos são verificados:
  - a. No caso de um elemento *OUT* é criado outro vector com os componentes principais desse elemento e recomeça-se o ponto 3.
  - b. Caso seja um elemento simples (componente não dependente) que defina um componente são extraídos os valores dos atributos.
  - c. Se o elemento define sub-componentes, como um *MenuBar*, esses são processados de forma recursiva.
  - d. Cada elemento após ter sido processado é adicionado à interface.
4. Após o processamento dos componentes é mostrada a interface.

Os componentes definidos no ficheiro são criados imediatamente após terem sido processados. Desta forma a interface leva mais tempo a ser gerada mas evitam-se esperas prolongadas quando está a ser utilizada. No entanto os

---

<sup>17</sup> Também designados por variáveis de instância, ou de classe.

componentes criados por acções despoletadas como, por exemplo, um componente para escolha de ficheiros, são criados à medida que são solicitados.

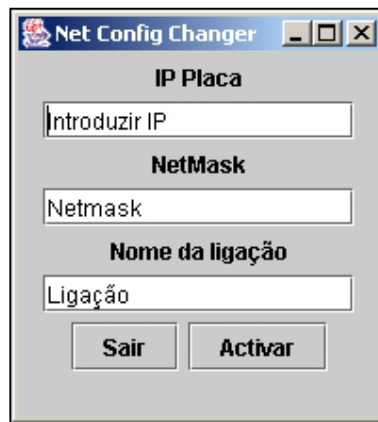
A atribuição de acções aos componentes é feita durante a fase 3.b. São adicionados objectos *DefaultAction* (acção definida com *ACTION*) ou *ActionClass* (Acção simples definida com *CLASS* e *METH*), como gestores de acções, aos componentes que têm acções associadas. Quando um componente é accionado, o gestor dele responde invocando o método definido, com o atributo *METH*. A invocação do método é feita através de um objecto do tipo *ReflectedObject* que implementa a capacidade *Reflection* da tecnologia Java.

## 4. Programas Exemplo

Neste capítulo são apresentadas algumas aplicações realizadas utilizando a *package* desenvolvida no projecto.

### 4.1 Aplicação 1

Esta aplicação, simples em funcionalidades, demonstra algumas das capacidades da *package* desenvolvida. O objectivo da aplicação é: mudar o endereço *ip* e a *netmask* de um computador em MS-Windows. Pode-se ver na *Figura 11* a interface gerada.



**Figura 11 – Interface da aplicação exemplo 1.**

O código Java e a definição XML estão disponíveis no **apêndice 6**. Existem alguns pontos de interesse a realçar nesta aplicação. A aplicação recorre a um programa exterior, o utilitário do Windows *netsh* para modificar o endereço *ip*. O *netsh* é invocado dentro do método atribuído ao botão *Activar*. De seguida a aplicação espera pelo fim da execução do *netsh* e informa o utilizador.

### 4.2 Aplicação 2

Esta aplicação, cuja interface é mostrada na *Figura 12*, demonstra muitas das funcionalidades da *package* desenvolvida. Pode-se ver na definição no **apêndice 6** que a interface já é razoavelmente complexa, não exigindo no entanto código da parte do programador para ser gerada. Realça-se que esta aplicação foi



desenvolvida por um programador sem necessidade de utilizar conhecimentos aprofundados de programação de interfaces gráficas. A aprendizagem da especificação XML e dos métodos da *package* foi feita consultando o manual de utilizador.

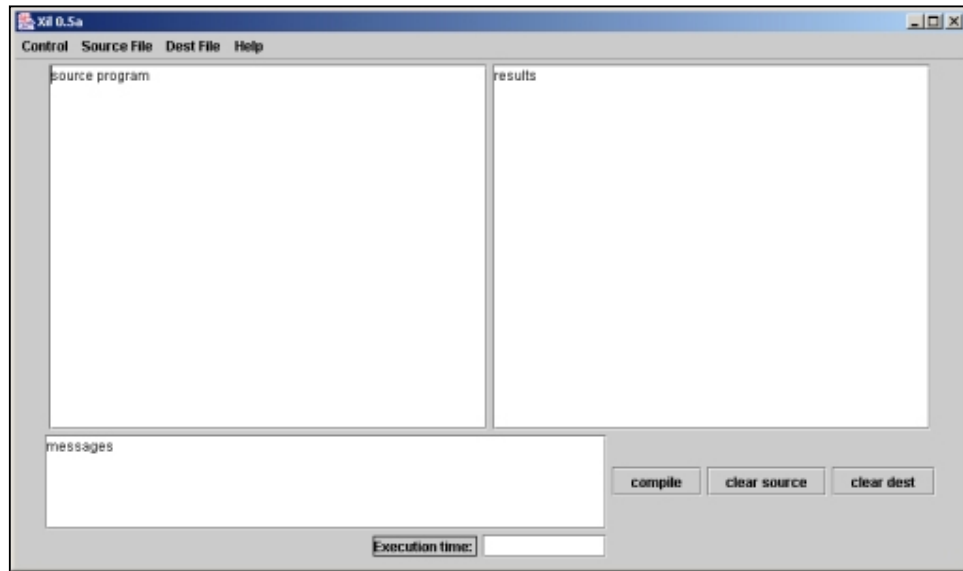


Figura 12 – Interface da aplicação exemplo 2.

São aqui demonstrados os vários tipos de acções possíveis para os componentes. Os menus da aplicação demonstram os vários tipos de acções possíveis. No menu ilustrado na *Figura 13*, vê-se um item de menu que acciona a saída do programa.

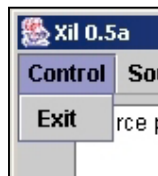


Figura 13 – Menu Control.

É uma acção *standard*, tipo (2), e não requer inserção de código pelo programador. A definição em XML para esta acção é a seguinte:

```
<MenuItem NAME="Exit" LABEL="Exit" ACTION="exit"/>>
```

No menu *Source File* e *Dest File*, *Figura 14*, temos o exemplo de acções do tipo (3). Acções que necessitam de tratamento pelo programador.

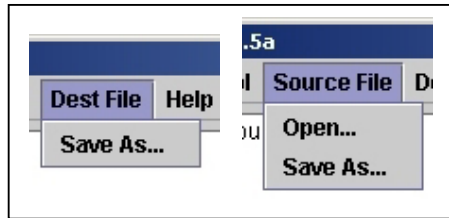


Figura 14 – Menu *Source File*.

Na definição em XML do item *Open*, pode-se ver que o método responsável por tratar a acção é o *openFile* pertencente à classe *Actions*.

```
<MenuItem NAME="FileOpen1" LABEL="Open..." ACTION="open"
CLASS="Actions" METH="openFile"/>
```

Pode-se ver na *Figura 15* o diálogo de escolha ficheiros que é mostrado quando o item *Open* é accionado.

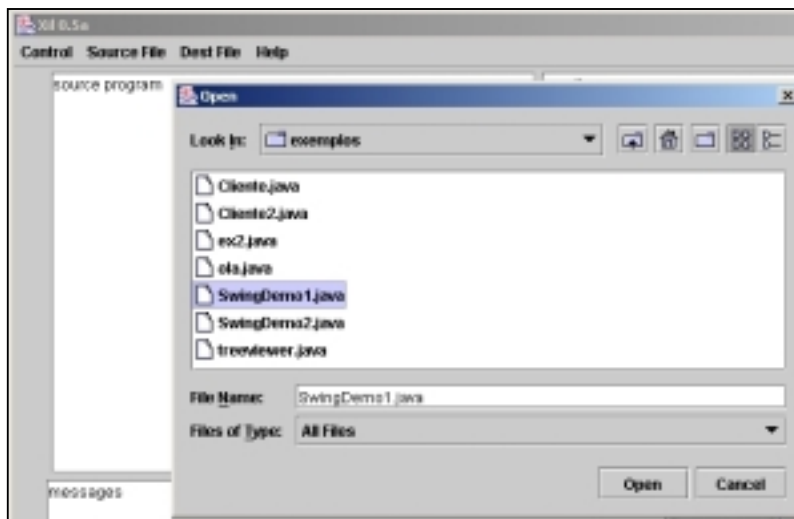


Figura 15 – Escolha de ficheiros.

Os itens do menu *Help*, *Figura 16*, têm associadas acções do tipo (1), abertura de outros componentes. Tanto o item *About* como o *Help* abrem outras janelas, definidas com *OUT* que mostram áreas de texto com mensagens. O

accionamento desta acções não necessitou que o programador adicionasse código Java.

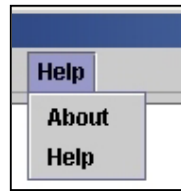


Figura 16 – Menu Help.

Por exemplo a definição do *About*:

```
<MenuItem NAME="About" LABEL="About" ACTION="AboutWindow" />
```

Indica que deve ser mostrado um componente dependente de nome *AboutWindow*. A definição deste é:

```
<OUT NAME="AboutWindow" >  
  <Size WIDTH="400" HEIGHT="100"/>  
  <TextArea NAME="text1" ROWS="3" COLS="30" TEXT="Author: Gil  
  Moreira Adviser: Joao M. P. Cardoso. Universidade do Algarve, July, 2003."/>  
</OUT>
```

Após este item ter sido accionado a janela da *Figura 17* é mostrada.

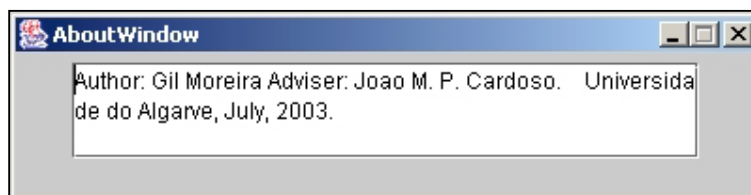


Figura 17 – Janela *AboutWindow*.

O objectivo desta aplicação é mostrar uma árvore sintáctica correspondente ao código Java (representado na área de texto do lado esquerdo), na área de texto do lado direito da interface. O código é obtido através de um ficheiro ou introduzido manualmente na área de texto do lado esquerdo da interface. São também

apresentadas mensagens de estado (área de texto no canto inferior esquerdo), e tempo de compilação (área de texto em baixo).

O botão *Compile* acciona a execução do analisador lexical e sintáctico de código Java que recebe o programa Java (na área de texto da esquerda) e imprime a árvore sintáctica na área de texto da direita. O tempo de execução necessário aparece no final na caixa em baixo, como se pode ver na *Figura 18*.

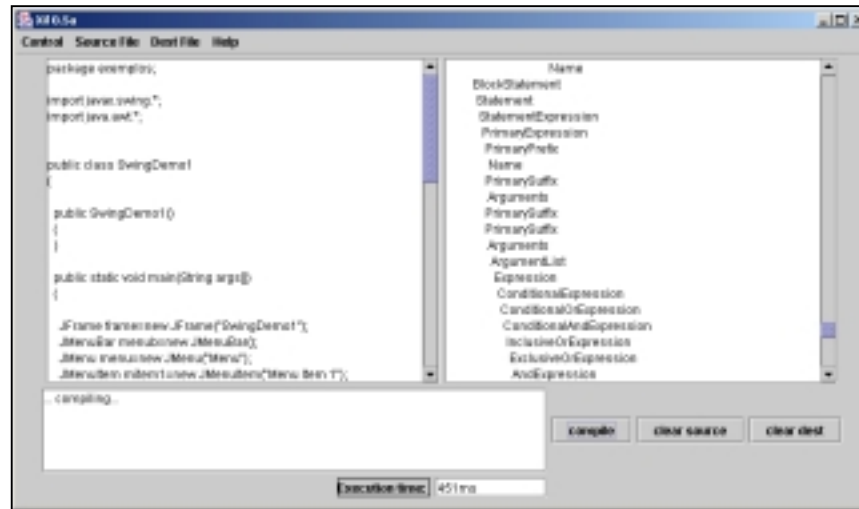


Figura 18 – Execução de um *Compile*.

O código completo da aplicação é incluído no **apêndice 6**. Está dividido em dois ficheiros. O ficheiro *principal.java* que ordena a construção da interface e o *Actions.java* que contém o código dos métodos atribuído aos componentes.

### 4.3 Aplicação 3

A aplicação exemplo 3 demonstra como utilizar caixas de verificação. Na interface gráfica visível na *Figura 19*, podem-se ver várias caixas de verificação que controlam o funcionamento da aplicação. O objectivo da aplicação é executar operações, ao pressionar o botão *GO!*, sobre o texto inserido nos campos de texto *String 1* e *String 2*. O resultado dessas operações é colocado no campo de texto *Result*. As operações que se podem fazer são controladas pelas caixas de verificação. As caixas *Upper Case* e *Lower Case* controlam se o texto é apresentado em maiúsculas ou minúsculas. Só uma destas caixas pode estar seleccionada ao

mesmo tempo. Quando se selecciona uma das caixas a outra deixa de estar seleccionada.

A caixa de verificação *Concat* controla o estado do campo de texto *String 2*, quando ela está seleccionada o campo fica activo e o texto aí inserido é concatenado com o texto do campo *String 1*.

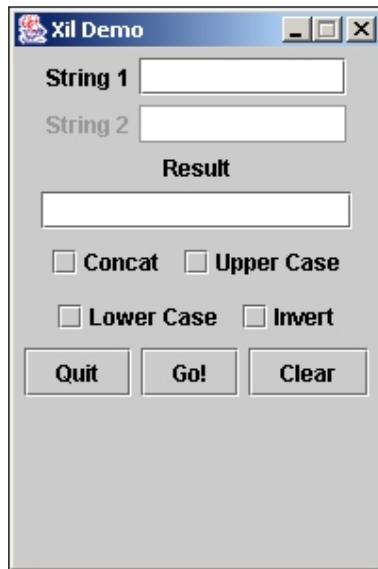


Figura 19 – Aplicação Exemplo 3.

Quando a caixa *Invert* está seleccionada o texto apresentado no campo *Result* aparece invertido. O botão *Clear* limpa o texto de todos os campos de texto e o botão *Quit* termina a aplicação. O código Java e a definição XML desta aplicação encontram-se no **apêndice 6**.

## 5. Conclusões e Trabalho Futuro

O objectivo principal deste projecto foi a simplificação da construção de interfaces gráficas. O trabalho realizado consistiu na definição de uma linguagem de formatação de interfaces gráficas em XML, e uma *package* Java para interpretar essa linguagem. Foi provado que é possível simplificar a construção de interfaces utilizando XML e Java. A linguagem de formatação foi desenvolvida para que fosse expressiva, clara e de forma a possuir o estritamente necessário. Pois para tornar fácil a construção de GUIs é necessário que esta especificação seja fácil e intuitiva. A *package* foi construída tendo em conta os mesmos princípios de expressividade e clareza.

Considera-se que o objectivo principal, simplificar a construção de interfaces gráficas, foi cumprido, tal como os objectivos secundários, aprendizagem de Java e conceitos de XML. No entanto, apesar da *package* desenvolvida cumprir os objectivos é possível melhorá-la. Destacam-se as seguintes possibilidades:

- Melhorar o mecanismo de posicionamento dos componentes, de forma a ser possível controlar com exactidão onde estes são colocados.
- Aumentar o número de componentes suportados.
- Melhorar a comunicação com os componentes de texto. De forma a evitar elevados tempos de resposta quando se colocam grandes quantidades de texto num destes componentes.
- Estudar a utilização de DTDs para validar o documento XML durante a definição da interface.
- Estudar a viabilidade de utilizar a *package* com *applets*.

Outras melhorias seriam externas à própria *package* como, por exemplo, criar uma ferramenta visual que permitisse a construção do GUI com geração posterior do código XML correspondente.

## Referências

[1] Erik T. Ray, Christopher R. Maden, ***Learning XML***, O'Reilly & Associates; 1st edition February 2001

[2] James Elliott, Robert Eckstein (Editor), Marc Loy, David Wood, Brian Cole, ***Java Swing***, O'Reilly & Associates; 2nd edition November 1, 2002

[3] Glen McCluskey, “**Core Java Technologies Tech Tips January 10,2003**”, <http://developer.java.sun.com>

[4] Elliotte Rusty Harold, ***Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX***, Addison Wesley Professional; 1st edition November 5, 2002

[5] Gil Moreira, “**Manual de Utilização da Package Xil 0.5A**”, Universidade do Algarve, Julho 2003

[6] António José Mendes, Maria José Marcelino, ***Fundamentos de Programação em Java 2***, Editora FCA 2002

[7] Mark Johnson, “**Programming XML in Java**”, JavaWorld, Março 2000, <http://javaworld.com>

[8] Mark Johnson, “**XML for the absolute beginner**”, JavaWorld, Abril 1999, <http://www.javaworld.com>

# Apêndices



## Apêndice 1 – Código que define uma interface em Swing

```
import javax.swing.*;
import java.awt.*;

public class SwingDemo1
{
    public static void main(String args[])
    {
        JFrame frame=new JFrame("Nome da janela");
        JMenuBar menub=new JMenuBar();
        JMenu menu=new JMenu("Menu");
        JMenuItem mitem1=new JMenuItem("Menu Item 1");
        JMenuItem mitem2=new JMenuItem("Menu Item 2");
        JMenuItem mitem3=new JMenuItem("Menu Item 3");
        JButton botao=new JButton("Botao");
        JTextField tfield=new JTextField("Texto");
        JTextArea tarea=new JTextArea("Uma area de texto");
        JCheckBox cbox=new JCheckBox("CheckBox");
        JLabel label=new JLabel("Label");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new FlowLayout());

        menu.add(mitem1);
        menu.add(mitem2);
        menu.add(mitem3);

        menub.add(menu);

        frame.setJMenuBar(menub);

        frame.getContentPane().add(botao);
        frame.getContentPane().add(tfield);
        frame.getContentPane().add(cbox);
        frame.getContentPane().add(label);
        frame.getContentPane().add(tarea);
        frame.setSize(300,300);
        frame.show();
    }
}
```

## Apêndice 2 – Código que adiciona uma acção a um botão

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class SwingDemo2 implements ActionListener
{

    public static void main(String args[])
    {

        JFrame frame=new JFrame("SwingDemo2");
        JButton botao=new JButton("Botao");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new FlowLayout());

        botao.addActionListener(new SwingDemo2());

        frame.getContentPane().add(botao);
        frame.setSize(300,150);
        frame.show();

    }

    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Carregou no botão! Obrigado");
    }

}
```

### Apêndice 3 – Tabela de componentes e atributos

Atributos	Tags/Palavras-Chave										
	GUI	Size	Button	CheckBox	ImageLabel & TextLabel	TextArea	TextField	MenuBar	Menu	MenuItem	OUT
ACTION			✓	✓						✓	
ALIGNH					✓						
ALIGNV					✓						
BORDER					✓						
CLASS			✓	✓						✓	
COLS						✓	✓				
ENABLED			✓	✓	✓	✓	✓	✓	✓	✓	✓
HEIGHT		✓									
IMAGE									✓		
LABEL			✓	✓						✓	
METH			✓	✓						✓	
NAME	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
RESIZE	✓										
ROWS						✓					
SELECT				✓							
TEXT					✓	✓	✓				
VISIBLE			✓	✓	✓	✓	✓	✓	✓	✓	
WIDTH		✓									

## Apêndice 4 – Exemplo de uma interface definida em XML

```
<?xml version="1.0"?>

<GUI>
  <Size WIDTH="300" HEIGHT="300"/>

  <MenuBar NAME="TopMenu">

    <Menu NAME="File" LABEL="File">
      <MenuItem NAME="FileOpen" LABEL="Open..." ACTION="open"
CLASS="principal" METH="openFile"/>
      <MenuItem NAME="FileSave" LABEL="Save"/>
      <MenuItem NAME="FileSaveAs" LABEL="Save As..."/>
      <MenuItem NAME="FileExit" LABEL="Exit" ACTION="exit"/>
    </Menu>

    <Menu NAME="Edit" LABEL="Edit">
      <MenuItem NAME="EditUndo" LABEL="Undo"/>
      <MenuItem NAME="EditCut" LABEL="Cut"/>
      <MenuItem NAME="EditPaste" LABEL="Paste"/>
      <MenuItem NAME="EditDelete" LABEL="Delete"/>
    </Menu>

    <Menu NAME="Help" LABEL="Help">
      <MenuItem NAME="HelpAbout" LABEL="About"/>
      <MenuItem NAME="HelpTutorial" LABEL="Tutorial"/>
    </Menu>

  </MenuBar>

  <OUT NAME="Config">
    <Size WIDTH="300" HEIGHT="300"/>
    <CheckBox NAME="Check1" LABEL="CheckBox 1" SELECT="false"/>
    <CheckBox NAME="Check2" LABEL="CheckBox 2" SELECT="false"/>
    <CheckBox NAME="Check3" LABEL="CheckBox 3" SELECT="true"/>
  </OUT>

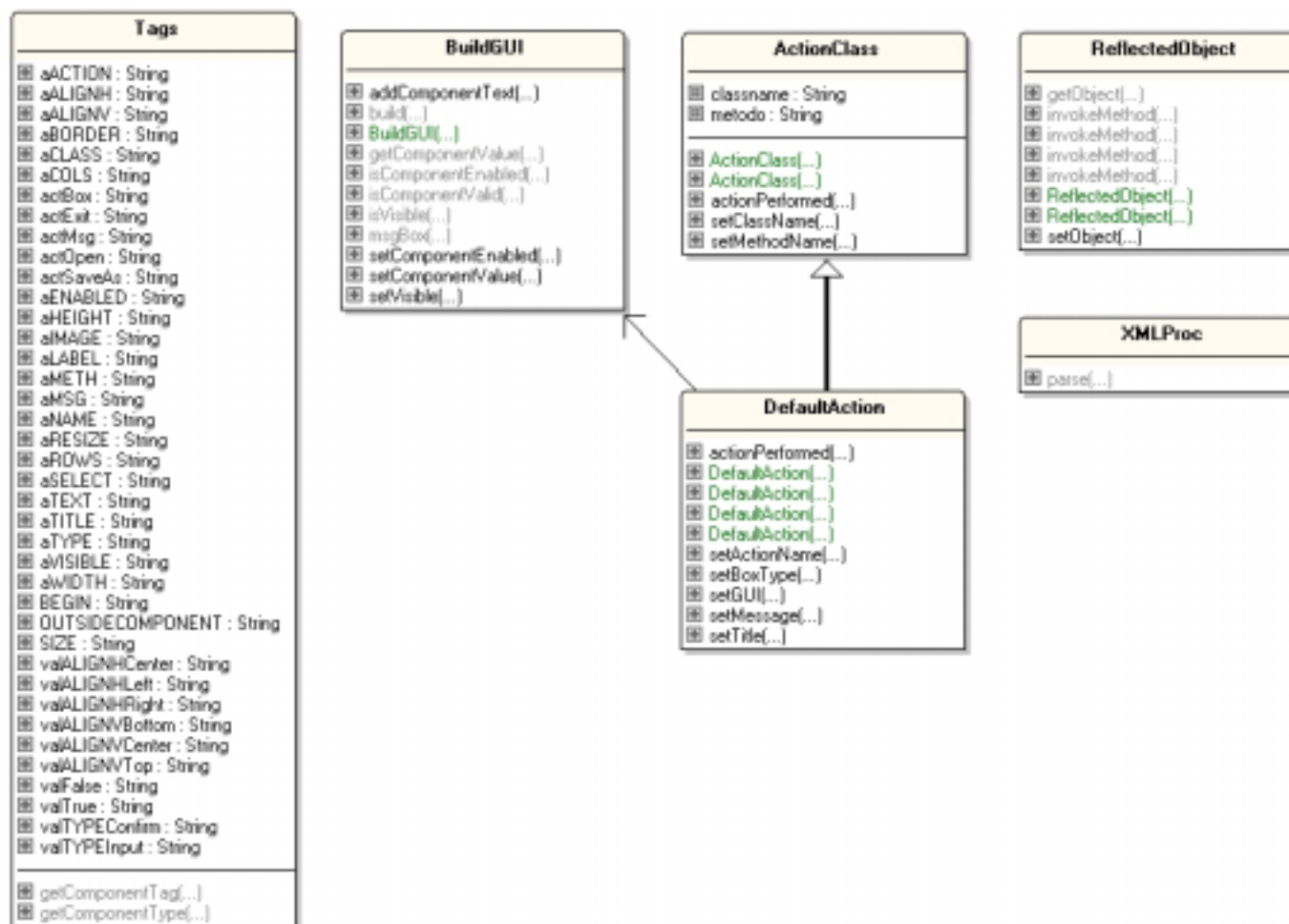
  <CheckBox NAME="Check1" LABEL="CheckBox 1" CLASS="principal" METH="check1_click"
SELECT="true"/>
  <CheckBox NAME="Check2" LABEL="CheckBox 2" SELECT="false" VISIBLE="true"/>

  <Button NAME="botao1" LABEL="Botao 1" CLASS="principal" METH="botao1_click"/>
  <Button NAME="botao2" LABEL="Botao 2" CLASS="principal" METH="botao2_click"/>
  <Button NAME="botao3" LABEL="Botao 3" CLASS="principal" METH="botao3_click"/>

  <TextField NAME="text1" TEXT="Caixa" COLS="20"/>
  <TextField NAME="text2" TEXT="De Texto" COLS="20"/>
  <TextField NAME="text3" TEXT="Res" COLS="20"/>

</GUI>
```

## Apêndice 5 – Diagrama de classes



Gerado pela ferramenta *ESS Model*. [www.essmodel.com](http://www.essmodel.com).

## Apêndice 6 – Código e definições das aplicações exemplo

### Aplicação 1

#### Definição XML:

```
<?xml version="1.0" encoding="UTF-16"?>

<GUI NAME="Net Config Changer">

    <Size WIDTH="200" HEIGHT="220"/>
    <TextLabel NAME="label1" TEXT="IP Placa"/>
    <TextField NAME="ip" TEXT="Introduzir IP" COLS="15"/>
    <TextLabel NAME="label2" TEXT="NetMask"/>
    <TextField NAME="nmask" TEXT="Netmask" COLS="15"/>
    <TextLabel NAME="label3" TEXT="Nome da ligação"/>
    <TextField NAME="nomenet" TEXT="Ligação" COLS="15"/>
    <Button NAME="botao2" LABEL="Sair" ACTION="exit"/>
    <Button NAME="botao1" LABEL="Activar" ACTION="box" TYPE="confirm" MSG="Activar
Configuração?" TITLE="?" CLASS="netchange.NetChange" METH="botaoClick"/>

</GUI>
```

#### Código Java:

O código relativo à package *aparece destacado*.

```
import xil.*;
import java.io.*;
import java.lang.Object;
import java.lang.Runtime;

/**
 * <p>Title: NetChange</p>
 * <p>Description: Programa que muda o ip e a netmask de um computador com o so ms-
windows</p>
 */

public class NetChange
{
    // atributo da classe que irá conter a interface
    private static BuildGUI g=null;

    public static void main(String args[])
    {
        // processamento do ficheiro xml
        g=new BuildGUI("gui.xml");
        // geração da interface
        g.build();
    }
}
```

```

// método que verifica algumas regras de validação para o ip
private static boolean checkIP(String ip)
{
    if (ip!=null && ip.length()<=15 && ip.length()>=7)
    {
        int pindex[]={-1,0,0,0,0};
        int a=0;

        do
        {
            a++;
            pindex[a] = ip.indexOf(".", pindex[a - 1]+1);
        }
        while(pindex[a]!=-1 && a<4);

        if (a<4)
            return false;

        Integer v;
        pindex[pindex.length-1]=ip.length();

        for(int i=0;i<pindex.length-1;i++)
        {
            try
            {
                v = new Integer(ip.substring(pindex[i] + 1, pindex[i + 1]));
            }
            catch(NumberFormatException e)
            {
                return false;
            }

            if(v.intValue()<1 || v.intValue()>254)
                return false;
        }

        return true;
    }
    return false;
}

// método que lida com o accionamento do botao
public void botaoClick(String res)
{
    if (res.equals("0")) // se o utilizador tiver carregado no OK
    {
        File fich=new File("script.ini");
        String val=g.getComponentValue("ip"); //obtem o número ip introduzido
        if (checkIP(val)!=true) // se o número for inválido
            g.msgBox("Mensagem","ip invalido","msg"); //envia uma mensagem ao utilizador
        else
        {
            try
            {
                // abertura do ficheiro para criação da configuração para o netsh
                FileWriter fwrite = new FileWriter(fich);
            }
        }
    }
}

```

```

BufferedWriter bwrite=new BufferedWriter(fwrite);
// inicio da construção do ficheiro de config
bwrite.write("pushd interface ip");
bwrite.newLine();

bwrite.write("set address name=\""+g.getComponentValue("nomenet")+"\"
source=static addr="); // obtem e escreve no ficheiro o valor da caixa nomenet, que indica o nome
da ligação de rede

// obtem e escreve no ficheiro o valor da mascara de rede
bwrite.write(val+" mask="+g.getComponentValue("nmask"));

bwrite.newLine();

bwrite.write("popd");

bwrite.close(); // fecha o ficheiro

// inicia o utilitario netsh do windows com o ficheiro como parametro
Process procnetsh=Runtime.getRuntime().exec("netsh -f script.ini");

if (procnetsh!=null)
{
    try
    {
        procnetsh.waitFor(); // espera pelo fim da execução do comando
        g.msgBox("Mensagem","Comando executado","msg"); // envia mensagem ao
utilizador
    }
    catch (InterruptedException e)
    {
        System.err.println("Erro ao executar o comando");
    }
}

}
catch(IOException e)
{
    System.out.println("Não foi possivel abrir o ficheiro "+fich);
}
}

}
}
}

```



## Aplicação 2

### Definição XML:

```
<?xml version="1.0"?>

<GUI NAME="Xil 0.5a">

  <Size WIDTH="850" HEIGHT="500"/>

  <MenuBar NAME="TopMenu">

    <Menu NAME="Program" LABEL="Control">
      <MenuItem NAME="Exit" LABEL="Exit" ACTION="exit" />
    </Menu>

    <Menu NAME="File1" LABEL="Source File">
      <MenuItem NAME="FileOpen1" LABEL="Open..." ACTION="open" CLASS="Actions"
METH="openFile"/>
      <MenuItem NAME="FileSaveAs1" LABEL="Save As..." ACTION="saveas" CLASS="Actions"
METH="saveFile1"/>
    </Menu  >

    <Menu NAME="File2" LABEL="Dest File">
      <MenuItem NAME="FileSaveAs2" LABEL="Save As..." ACTION="saveas" CLASS="Actions"
METH="saveFile2" />
    </Menu>

    <Menu NAME="HelpAbout" LABEL="Help">
      <MenuItem NAME="About" LABEL="About" ACTION="AboutWindow" />
      <MenuItem NAME="Help" LABEL="Help" ACTION="HelpWindow" />
    </Menu>

  </MenuBar>

  <TextArea NAME="source" LABEL="source program" ROWS="20" COLS="35" />
  <TextArea NAME="dest" LABEL="results" ROWS="20" COLS="35" />
  <TextArea NAME="msg" LABEL="messages" ROWS="5" COLS="45" />

  <Button NAME="compile" LABEL="compile" CLASS="Actions" METH="compile"/>
  <Button NAME="clear1" LABEL="clear source" CLASS="Actions" METH="clearSrc"/>
  <Button NAME="clear2" LABEL="clear dest" CLASS="Actions" METH="clearDst"/>

  <ImageLabel NAME="tlabel1" TEXT="Execution time: " BORDER="true" />
  <TextField NAME="time" TEXT="" COLS="10" />

  <OUT NAME="AboutWindow" >
    <Size WIDTH="400" HEIGHT="100"/>
    <TextArea NAME="text1" ROWS="3" COLS="30"
    TEXT="Author: Gil Moreira Adviser: Joao M. P. Cardoso.
    Universidade do Algarve, July, 2003."
    />
  </OUT>
```

```

        <OUT NAME="HelpWindow" >
        <Size WIDTH="400" HEIGHT="200"/>
        <TextArea NAME="text2" ROWS="8" COLS="30"
        TEXT="This Graphical User Interface was developed using the Xil0.5 toolkit. Xil0.5 has been
        developed in the University of Algarve by Gil Moreira and Joao M. P. Cardoso."
        />
        </OUT>

</GUI>

```

## Código Java:

O código relativo à *package* aparece **destacado**.

### principal.java

```

// the Xil0.5 toolkit
import xil.*;

/**
 * <p>Title: Example showing some Xil0.5 toolkit features. </p>
 * <p>Description: The example creates a GUI where a Java class can be written or loaded from a file
 * and it generates the concrete syntax tree for the given Java class. </p>
 * <p>Copyright: Copyright (c) 2003 </p>
 * <p>Company: Universidade do Algarve </p>
 * <p>Date: July 2, 2003 </p>
 * @author not attributable
 * @version 1.0
 */
public class principal {

    // the reference to the GUI
    static BuildGUI myGUI;

    // the reference to the parser (see class Actions)
    static JavaParser parser;

    /*
     * Main program which generates the GUI.
     */
    public static void main(String[] s) {
        principal.myGUI = new BuildGUI("gui.xml");
        principal.myGUI.build();
    }
}

```

## Actions.java

```
import java.io.*;
import java.lang.*;

/**
 * <p>Title: Example showing some Xil0.5 toolkit features. </p>
 * <p>Description: The example creates a GUI where a Java class can be written or loaded from a file
 and it generates the concrete syntax tree for the given Java class. </p>
 * <p>Copyright: Copyright (c) 2003 </p>
 * <p>Company: Universidade do Algarve </p>
 * <p>Date: July 2, 2003 </p>
 * @author not attributable
 * @version 1.0
 */
public class Actions {

    /**
     * Method associated with the openFile action defined int the XML file.
     * The method opens a file and redirects its content to the textbox: "source".
     */
    public void openFile(File fich) {
        try {
            FileReader srcFile = new FileReader(fich);
            String source="";
            int char1;
            // add all the chars of the file to a String
            while((char1 = srcFile.read()) != -1) {
                source += (char) char1;
            }
            srcFile.close();
            // outputs the string to the "source" textbox
            principal.myGUI.setComponentValue("source", source);
        } catch (java.io.IOException e) {
            principal.myGUI.addComponentText("msg",e.getMessage());
            principal.myGUI.addComponentText("msg","Exception reading File " +
fich.toString());
        }
    }

    /**
     * Method associated to the compile action int the XML file.
     * The method creates the parser and executes it using the content
     * on the "source" textbox as input.
     * It outputs the abstract syntax tree to the "dest" textbox of the GUI.
     * It also computes the execution time to parse and to create the tree.
     */
    public void compile() {
        principal.myGUI.setComponentValue("msg","... compiling...\n");

        try {
            principal.myGUI.setComponentValue("dest","\n##### Concrete Syntax
Tree:\n");

            // to compute the execution time
            long startTime = System.currentTimeMillis();

```

```

        // create the parser
        principal.parser = new JavaParser(new
StringBufferInputStream(principal.myGUI.getComponentValue("source")));
        // start the parser and returns the root node of the abstract syntax tree
        SimpleNode rootNode = principal.parser.CompilationUnit();
        // to compute the execution time
        long stopTime = System.currentTimeMillis();
        long parseTime = stopTime - startTime;

        // to output to the "time" textbox of the GUI the execution time
principal.myGUI.setComponentValue("time", String.valueOf(parseTime) + "ms");

        // output the abstract syntax tree to the GUI
        // the second argument is used in the method dump only to print the tree
        rootNode.dump(" ", principal.myGUI);

    } catch (ParseException e) {
        // error and warning messages are output to the "msg" textbox of the GUI
principal.myGUI.addComponentText("msg", e.getMessage());
principal.myGUI.addComponentText("msg", "Java Parser (for Java 1.2 code):
Encountered errors during parse.");
    }

}

/*
 * The method associated with the saveFile1 action defined in the XML file.
 * It creates a new file with the content of the "source" textbox of the GUI.
 */
public void saveFile1(File fich) {
    try {
        FileWriter f1 = new FileWriter(fich);
        f1.write(principal.myGUI.getComponentValue("source"));
    } catch (IOException e) {
        // error and warning messages are output to the "msg" textbox of the GUI
principal.myGUI.addComponentText("msg", e.getMessage());
    }
}

/*
 * The method associated with the saveFile2 action defined in the XML file.
 * It creates a new file with the content of the "dest" textbox of the GUI.
 */
public void saveFile2(File fich) {
    try {
        FileWriter f1 = new FileWriter(fich);
        f1.write(principal.myGUI.getComponentValue("dest"));
    } catch (IOException e) {
        // error and warning messages are output to the "msg" textbox of the GUI
principal.myGUI.addComponentText("msg", e.getMessage());
    }
}

/*
 * The method associated with the clearSrc action defined in the XML file.
 * It replaces the "source" textbox of the GUI with the text: "source program".
 */
public void clearSrc() {

```

```
principal.myGUI.setComponentValue("source","source program");
}

/*
 * The method associated with the clearDst action defined int the XML file.
 * It replaces the "dest" textbox of the GUI with the text: "results".
 */
public void clearDst() {
principal.myGUI.setComponentValue("dest","results");
}
}
```

### Aplicação 3

#### Definição XML:

```
<?xml version="1.0" encoding="UTF-16"?>

<GUI NAME="Xil Demo" RESIZE="false">

    <Size WIDTH="200" HEIGHT="300"/>

    <TextLabel NAME="label1" TEXT="String 1" />
    <TextField NAME="text1" COLS="10"/>
    <TextLabel NAME="label2" TEXT="String 2" ENABLED="false"/>
    <TextField NAME="text2" COLS="10" ENABLED="false"/>
    <TextLabel NAME="label3" TEXT="Result" />
    <TextField NAME="text3" COLS="15"/>

    <CheckBox NAME="checkConcat" LABEL="Concat" CLASS="exemplos.Demo1"
METH="checkConcat"/>
    <CheckBox NAME="checkMaius" LABEL="Upper Case" CLASS="exemplos.Demo1"
METH="checkSelectUp"/>
    <CheckBox NAME="checkMin" LABEL="Lower Case" CLASS="exemplos.Demo1"
METH="checkSelectLo"/>
    <CheckBox NAME="checkInv" LABEL="Invert"/>
    <Button NAME="bExit" LABEL="Quit" ACTION="exit"/>
    <Button NAME="bDo" LABEL="Go!" CLASS="exemplos.Demo1" METH="bGo"/>
    <Button NAME="bClear" LABEL="Clear" CLASS="exemplos.Demo1" METH="bClear"/>

</GUI>
```

#### Código Java:

O código relativo à *package* aparece **destacado**.

```
import xil.*;

public class Demo1
{
    static BuildGUI gui=null;

    public static void main(String args[])
    {
        //construir interface
        gui=new BuildGUI("gui.xml");
        gui.build();
    }

    // metodo que inverte uma String
    private String invertString(String s)
    {
        String res = null;
        if (s!=null)
```

```

{
    res=new String();

    for (int i = s.length()-1; i >= 0; i--)
        res = res + s.charAt(i);
    }
    return res;
}

// codigo do botao Go
public void bGo()
{
    String res=gui.getComponentValue("text1"); // apanha a primeira string

    if (gui.getComponentValue("checkConcat").equals("true"))
        res=res+gui.getComponentValue("text2"); //concatena as strings

    if (gui.getComponentValue("checkInv").equals("true"))
        res=invertString(res); //inverte a string

    // verificar as checkboxes upper lower case
    if (gui.getComponentValue("checkMin").equals("true"))
        res=res.toLowerCase(); // converte para lowercase
    else
        if (gui.getComponentValue("checkMaius").equals("true"))
            res=res.toUpperCase(); // converte para uppercase

    // coloca a string resultado no campo
    gui.setComponentValue("text3", res);
}

// codigo do botao clear
public void bClear()
{
    // limpa todas os campos de texto
    gui.setComponentValue("text1", "");
    gui.setComponentValue("text2", "");
    gui.setComponentValue("text3", "");
}

//codigo da checkbox upper
public void checkSelectUp()
{
    // so pode estar uma activa de cada vez por isso
    // se esta tiver sido activada desactiva a outra
    if (gui.getComponentValue("checkMin").equals("true"))
        gui.setComponentValue("checkMin", "false");
}

//codigo da checkbox lower
public void checkSelectLo()
{
    // so pode estar uma activa de cada vez por isso
    // se esta tiver sido activada desactiva a outra
    if (gui.getComponentValue("checkMaius").equals("true"))
        gui.setComponentValue("checkMaius", "false");
}
}

```

```

//codigo da checkbox concat
public void checkConcat()
{
    // se tiver sido activada
    if (gui.getComponentValue("checkConcat").equals("true"))
    {
        // activa a caixa da string 2
        gui.setComponentEnabled("text2", true);
        gui.setComponentEnabled("label2", true);
    }
    else
    {
        // caso contrario desactiva
        gui.setComponentEnabled("text2", false);
        gui.setComponentEnabled("label2", false);
    }
}
}

```