Template Multi-File

Diogo Ribeiro[A84442] e Rui Reis[A84930]

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a84442,a84930}@alunos.uminho.pt

Resumo Construção de uma ferramenta que permite, através do input de *templates* formatados, a criação de árvores de diretorias com ficheiros com conteúdo variável.

1 Introdução e Contextualização

Especialmente no ramo informático, é constante o desenvolvimento de diversos projectos. Um dos pontos vitais para o sucesso de qualquer projecto é uma boa estruturação desde a sua etapa inicial. Uma boa organização representa um bom fluxo de trabalho.

Desta forma, saber desenvolver uma boa estruturas de ficheiros e pastas é essencial para qualquer pessoa que pretenda começar a desenvolver o seu próprio projecto. Os projectos normalmente tendem a não divergir muito de um certo padrão de organização, por isso a tarefa de criar uma estrutura pode ser repetitiva.

Com isso em mente decidimos implementar uma ferramenta que permite a construção deste tipo de estruturas a partir de *template*, tendo sempre assente o polimorfismo de estruturas e variáveis pretendidas.

2 Teoria Essencial

Assumimos que o ficheiro de input é sempre composto pela mesma organização, sendo descrito da seguinte forma:

1. Introdução de metadados.
2. Explicitação da árvore de diretorias.
3. Preenchimento dos ficheiros.

Desta forma, conseguimos assentar a nossa base de conhecimento em pressupostos que nós permitem escolher estruturas de dados que mais eficientemente permitem atingir o objetivo desejado.

2.1 Condições de Contexto

No densevolvimento do nosso projeto conseguimos identificar 5 condições de contexto estritamente necessárias para a aplicação eficiente da ferramente. Cada condição de contexto possui um objetivo bem definido.

- **META:** Estado onde são lidos os metadados e armazenados em num Dicionário, para uso posterior.
- **FOLDER:** Estruturação da árvore de diretorias com o auxilio de uma estrutura de dados eficiente para o efeito.
- **TITLE:** Está que interpreta todos os títulos. Análise todas as linhas começadas com `===` e permite fazer a transição para o respectivo estado.
- **FILERD:** Permite o preenchimento de ficheiros que se encontrar explicitados no estado **TREE**.
- **SKIP:** Visa ignorar qualquer *input* até que se encontre outro título. Este estado tem como objectivo ignorar o preenchimento de ficheiros que não foram explicitados na árvore de diretorias, sendo por essa razão desnecessários.

Finalmente, os estados acima culminam no seguinte autómato, que representa as diferentes interacções macroscópicas ao nível das condições de contexto. Lê-se a *validez* de um ficheiro, como sendo a sua explicitação na árvore de diretorias processada em **TREE**.

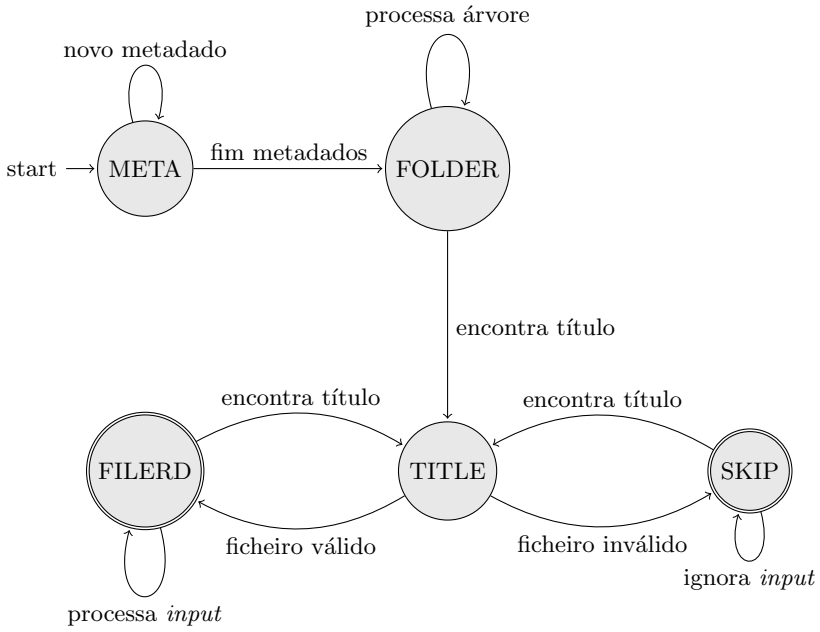

Figura 1. Autómato Determinístico das Condições de Contexto.

2.2 Introdução de Metadados

A introdução de novos metadados funciona por meio de um dicionário, uma tabela de *hash*, que armazena correspondências entre a chave dos metadados e o seu respectivo valor. Ao interpretar um novo metadado, este, e o seu valor, são

armazenados no dicionário, se e só se não existir uma definição prévia utilizando a mesma chave do metadado.

Por definição, há um metadado transversal a todos os *templates*. O metadado `name` é obrigatoriamente passado como argumento à ferramente e pode ser posteriormente utilizado dentro do *template* sem a sua explicitação.

2.3 Explicitação da Árvore de Diretorias

As árvores de diretorias são estruturas que devem ser explicitadas hierarquicamente, ou seja, abaixo de uma diretoria estão todos os ficheiros, ou pastas, que nela estão contidos.

A distinção entre ficheiros e pastas é feita colocando um carácter `/` no final do nome. Ao ser submetido ao sistema uma nova pasta, esta é indexada numa *stack* que vai permitir garantir a qualidade hierarquicamente desta estrutura.

O nível de profundidade de um ficheiro ou pasta é igual ao número de traços, `-`, que surgem antes do respectivo nome. Este nível permite associar cada elemento à pasta na qual ele deve estar contido.

Por exemplo, uma pasta principal contém uma profundidade de 0, um ficheiro contido nesta pasta possui uma profundidade de 1.

Através da *stack* e do nível de profundidade, é possível eficientemente construir uma representação idêntica da árvore de diretorias na forma de árvore N-ária.

2.4 Preenchimento dos Ficheiros

O preenchimento do conteúdo dos ficheiros é feito por meio de um dicionário, que vai associar a cada nome de ficheiro o respectivo conteúdo. Conteúdo este que já terá sido processado, substituindo todos os metadados pelos respectivos valores, destas forma o conteúdo está pronto a escrever.

A escolha de um dicionário para este contexto tem como principal vantagem a verificação em tempo assimptótico constante da existência de um dado ficheiro na árvore, sem necessidade percorrer esta.

O conteúdo associado a cada ficheiro provém diretamente do estado **FILERD**, que lê os conteúdos. O dicionário permite também a possibilidade da utilização do estado **SKIP**, pois é imediata a verificação da existência de um ficheiro ou não, permitindo assim que este seja ignorado.

3 Implementação em *Flex*

A concretização da teoria deste projecto foi feita em *Flex*, onde foram providenciados todos os mecanismos microscópicos subjacentes à interação entre diferentes condições de contexto.

Com o auxilio do *Flex* conseguimos obter o ponto em que nos encontramos a nível de processamento do *template*, *i.e.* armazenar buffers, identificar títulos, analisar conteúdo de um ficheiro.

Posto isso, utilizamos ficheiros em código *C* referentes ao trabalho realizado por estruturas de dados mais especificas, como a árvore *N*-ária. Permitindo assim uma programação dos pares <ação, reação> de mais alto nível no ficheiro *Flex*.

Para acomodar este tipo de programação, dividimos os ficheiros de código de forma a encapsular o comportamento das estruturas auxiliares utilizadas.

3.1 Estruturas Auxiliares

- ***String***[1]**:** É indispensável o encadeamento e processamento de diferentes entradas textuais, por essa razão decidimos utilizar esta estrutura de dados. Permitindo assim uma maior abstracção no que conta a entradas textuais, que muitas vezes causam um entrave em linguagens como *C*.
- ***Dictionary***[2]**:** Estrutura de dados utilizadas para dois fins. O primeiro é de armazenar os metadados e as respectivas correspondências. Para além disso, é também utilizada para associar os ficheiros aos seus respectivos conteúdo já processados (com os metadados substituídos) permitindo uma escrita mais eficiente.
- ***Stack***[3]**:** O nível de profundidade de uma pasta está associado a uma posição de um *array* de *Stacks*, o que esta configuração permite obter é a pasta mais recentemente aberta, numa dada profundidade, o que permite atribuir ficheiros e sub-pastas nesta.
- ***N-ary Tree***[4]**:** A estrutura de uma árvore de diretorias é necessariamente a de uma àrvore com *N* ramos. Tendo isso em mente, utilizamos a estrutura de dados que mais realisticamente permite emular este comportamento. Isso permite anexar ficheiros e subpastas às respectivas pastas, abaixo do respectivo nó na àrvore.

4 Conclusões e Trabalho Futuro

Em suma, achamos que conseguimos eficientemente consolidar todos diferentes conceitos envolvidos no desenvolvimento da presente ferramenta. Culminando numa versão que através do uso de expressões regulares e condições de contexto, bem como algum código *C*, a descrição e interpretação de um universo de diretorias, ficheiros e os seus conteúdos, contemplando também a configuração de metadados.

Ademais, gostaríamos de impulsionar a nossa ferramenta, tirando mais partido de estruturas como a *Stack*, que permitiria a introdução de um operador de fecho de pasta. Autorizando o pôr conteúdo na penúltima pasta aberta, num dado nível de profundidade. Para além disso, também achamos que seria cativante a admissão de *inline shell commands* permitindo o redireccionamento do *output* de comandos *shell*, diretamente para o conteúdo do ficheiro indicado. Esta tecnologia poderia ser chamada, por exemplo, da seguinte forma:

[1] https://developer.gnome.org/glib/stable/glib-String-Utility-Functions.html
[2] https://developer.gnome.org/glib/stable/glib-Hash-Tables.html
[3] https://developer.gnome.org/glib/stable/glib-Double-ended-Queues.html
[4] https://developer.gnome.org/glib/stable/glib-N-ary-Trees.html

```
=== meta

cp: /etc/passwd

=== tree

copiedFile

=== copiedfile
‘‘cat {%cp%}‘‘
```

Este *template* criaria um ficheiro `copiedFile` com o conteúdo exato do ficheiro origem, passado como metadado no comando *shell* através da variável `cp`.

A título exemplificativo, foram colocadas demonstrações tecnológicas na seguinte página.

A Demonstração Tecnológica

No intuito de atestar as capacidades funcionais da nossa ferramenta, especialmente no espectro de:

- Criação de subdiretorias.
- Composição de metadados.
- Utilidade Prática.

Foram desenvolvidos as seguintes *demos*, anexando o respectivo resultado de cada um dos *inputs*.

A.1 Relatório LaTeX

```
=== meta

proj: PL
email: xico@alunos.uminho.pt
author: Xico Fininho
uni: Universidade do Minho
authrun: R. Reis, D. Ribeiro

=== tree

{%proj%}/
- chapters/
-- intro.tex
-- teoria.tex
-- estruturas.tex
-- flex.tex
-- demo.tex
-- final.tex
- diagrams/
-- .open
- tables/
-- .open
- images/
-- .open
- main.tex

=== main.tex
\documentclass[runningheads]{llncs}

\usepackage[portuguese]{babel}
\usepackage{graphicx}
\usepackage{amsmath}
```

```
\usepackage[utf8]{inputenc}
\usepackage{float}

\begin{document}

\renewcommand{\labelitemi}{$\bullet$}

\title{{%name%}}

\author{
{%author%}
}

\titlerunning{{%uni%}}

\authorrunning{{%authrun%}}

\institute{
Universidade do Minho, Departamento de Informatica\\
e-mail: {%email%}
}

\maketitle

\begin{abstract}

\end{abstract}

\section{Introducao e Contextualizacao}
\input{chapters/intro}

\section{Teoria Essencial}
\input{chapters/teoria}

\section{Estruturas Auxiliares}
\input{chapters/estruturas}

\section{Implementacao em \textit{Flex}}
\input{chapters/flex}

\section{Demonstracao Tecnologica}
\input{chapters/demo}

\section{Conclusoes e Trabalho Futuro}
\input{chapters/final}
```

```
\end{document}

=== intro.tex
=== teoria.tex
=== estruturas.tex
=== flex.tex
=== demo.tex
=== final.tex
=== .open
```

Que dá origem à seguinte árvore de diretorias:

```
PL
├── chapters
│   ├── demo.tex
│   ├── estruturas.tex
│   ├── final.tex
│   ├── flex.tex
│   ├── intro.tex
│   └── teoria.tex
├── diagrams
├── images
├── main.tex
└── tables
```

A.2 *Sub-Directory Conqueror*

```
=== meta

proj: conquerer
sp: sub
fullname: {%sp%}dir-{%proj%}
uni: uminho
dep: informatic
org: kings@{%uni%},{%dep%}

=== tree

{%proj%}/
- {%sp%}0/
-- {%sp%}1/
--- {%sp%}2/
---- {%sp%}3/
----- {%sp%}4/
------ {%org%}/
```

```
──────── {%uni%}1/
──────── {%uni%}2/
───────── {%uni%}3/
────────── {%uni%}4/
─────────── {%uni%}5/
──────────── {%uni%}6/
──────────── .open
- {%sp%}00/
-- {%sp%}11/
--- {%sp%}22/
---- {%sp%}33/
----- {%sp%}44/
{%sp%}/

{%fullname%}/

{%uni%}/

=== .open
```

Que dá origem à seguinte árvore de diretorias:

```
.
├── conquerer
│   ├── sub0
│   │   └── sub1
│   │       └── sub2
│   │           └── sub3
│   │               └── sub4
│   │                   └── kings@uminho,informatic
│   │                       ├── uminho1
│   │                       └── uminho2
│   │                           └── uminho3
│   │                               └── uminho4
│   │                                   └── uminho5
│   │                                       └── uminho6
│   │                                           └── .open
│   └── sub00
│       └── sub11
│           └── sub22
│               └── sub33
│                   └── sub44
├── sub
├── subdir-conquerer
└── uminho
```