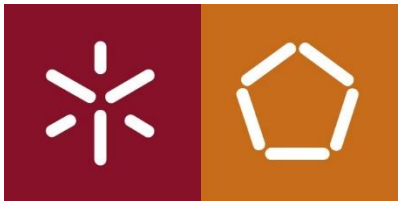
Gramáticas na Compreensão de Software

Mestrado Integrado em Engenharia Informática

Universidade do Minho



Trabalho Prático - SelRA

Grupo nº 4

Filipa Alves dos Santos (A83631)

José Ricardo Sousa Mendes da Cunha (A84302)

14 de janeiro de 2021

Índice de conteúdos

1. Introdução

Como trabalho prático da unidade curricular de Gramáticas na Compreensão de Software, foi pedido aos alunos que desenvolvessem uma DSL (*Domain-Specific Language*), de nome SelRA, que servisse de auxílio a um professor que estivesse a escolher quais os **recursos de aprendizagem** (RA) mais adequados para um determinado **aluno** (Al), que tem as suas características psicológicas específicas, para lhe ensinar um determinado **conceito de programação** (CProg).

Assim, a SelRA funciona como um algoritmo que recebe um certo aluno e um conceito de programação e, através de uma relação lógica, devolve os 3 recursos de aprendizagem mais apropriados para o específico cenário em questão.

Neste relatório, será primeiro abordado a pesquisa / escolha feita relativamente às características que se considerou em relação aos alunos, quais os recursos de aprendizagem disponibilizados e os CProg considerados. De seguida, uma explicação mais pormenorizada sobre a ontologia elaborada, o código em si e sobre a gramática utilizados na concretização desta DSL. Por fim, será apresentado a parte dos testes, através de um inquérito em Google Forms, como pedido no enunciado.

2. Desenvolvimento

2.1. Pesquisa

Para começar este trabalho prático, foi necessário determinar quais os dados que se iriam utilizar, em termos de características dos alunos, recursos e conceitos de programação. Para além disso, também foi necessário estabelecer a relação entre estes para que o algoritmo conseguisse determinar os recursos ideais.

2.1.1. Alunos

Relativamente às características dos alunos, era pedido que, no mínimo, tivesse associado algum tipo de identificação do aluno, a sua idade e algumas características emocionais / sociais que interferem na aprendizagem.

Assim, cada aluno é representado neste formato:

AL Id "Nome" (Caracteristica1, Caracteristica2, ...) Idade

Após alguma pesquisa e conclusões tiradas por experiência própria do grupo no que toca ao ensino, foram disponibilizadas 10 distintas características para serem atribuídas aos alunos: tímido, extrovertido, hiperativo, emotivo, desconcentrado, desorganizado, falador, observador, curioso e energético.

2.1.2. Conceitos de Programação

Já os conceitos de programação são apresentados como:

CProg "Nome"

O grupo decidiu abordar conceitos mais gerais, servindo-se de exemplo se algumas áreas de informática estudadas ao longo do curso. Deste modo, foram considerados os seguintes temas de programação: Aprendizagem Automática, Programação Funcional, Computação Gráfica, Cálculo de Programas, Algoritmos, Desenvolvimento de Aplicações Web, Programação Imperativa, Programação Orientada a Objetos, Base de Dados, Gramáticas na Compreensão de Software, Programação Orientada a Objetos, Bases de Dados NoSQL e Computação em R.

2.1.3. Recursos de Aprendizagem

Por último, cada recurso de aprendizagem tem um id e descrição, características psicológicas associadas e a idade mais adequada para cada um deles. Além disso, também tem associado todos os CProg que disponibilizam tal recurso:

RA Id "Descrição" (Caracteristica1, Caracteristica2, ...) Idade
CProg1
CProg2

Em termos de recursos, optou-se pelos mais imediatos quando se pensa em instrumentos de aprendizagem para os conceitos de programação considerados: livros, vídeos, demonstrações práticas, palestras, projetos de grupo, jogos / aplicações educacionais e avaliações individuais.

No caso dos livros e vídeos, pode-se destacar alguns em específico como o livro "Livro: JAVA 8 - POO e Construções Funcionais" ou o vídeo "Video: What is Database & SQL?". Todos os recursos disponíveis estão disponíveis do ficheiro "**teste.txt**".

Também no mesmo ficheiro, temos todos os alunos, cujos nomes foram gerados utilizando o site https://gerador-nomes.herokuapp.com/ e as suas características psicológicas e idade foram atribuídas de modo arbitrário.

2.2. DSL

Nesta secção, irá ser explicado com mais detalhe todo o processo de como desenvolvemos a nossa DSL, nomeadamente a ontologia, o algoritmo usado e alguns pormenores importantes sobre o Lexer e Parser.

2.2.1. Ontologia

Uma ontologia é uma especificação explícita de uma conceptualização:

• **Explícita**: no sentido em que as entidades da ontologia são/estão claramente definidas, distintas e inter-relacionadas entre si.

• **Conceptualização**: no sentido em que representa um modelo abstrato e cognitivo de um domínio de conhecimento, identificando os conceitos e as características desse mesmo domínio.

A ontologia realizada pelo grupo é a seguinte:

```
conceitos{Recurso,Conceito,Aluno,Caracteristica}
individuos{}
relacoes{é explicado por, }
triplos{
        conceito = é explicado por => recurso
        aluno = tem => caracteristica
        recurso = tem => caracteristica
}
```

2.2.2. Algoritmo

Relativamente ao algoritmo, optou-se pela seguinte fórmula:

```
int nota = (int)(((entry.getValue().size()*100)/aluno.getCaracteristicas().size()) * 0.7 + 0.3 *
            (aluno.diferencaGeracao(entry.getKey().getIdade()) * 0.9 +
             (aluno.diferencaIdade(entry.getKey().getIdade())) * 0.1));
```

Como se pode observar, as características psicológicas do aluno influenciam 70% da sua "nota" enquanto os restantes 30% são atribuídos à idade do aluno. Sendo que não se intendia que pequenas diferenças da idade ideal para o recurso influenciassem gravemente o resultado, deu-se mais relevância à diferença geracional.

Assim, este algoritmo vai primeiro ver quantas características a aluno tem em comum com o recurso em questão e a diferença da idade do aluno relativamente à ideal e determina um número final que indica se o recurso vai ser recomendado ou não.

2.2.3. Lexer

O **Lexer** é a parte da gramática responsável por receber texto como input e devolver os respetivos tokens como output. Na seguinte imagem, temos a definição de todos os tokens utilizados:

```
RA : [Rr][Aa]
   ;

AL : [Aa][Ll]
   ;

CPROG : [Cc][Pp][Rr][Oo][Gg]
      ;

ALUNO : [Aa|Pp][0-9]+
       ;

PAL : [a-zA-Z0-9]+
        ;
```

```
QUOTE : '"' ~('\r' | '\n' | '"')* '"';
COM :  '/*' ~([*/])* '*/' -> skip;

VIRGULA: ',';
PONTO: '.';
PONTOVIRGULA: ';';
LPARENT: '(';
RPARENT: ')';
ASPAS : '"';

WS: ('\r'? '\n' | ' ' | '\t')+ ->skip;
```

Para além dos típicos tokens (como o WS, ASPAS, RPARENT, …, PAL), foram criados tokens adicionais para representar os recursos (RA), os alunos (AL) e os conceitos de programação (CPROG).

2.2.4. Parser e Código Java

O **Parser** é responsável por receber e reconhecer os tokens vindos do Lexer, e devolver como output a árvore de derivação correspondente. Foram feitas as seguintes produções:

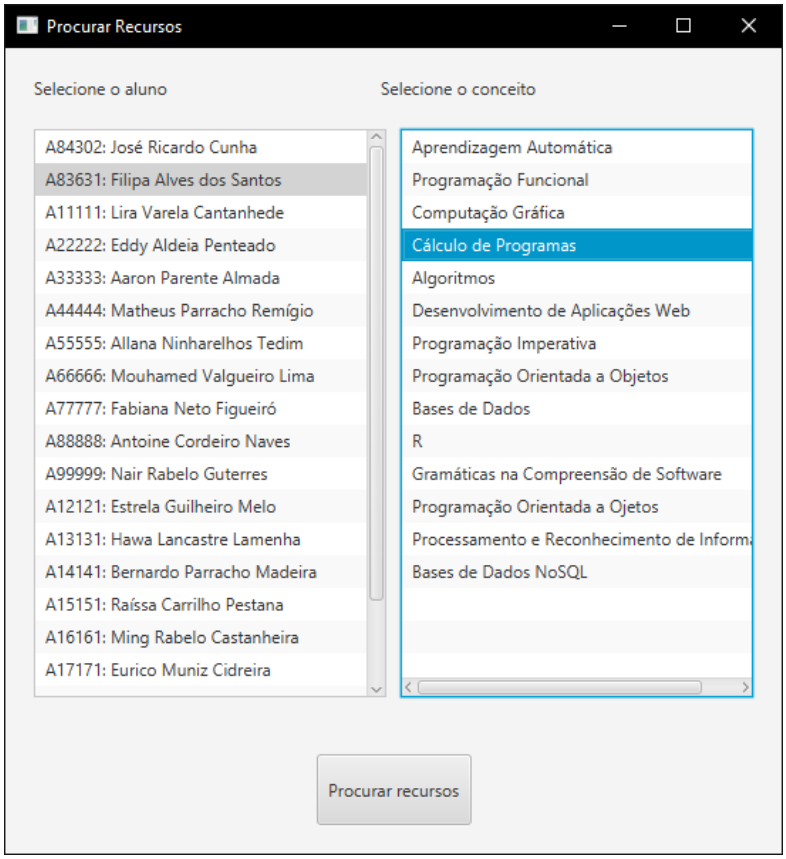
- **tr:** produção principal
- **ra** e **recurso:** produções dos recursos (sendo a primeira mais geral)
- **al:** produção dos alunos
- **cprog:** produção dos conceitos de programação
- **desc:** produção relativa às características dos alunos e recursos
- **quote:** produção relativa aos nomes dos recursos, conceitos e alunos

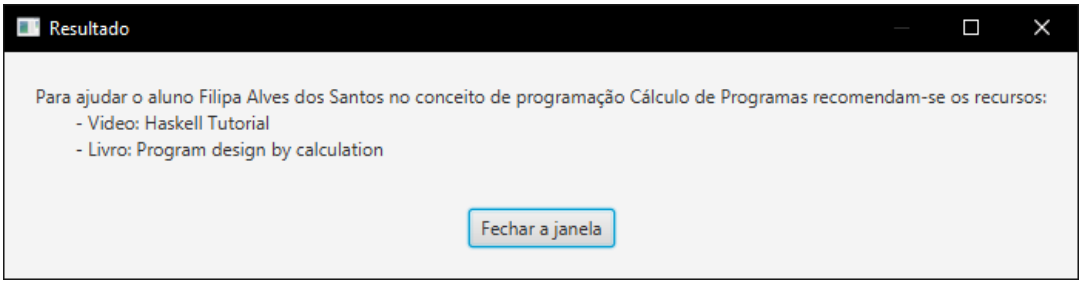
Como auxílio à gramática, utilizou-se **código em Java** para realizar algumas das ações relativas ao nosso programa. Temos 3 diferentes classes: Recurso, Aluno e Conceitos – cada uma com os dados descritos na secção 2.1 deste relatório. Em cada uma destas classes temos funções que efetuam simples operações sendo as únicas merecedoras de explicação as seguintes:

- **diferencaGeracao** e **diferencaIdade:** funções responsáveis pelo cálculo do score de compatibilidade do recurso e do aluno em questão, em termos de idade.

- **addRecurso e getRecursos**: a primeira função coloca numa hash um recurso e as suas respetivas características e a segunda é a função principal, que devolve a lista de recursos apropriados para um certo aluno, através do algoritmo explicado na secção.

Também através de librarias do Java, foi desenvolvida a interface seguinte para facilitar a experiência dos utilizadores. Para a executar, é apenas preciso correr o ficheiro **SeIRA.jar** na mesma localização que o ficheiro de testes a ser utilizado (neste caso, teste.txt).

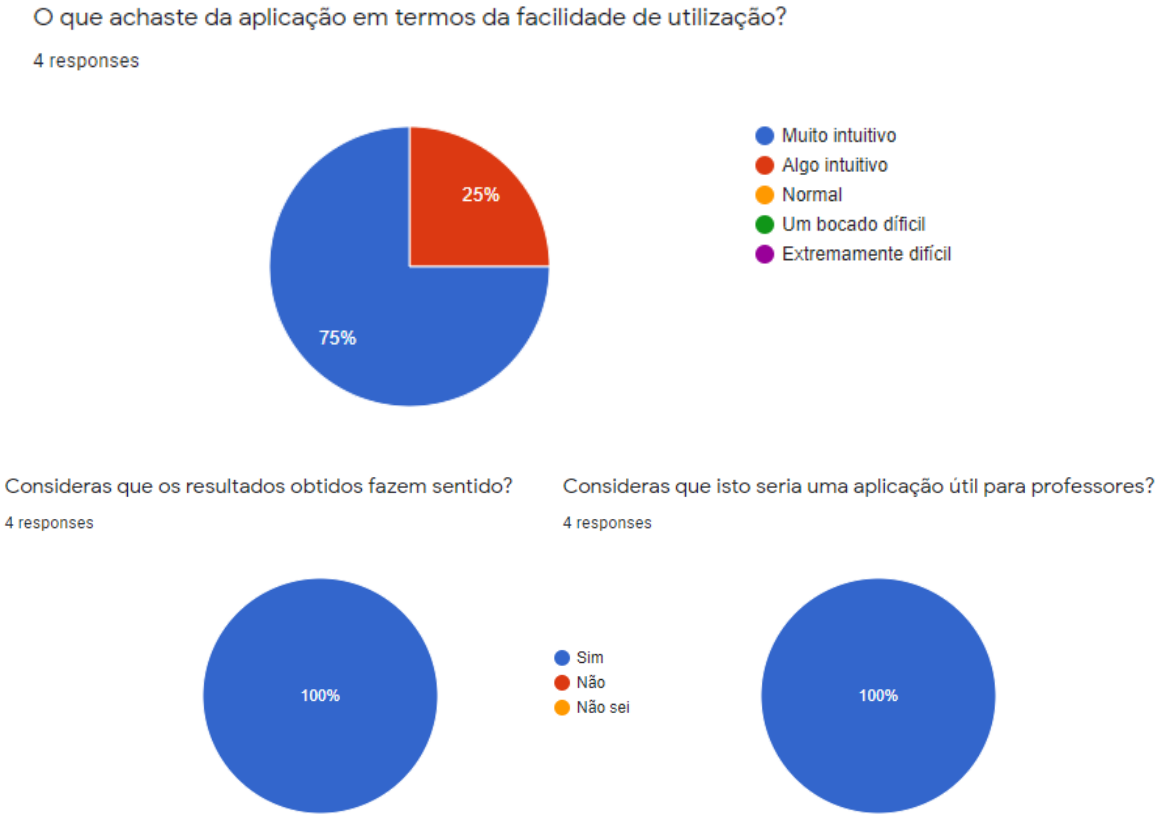




2.4. Testes

Para testar e validar o sistema desenvolvido, foi pedido a 4 pessoas que testassem a aplicação e preenchessem um rápido formulário (**Google Forms**) sobre a mesma. Foi questionado sobre a **facilidade de utilização**, **se os resultados obtidos fizeram sentido**, **se seria uma aplicação útil** para os professores e, por último, **comentários extra** sobre o projeto.

Estes foram os resultados obtidos:



Comentários extra:

2 responses

Uma aplicação do género seria muito útil no contexto universitário sendo que muitas vezes o método de ensino utilizado nem sempre é o mais apropriado

Se este programa fosse efetivamente utilizado no futuro, era interessante dar mais opções ao professor como atribuir um peso maior a certas características do aluno ou algo do género

3. Conclusão

Este trabalho prático deu-nos uma melhor compreensão do processo que é construir uma DSL (*Domain-Specific Language*) desde a sua origem, com liberdade de escolha relativamente à interpretação feita e algoritmo considerado.

O grupo encontra-se satisfeito com resultado final, sendo que cumpriu todos os objetivos que tinha definido para este projeto: o algoritmo parece coeso e fazer sentido no contexto do objetivo definido no enunciado, foram fornecidos vários dados para se testar a aplicação em si e a gramática utilizada é de fácil leitura e eficaz.

Em suma, este projeto prático foi essencial para consolidar a matéria dada na sala de aula, praticar a escrita de gramáticas e o uso de ferramentas úteis como o ANTLR4.