Universidade do Minho
Trabalho Prático de Laboratório de Métodos Formais
Relatório de Desenvolvimento

Carlos Daniel Moutinho Machado nº35810

4 de Abril de 2006

Conteúdo

Capítulo 1

Introdução

Este trabalho foi realizado no âmbito da disciplina Laboratório de Métodos Formais. O objectivo do trabalho consiste na análise e implementação de um repositório de cursos de métodos formais [2], tendo por base um protótipo especificado na linguagem `CAMILA`.

Assim, foi necessário a migração do modelo de dados especificado em `CAMILA`, para `VDM++`, e a implementação de funcionalidades para criação, edição e consulta dos dados. Foi também criada uma interface web para permitir a manipulação e visualização remota dos dados. Esta interface foi construida modularmente, separando os componentes de interacção com o utilizador, dos componentes de comunicação com o interpretador de `VDM`, de modo a tornar a interface independente do sistema de gestão dos dados. Assim, futuramente, o interpretador de `VDM` poderá ser facilmente substituido por um Sistema de Gestão de Bases de Dados.

Capítulo 2

Especificação

Como ponto de partida, utilizou-se a especificação formal escrita na linguagem CAMILA.

```
Txt = STR | STR-seq;

FMESOE :: courses:     Courses
          deadcourses: Courses
          urls:        UrlDb
          alltopics:   STR -> Topic
          countries:   STR -> STR-set;

Courses = Course-set;

UrlDb = STR -> Url ;

Course :: ref: STR
          year: STR
          url: STR
          inst: STR
          contact: STR
          topics: STR-set
          langs: STR-set
          tools: STR-set;

Topic :: desc: STR
         keywords: STR-set;

Url :: name: STR
       http: Txt
```

```
       alt: STR
       email: Emailx;

Email :: A: STR
         B: STR;

Emailx = [ Email ];
```

Em seguida, implementou-se este modelo de dados na linguagem `VDM++`. No entanto, optou-se por alterar o tipo de dados `Url`, passando os campos `name` e `email` a serem simplesmente do tipo `STR`.

O modelo de dados em notação `VDM` é então o seguinte:

```
public FMESOE :: courses:     Courses
                 deadcourses: Courses
                 urls:        UrlDb
                 alltopics:   TopicDb
                 countries :  CountrieDb;

public Courses = set of Course;

public UrlDb = map STR to Url ;

public TopicDb = map STR to Topic;

public CountrieDb = map STR to set of STR;

public Course :: ref: STR
                 year: STR
                 url: STR
                 inst: STR
                 contact: STR
                 topics: set of STR
                 langs: set of STR
                 tools: set of STR;

public Topic :: desc: STR
                keywords: set of STR;

public Url :: name: STR
              http: STR
              alt: STR
              email: STR;
```

Capítulo 3

Implementação

A implementação do sistema foi feita de modo bastante modular, de forma a permitir uma divisão entre camadas independentes, as quais podem ser facilmente substituidas.

Apresenta-se de seguida um diagrama que mostra, de modo geral, como estão divididos estes componentes.

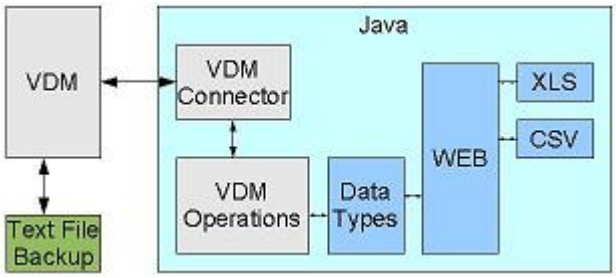

Figura 3.1: Diagrama de componentes.

Os dados são geridos pelas funções/operações que correm no interpretador de `VDM`. Este utiliza um simples ficheiro de texto para armazenamento dos dados. Os componentes em `Java` comunicam com o interpretador, e apresentam os dados ao utilizador numa página web, permitindo ainda a importação/exportação de dados utilizando ficheiros `CSV` *(comma separated values)* ou `XLS (excel spreadsheets)`.

3.1 VDM

Partindo do modelo de dados em `VDM++`, construiram-se as funcionalidades necessárias. Tendo por base as regras de geração de objectos apartir de algebras

[1], criaram-se duas classes, uma delas contendo o tipo de dados, e as funções sobre este, e a outra contendo uma instancia deste tipo, e operações sobre o qual podem ser invocadas.

Todas as operações estão associadas a funções sobre o tipo de dados. Cada operação é definida pela invocação da respectiva função associada tendo como parâmetro a instância sobre a qual se realiza a operação.

Assim, serão apenas descritas as operações existentes.

```
public EMPTY : ()  ==> FMESOE
```

Cria uma instância vazia.

```
public READ_BACKUP : seq1 of char ==> bool
```

Substitui a instância actual, por uma instância prêviamente gravada num ficheiro de texto. Devolve como resultado a indicação do sucesso da operação.

```
public WRITE_BACKUP : seq1 of char ==> bool
```

Guarda a instancia actual num ficheiro de texto. Devolve como resultado a indicação do sucesso da operação.

```
public ADD_COURSE : Course ==> ()
```

Adiciona um curso.

```
public DELETE_COURSE : STR ==> ()
```

Apaga um curso.

```
public UPDATE_COURSE : Course ==> ()
```

Actualiza um curso.

```
public SELECT_COURSES : (Course -> bool) ==> set of Course
```

Selecciona cursos utilizando a função de filtragem recebida como argumento.

```
public GROUP_COURSES :
(set of STR)*(STR -> Course -> bool) ==> map STR to set of Course
```

Agrupa cursos em função de um atributo. Recebe como argumentos a lista de valores possíveis para esse atributo e uma função que relaciona um curso com o atributo.

```
public GROUP_COURSES_BY_INST : () ==> map STR to set of Course
```

Agrupa cursos por instituições.

```
public GROUP_COURSES_BY_CONTACT : () ==> map STR to set of Course
```

Agrupa cursos por contactos.

```
public GROUP_COURSES_BY_TOPIC : () ==> map STR to set of Course
```

Agrupa cursos por tópicos.

```
public GROUP_COURSES_BY_LANG : () ==> map STR to set of Course
```

Agrupa cursos por linguagens.

```
public GROUP_COURSES_BY_TOOL : () ==> map STR to set of Course
```

Agrupa cursos por ferramentas.

```
public ALL_COURSES : () ==> set of Course
```

Devolve todos os cursos.

```
public GET_COURSE_URLS : Course ==> map STR to Url
```

Devolve os urls associados a todas as chaves presentes num curso.

```
public GET_COURSES_URLS : set of Course ==> map STR to Url
```

Devolve os urls associados a todas as chaves presentes num conjunto de cursos.

```
public ADD_URL : STR * Url ==> ()
```

Adiciona um url.

```
public DELETE_URL : STR ==> ()
```

Apaga um url.

```
public UPDATE_URL : STR * Url ==> ()
```

Actualiza um url.

```
public GET_URL : STR ==> Url
```

Devolve um url.

```
public ALL_URLS : () ==> map STR to Url
```

Devolve todos os urls.

```
public SELECT_URLS : (Url -> bool) ==> map STR to Url
```

Selecciona urls utilizando a função de filtragem recebida como argumento.

```
public ADD_TOPIC : STR * Topic ==> ()
```

Adiciona um tópico.

```
public DELETE_TOPIC : STR ==> ()
```

Apaga um tópico.

```
public UPDATE_TOPIC : STR * Topic ==> ()
```

Actualiza um tópico.

```
public ALL_TOPICS : () ==> map STR to Topic
```

Devolve todos os tópicos.

```
public GET_TOPIC_URLS : Topic ==> map STR to Url
```

Devolve os urls associados a todas as chaves presentes num tópico.

```
public ADD_COUNTRY : STR * set of STR ==> ()
```

Adiciona um país.

```
public DELETE_COUNTRY : STR ==> ()
```

Remove um país.

```
public UPDATE_COUNTRY : STR * set of STR ==> ()
```

Actualiza um país.

```
public ADD_INST : STR * STR ==> ()
```

Adiciona uma instituição a um país.

```
public DELETE_INST : STR * STR ==> ()
```

Remove uma instituição de um país.

```
public ALL_COUNTRIES : () ==> map STR to set of STR
```

Devolve todos os países.

```
public GET_COUNTRY_URLS : STR ==> map STR to Url
```

Devolve os urls associados a todas as instituições de um país.
Além destas existem ainda algumas operações auxiliares.

3.2 Java - package fmesoe

Esta package contem todos os componentes em `Java`. A qual se divide ainda em 5 subpackages que serão descritas de seguida.

3.2.1 package fmesoe.datatypes

Esta package contem as classes `Java` que correspondem a um refinamento dos principais tipos de dados existentes na especificação `VDM`. Estas são:

- classe `Course`
- classe `Url`
- classe `Topic`
- classe `Country`

3.2.2 package fmesoe.VDM

Esta é a package responsável pela interacção com o interpretador de `VDM`. Para tal utiliza a biblioteca `ToolboxAPI.jar` [4] incluída nas `VDMTools`.

VDMConnector

Esta é a classe responsavel por estabelecer ligação com o interpretador de VDM e inicializa-lo. Note-se que nesta classe existem 4 parâmetros que devem ser alterados de acordo com a configuração do sistema. Estes são:

- fmesoe_vpp_file : Localização do ficheiro que contem as classes `VDM`
- io_lib_file : Localização da biblioteca IO das `VDMTools`.
- vppref_ior_file : Localização do ficheiro `vppref.ior` utilizado para comunicar com o interpretador.
- backup_file : Localização do ficheiro de backup utilizado pelas classes `VDM` para guardar os dados.

Note-se ainda que esta classe pode ser reutilizada em outras aplicações que necessitem de estabelecer comunicação com as `VDMTools`.

VDMOperations

Esta é a classe responsável por invocar as operações no interpretador de `VDM`. Todas as operações disponíveis têm um método nesta classe que lhes corresponde.

Para realizar operações de envio (resp. consulta) de dados, é necessário invocar os métodos de abstracção (resp. representação) entre os tipos de dados em `Java` e os tipos de dados em `VDM`. Para tal foram criadas duas classes responsáveis por tais tarefas.

VDM2Java

Esta classe implementa as "funções de representação"entre os tipos de dados em *VDM* e os tipos de dados em **Java**. Assim estão disponíveis as seguintes representações:

- *seq of char* para **String**
- *set* para **"Array"**
- *"record"* para **"Array"**
- *set of seq of char* para **String[]**
- *map* para **HashMap**
- *Course* para **Course**
- *Url* para **Url**

- *Topic* para **Topic**
- *Country* para **Country**

Java2VDM

Esta classe implementa as "funções de abstracção"entre os tipos de dados em **Java** e os tipos de dados em *VDM*. Assim estão disponíveis as seguintes abstracções:

- **String** para *seq of char*
- **"Array"** para *seq*
- **String**[] para *set of seq of char*
- **"Array"** para *"tuple"*
- **Course** para *"tuple"*
- **Course**[] para *set of "tuple"*
- **Url** para *"tuple"*
- **Url**[] para *set of "tuple"*
- **Topic** para *"tuple"*
- **Topic**[] para *set of "tuple"*

Note-se que na `API` de comunicação entre `VDM` e `Java` não existem *records*, logo é necessária a conversão dos tipos de dados para simples *tuples*, e a posterior invocação das funções auxiliares criadas em `VDM` para a criação do respectivo tipo de dados.

PreConditionFailedException e PostConditionFailedException

O mecanismo de pré e pos-condições disponíveis em `VDM` é aqui representado através do mecanismo de excepções em `Java`.

3.2.3 package fmesoe.web

A interface web foi construida utilizando `Java Servlets`. Esta package contem os `Servlets` responsaveis por gerar dinamicamente as páginas para visualização e manipulação de dados.

Para permitir o upload de ficheiros para o servidor, esta package faz uso das bibliotecas `Commons IO` e `Commons FileUpload` disponibilizadas pelo projecto open source `The Apache Jakarta Project` [3].

MainServlet

Este é o `Servlet` que constroi a página inicial. Esta fornece o acesso a todas as funcionalidades que estão implementadas nos restantes `Servlets` :

- Visualização e pesquisa de cursos
- Agrupamento de cursos por um dado atributo
- Exportação de cursos para CSV e XLS
- Importação de cursos de CSV e XLS
- Visualização e pesquisa de urls
- Exportação de urls para CSV e XLS
- Importação de urls de CSV e XLS
- Visualização de tópicos
- Exportação de tópicos para CSV e XLS
- Importação de tópicos de CSV e XLS
- Visualização de países
- Exportação de países para CSV e XLS
- Importação de países de CSV e XLS

GenerateHTML

Esta é a classe responsavel pela geração de `HTML` apartir dos vários tipos de dados existentes. Note-se que para gerar o código `HTML` associado a um tipo de dados, é preciso fornecer ao respectivo método, um `HashMap` contendo a associação entre todas as chaves de url presentes no tipo de dados, e os correspondentes Urls.

3.2.4 package fmesoe.CSV

Esta package contêm as funcionalidades responsáveis pela importação e exportação de dados através de CSV *(comma separated values)*.

3.2.5 package fmesoe.XLS

Esta package contêm as funcionalidades responsáveis pela importação e exportação de dados através de ficheiros XLS *(excel spreadsheets)*. Para tal, faz uso da biblioteca `POI` disponibilizada pelo projecto open source `The Apache Jakarta Project` [3].

Capítulo 4

Conclusões / Trabalho Futuro

O VDM fornece um mecanismo de verificação de invariantes e pre/pós-condições. Estas foram utilizadas com frequência ao longo das várias funcionalidades especificadas. No interpretador é possivel activar/desactivar a opção de verificação dinâmica destas condições. No entanto, visto que se está a utilizar o interpretador através de uma API, e não directamente pelo utilizador, não queremos que este interrompa a sua execução sempre que necessita de avisar sobre a falha de uma destas condições. Visto que a `ToolBoxAPI` não disponibiliza uma forma de comunicar estes "avisos", optou-se por declarar explicitamente operações em `VDM` correspondentes às verificações das pré e pós-condições, invocando-as sempre que necessário, sendo depois mapeadas no mecanismo de excepções em `Java`.

Uma grande parte do projecto, consistiu na investigação de uma interface adequada à inserção/actualização de dados. Os habituais formulários web para inserção de dados tornariam ineficiente e pouco prática a criação/actualização de vários cursos em simultâneo. Por outro lado, as folhas de cálculo são um mecanismo bastante prático tanto para visualização, como edição de dados. Assim, procurou-se encontrar uma ferramenta que permitisse a visualização de dados em forma de folha de cálculo directamente no *browser*. Infelizmente, nenhuma das ferramentas encontradas permitia a manipulação dos dados, sendo apenas possível a sua consulta. Optou-se portanto por permitir a visualização dos dados em `HTML` e a opção de os importar/exportar para formato *excel* (e tambem CSV). Com a evolução da tecnologia `AJAX`, este problema será em breve ultrapassado e poderá ser um melhoramento a desenvolver.

Este trabalho permitiu a aplicação de temas abordados nas disciplinas de métodos formais, tais como a especificação formal, refinamento de dados, etc.

No entanto, visto o interpretador de `VDM` ser uma ferramenta adequada para a animação de especificações e não como um sistema de gestão de dados, este

torna o sistema bastante ineficiente. Por exemplo, as pesquisas de cursos (os quais existem cerca de 180) é uma operação que pode demorar alguns minutos.

Assim, o objectivo final, e sobre o qual foi desenhado todo o sistema, é a substituição do interpretador, por um sistema gestor de base de dados, tal como se mostra na seguinte figura.

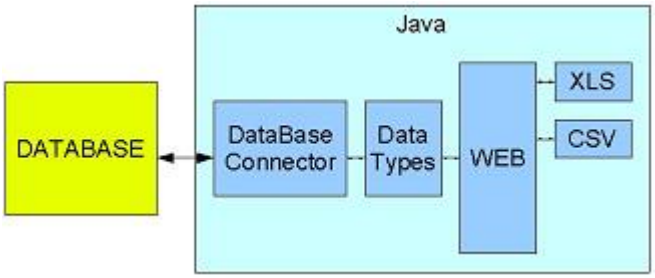

Figura 4.1: Sistema futuro.

A divisão do sistema em componentes independentes, permitirá esta actualização com grande facilidade.

Um dos problemas não abordados, é a questão da concorrência. No sistema final, este problema está a cargo do sistema gestor de bases de dados utilizado. Na aplicação desenvolvida, o ficheiro que guarda a informação é lido e escrito sempre que necessário, nada fica guardado em memória, de modo a prevenir que um utilizador esteja a manipular informação desactualizada. Embora as operações de escrita ocorram com menos frequência que as operações de leitura, nada garante a consistência e preservação da informação actualizada em caso de um acesso simultâneo. Isto pode ser garantido, utilizando algoritmos de gestão de *"locks"* (por ex: *"two phase lock"*). Vários utilizadores podem solicitar um *"lock"* de leitura, mas um *"lock"* de escrita impedirá a atribuição de novos *"locks"*.

Capítulo 5

Apêndice

Dentro da directoria principal da aplicação, encontram-se as seguintes pastas e ficheiros:

- \lib - contêm as bibliotecas externas utilizadas
- \src - contêm todo o código fonte `Java`
- \vdm - contêm o código `VDM` e o ficheiro de backup da informação
- \web - contêm informação necessária à configuração dos `Servlets`
- build.xml - ficheiro utilizado pelo `Ant` para fazer o `deploy` da aplicação
- build.properties - ficheiro auxiliar do anterior
- INSTALL NOTES.txt - contêm a informação de instalação da aplicação

Bibliografia

[1] A. Cruz, L. Barbosa, and J.N. Oliveira. From algebras to objects: Generation and composition. *j-jucs*, 11(10):1580–1612, 2005.

[2] FME Subgroup on Education (convenor: J.N. Oliveira). A survey of formal methods courses in european higher education. In C. Neville Dean and Raymond T. Boute, editors, *Proc. CoLogNet/Formal Methods Europe Symposium on Teaching Formal Methods 2004http://www.intec.rug.ac.be/ roupsites/formal/Sympos2004/Sympos2004.htm, Ghent University, Belgium, Nov 18-19, 2004*, number 3294 in Lecture Notes in Computer Science, pages 235–248. Springer-Verlag, 2004. Web version (HTML) available from `http://www.fmeurope.org/` ↦ `Formal Methods` ↦ `Education`.

[3] The apache jakarta project - http://jakarta.apache.org.

[4] Vdm tools - toolbox api.