DSL de Figuras – Documentação Técnica

Objetivo

A linguagem específica de domínio (DSL) foi criada para permitir que a Shodrone especifique figuras aéreas a um nível elevado, **desacoplando a definição de figuras da implementação dos drones**. A Shodrone requer uma linguagem neutra de alto nível para a descrição de figuras, evitando a dependência de fornecedores. Cada *figura* num espetáculo é definida como uma composição de formas geométricas ou bitmaps 3D (não ambos em simultâneo), com movimentos associados. A DSL capta estes conceitos de domínio (por exemplo, círculo, linha, rotação, translação, tipo de drone, posição), para que os designers possam criar novas formações facilmente. O input da DSL (de uma ferramenta externa) é então validado e compilado em scripts de voo por drone, os quais são usados na simulação.

Pressupostos e Decisões de Design

- A DSL é uma **linguagem textual externa** (não embutida).
- Os templates podem conter formas geométricas ou bitmaps, mas nunca os dois em simultâneo.
- Suporte para formas e movimentos (ex. rotate, translate) com sintaxe específica.
- Os tipos de drones são definidos no domínio e referenciados nas declarações de figuras.
- Utilização do ANTLR4 para gerar o parser e o visitor.
- Inclusão de palavras-chave case-insensitive, sintaxe de coordenadas e literais numéricos.

Gramática DSL (ANTLR)

```
grammar dsl;

dsl
  : version
    drone_model
    variable_declaration*
    element_definition*
    block_statement*
    unscoped_statement*
  ;

version
  : 'DSL' 'version' VERSION_NUMBER ';'
  ;

drone_model
  : 'DroneType' ID ';'
  ;

variable_declaration
  : position_declaration
  | velocity_declaration
  | distance_declaration
  ;

position_declaration
  : 'Position' ID '=' vector ';'
  ;

velocity_declaration
  : 'Velocity' ID '=' expression ';'
  ;

distance_declaration
  : 'Distance' ID '=' expression ';'
  ;

vector
  : '(' expression ',' expression ',' expression ')'
  ;

element_definition
  : ID ID '(' parameter_list? ')' ';'
```

```
  ;

parameter_list
  : parameter (',' parameter)*
  ;

parameter
  : ID
  | vector
  | expression
  ;

block_statement
  : block_type statement+ end_block_type
  ;

unscoped_statement
  : statement
  | pause_statement
  ;

block_type
  : 'before'
  | 'after'
  | 'group'
  ;

end_block_type
  : 'endbefore'
  | 'endafter'
  | 'endgroup'
  ;

statement
  : ID '.' method ';'
  | group_statement_block
  ;

group_statement_block
  : 'group' statement+ 'endgroup'
  ;

method
  : 'move' '(' vector ',' expression ',' ID ')'
```

```
  | 'rotate' '(' rotate_param ',' rotate_param ',' rotate_param ',' rotate_param ')'
  | 'lightsOn' '(' parameter_list ')'
  | 'lightsOff()'
  ;

rotate_param
  : vector
  | expression
  ;

pause_statement
  : 'pause' '(' expression ')' ';'
  ;

expression
  : NUMBER
  | ID
  | expression ('*' | '/' | '+' | '-') expression
  | '(' expression ')'
  ;

VERSION_NUMBER : [0-9]+ '.' [0-9]+ '.' [0-9]+ ;

ID             : [a-zA-Z_][a-zA-Z0-9_]* ;

NUMBER         : '-'? [0-9]+ ('.' [0-9]+)? ;

WS             : [
]+ -> skip ;
```

Componentes da Validação DSL

A arquitetura da validação DSL é modular e extensível, baseada no padrão Visitor do ANTLR e separação de responsabilidades por tipo de validação semântica. Todos os validadores se encontram no pacote `validator`.

Arquitetura Geral

- **ANTLR Grammar** (`dsl.g4`): Define toda a sintaxe da linguagem (palavras-chave, expressões, estruturas de blocos).
- **ANTLR Parser & Lexer**: Gera os ficheiros `dslParser` e `dslLexer` a partir da gramática.
- **`FigureValidationPlugin`**:
  Plugin principal que coordena todo o processo de validação. Responsável por:
  - Guardar o estado da validação (variáveis, elementos, erros).

- Iniciar o parser e aplicar os visitantes.
- Armazenar os erros semânticos e sintáticos encontrados.
- Resetar estado entre validações.

Visitantes / Validadores

Cada visitante é responsável por validar uma parte específica do DSL. Todos herdam de `dslBaseVisitor`.

DslVersionValidator

- Garante que a primeira linha contém a versão no formato `DSL version X.Y.Z`.
- Verifica se o formato do número de versão é válido.
- Armazena a versão no plugin para uso posterior.

DroneModelValidator

- Verifica se existe exatamente **uma** declaração `DroneType`.
- Reporta erro se houver mais de uma declaração ou se estiver ausente.
- Armazena o nome do modelo de drone no plugin.

DeclaredVariablesValidator

- Valida declarações de:
  - `Position`: Vetores de 3 componentes (x, y, z).
  - `Velocity` e `Distance`: Expressões numéricas válidas.
- Garante que variáveis duplicadas não são permitidas.
- Verifica que expressões são válidas (valores numéricos ou variáveis declaradas anteriormente).
- Armazena variáveis em mapas categorizados por tipo.

ElementDefinitionValidator

- Valida a criação de elementos (`Line`, `Rectangle`, etc.).
- Garante que os parâmetros estão corretos:
  - O primeiro é uma posição (literal ou variável).
  - Os seguintes são numéricos (ou variáveis numéricas).
  - O último é o nome do drone declarado.
- Verifica que o drone usado corresponde ao `DroneType` declarado.
- Adiciona o nome do elemento à lista de elementos declarados.

StatementBlockValidator

- Valida blocos `before`, `after`, `group`.
- Garante que blocos estão corretamente abertos e fechados (`before` → `endbefore`, etc.).
- Valida regras de aninhamento:
  - Um bloco `before` não pode aparecer depois de um `after`.
  - Blocos `group` só podem existir dentro de `before`, `after`, ou no nível global.

- Valida invocações de métodos em elementos declarados.
    - `move()`, `rotate()`, `lightsOn()`, `lightsOff()`:
        - Valida os parâmetros e tipos de dados esperados.
        - Verifica se variáveis passadas como argumentos existem e são do tipo correto.

UnscopedStatementValidator

- Valida declarações fora de blocos (`unscoped`), como:
    - `pause(10);`
    - `element.lightsOn(255, 255, 255);`
- Garante que apenas métodos válidos estão a ser usados fora de blocos.
- Verifica se o elemento usado foi previamente declarado.
- Valida o número e tipo de parâmetros (e.g., `pause` espera número/variável numérica).

Estrutura de Dados no Plugin

`FigureValidationPlugin` contém:

- `Map<String, Map<String, String>> declaredVariables`

  Variáveis agrupadas por tipo: `Position`, `Velocity`, `Distance`, `DroneType`.

- `List<String> declaredElements`

  Nomes dos elementos (figuras) declarados no script.

- `List<String> unscopedStatements`

  Lista de instruções fora de blocos (`pause`, `lightsOn`, etc.).

- `List<String> errors`

  Lista completa de erros semânticos e sintáticos acumulados durante a validação.

- `String droneModelName`

  Nome do drone declarado com `DroneType`.

- `String dslVersion`

  Versão especificada na primeira linha.

Exemplos de Scripts

Válido

```
DSL version 1.1.1;
DroneType Mavic3Classic;

Position aPos = (0,0,0);
Position anotherPos = (0, 10, 0);
Position zAxis = (0,0,1);

Velocity aVelocity = 5.1;
Velocity rotVelocity = 3.14/10;

Distance aLenght = 20;

Line aLine(aPos, aLenght, Mavic3Classic);
Rectangle aRectangle(anotherPos, aLenght, aLenght, Mavic3Classic);

before
    aLine.lightsOn(255,255,0);
    aLine.move((0, 0, 1),30, aVelocity);
    aRectangle.lightsOn(0,125,125);
    aRectangle.move((0, 0, 1),40, aVelocity);
endbefore

group
    aLine.rotate(aPos, zAxis, 2*3.14, rotVelocity);
    aRectangle.rotate(aPos, zAxis, -2*3.14, rotVelocity);
endgroup

pause(10);

aLine.lightsOn(0,1,2);
aRectangle.lightsOn(255,0,0);

after
    aLine.move((0, 0, -1),30, aVelocity);
    aRectangle.move((0, 0, -1),40, aVelocity);
    aLine.lightsOff();
    aRectangle.lightsOff();
endafter
```

Inválido

```
DSL version 1.1.1;
DroneType Mavic3Classic;

Position aPos = 12;
Position anotherPos = (0, 10, 0);
Position zAxis = (0,0,1);

Velocity aVelocity = (1,1,2);
Velocity rotVelocity = 3.14/10;

Distance aLenght = 20;

Line aLine(aPos, aLenght, Mavic3Classic);
Rectangle aRectangle(anotherPos, aLenght, aLenght, Mavic3Classic);
```

> *`Position` mal definida como escalar em vez de vetor.*
> *`Velocity` mal definida como vetor. Erros semânticos e sintáticos!*

Conclusão

A DSL permite um controlo expressivo e garante correção técnica via parsing e validação. Simplifica a lógica complexa da simulação em templates legíveis e validados, seguindo boas práticas de DSL e integração modular com a infraestrutura da Shodrone.