

Relatório LPROG

Sprint 3 – Grupo 13

Maria Francisca Branco, 1230540

Maria Helena Pinho, 1220849

Afonso Sousa, 1230978

Ricardo Keng, 1231500

Davide Freitas, 1221363

Professor Alberto Sampaio (ACS)

User Stories:

US340 - DSL Plugin

“As a Drone Tech, I want to deploy and configure a plugin to be used by the system to analyse the figure high-level description”

US341 - Validate figure description

“As a Show Designer, I want to validate the syntax of the figure description (DSL), so that I can register the figure in the system”

US344 - Generation of a drone program

As a Drone Tech, I want the system to use the figure/show high-level description code to generate the code of the drones to be used in the simulation/test

US345 - Drone language Plugin

As a Drone Tech, I want to deploy and configure a plugin to be used by the system to analyse/validate a drone program. There must be a plugin for each different drone language.

US346 - Validation of a drone program

As a Drone Tech, I want to validate the syntax of the code for a specific drone in a given show/figure, so that I can later test the figure/show.

US347 - Proposal generation

As CRM Manager, I want the system to generate and to validate the proposal template and to generate the proposal document for the show proposal.

US348 - Show generation

As a Drone Tech, I want the system to generate the show high-level description from its set of figures, so that I can later generate the actual code for each drone.

O que fizemos?

Para este sprint final, procedemos à divisão das tarefas com base na organização definida pela unidade curricular de EAPLI. A distribuição inicial das user stories foi a seguinte:

- US340, foi atribuída à Maria Helena

- US341, foi atribuída à Maria Francisca

- US344, foi atribuída ao Afonso

- US345, foi atribuída inicialmente ao Rodrigo; no entanto, por este não estar inscrito na unidade curricular, a tarefa foi atribuída à Maria Francisca

- US346, foi atribuída ao Ricardo

As últimas duas user stories permaneceram por distribuir nessa fase inicial.

Já durante o sprint, fomos informados de que a **US344** não deveria ser desenvolvida. Assim, as user stories restantes foram distribuídas da seguinte forma:

- US347, foi atribuída ao Davide; e com participação e auxílio do Afonso

- US348, foi atribuída ao Davide; e com participação e auxílio do Afonso

De forma resumida, a **US345 e US346** foi implementada da seguinte maneira:

1) Classe **Drone**

O método *droneOnePlugin* recebe como argumento uma string *path*, que indica o caminho para o ficheiro que contém o código da linguagem específica associada ao drone.

A primeira tarefa do método é realizar a validação sintática do ficheiro, utilizando o método *validate(path)* da classe **DroneSyntaxValidator**.

* Caso a validação seja bem-sucedida, o método procede ao parsing do ficheiro, invocando *lexerAndParser(path)* e retorna true.
* Caso a validação falhe, é exibida uma mensagem de erro e o método retorna false.

public static boolean droneOnePlugin(String path) {  
 System.*out*.println("Drone One:");  
 if (drones.validations.DroneSyntaxValidator.*validate*(path)) {  
 *lexerAndParser*(path);  
 return true;  
 } else {  
 System.*out*.println("Validation failed. Not parsing.");  
 return false;  
 }  
 }

Da mesma forma, existe um método muito idêntico, só que é invocado para a linguagem de drone Two.

public static boolean droneTwoPlugin (String path) {  
 System.*out*.println("Drone Two: ");  
 if (drones.validations.DroneSyntaxValidator.*validate*(path)) {  
 *lexerAndParser*(path);  
 return true;  
 } else {  
 System.*out*.println("Validation failed. Not parsing.");  
 return false;  
 }  
}

- Método *lexerAndParser*

Este método recebe o caminho do ficheiro e é responsável pela leitura do conteúdo, construção do lexer, do parser e da árvore sintática.

Após a criação da árvore, o método invoca o visitante (*parseWithVisitor*) para visitar e interpretar a árvore.

public static void lexerAndParser(String path){  
 try {  
 String content = new String(Files.*readAllBytes*(Paths.*get*(path)));  
 DronesLexer lexer = new DronesLexer(CharStreams.*fromString*(content));  
 CommonTokenStream tokens = new CommonTokenStream(lexer);  
 DronesParser parser = new DronesParser(tokens);  
 ParseTree tree = parser.program();  
 *parseWithVisitor*(tree);  
 } catch (IOException e) {  
 System.*err*.println("Error reading file: " + e.getMessage());  
 }  
 }

- Método *parseWithVisitor*

Este método recebe a árvore sintática gerada e cria um visitor. Invoca a classe **DroneInterpreterVisitor**, que vai percorrer a árvore e executar as ações necessárias para interpretar o código.

public static void parseWithVisitor(ParseTree tree) {  
 DroneInterpreterVisitor visitor = new DroneInterpreterVisitor();  
 visitor.visit(tree);  
 System.*out*.println("Visit ended (visitor)");  
 }

2) Classe **DroneInterpreterVisitor**

Nesta classe é implementada a lógica para interpretar cada parte da árvore sintática.

- Com o método *visitProgram*: visita o nó program

Imprime uma mensagem de início e prossegue a visitar aos nós filhos

- Com o método *visitInfoInstDroneOne* ou *visitInfoInstDroneTwo*:

Extrai informações e imprime-as (a instrução seguida dos seus respetivos argumentos)

- Com o método *visitInfoVarDroneOne*:

Extrai informações e imprime a variável declarada

public Void visitProgram(DronesParser.ProgramContext ctx) {  
 System.*out*.println("Start of the program (visitor)...");  
 return visitChildren(ctx);  
 }  
  
 public Void visitInfoInstDroneOne(DronesParser.InfoInstDroneOneContext ctx) {  
 String instrucao = ctx.instructionDroneOne().getText();  
 System.*out*.print("Instruction: " + instrucao + " with arguments: ");  
 if (!ctx.info().isEmpty()) {  
 for (DronesParser.InfoContext infoCtx : ctx.info()) {  
 System.*out*.print(infoCtx.getText() + " ");  
 }  
 }  
 System.*out*.println();  
 return super.visitChildren(ctx);  
 }  
…  
 public Void visitInfoVarDroneOne(DronesParser.InfoVarDroneOneContext ctx) {  
 System.*out*.println("Variable: " + ctx.getText());  
 return visitChildren(ctx);  
 }  
…

Para esta user story, usamos visitors; pois assim, cada método trata um tipo específico de nó da árvore, facilitando a sua leitura.

3) Classe **DroneSyntaxValidator**

Esta classe é responsável por validar a sintaxe dos ficheiros que contêm código das linguagens específicas dos drones, garantindo que o código está conforme a gramática definida antes de avançar para o parsing e interpretação.

Para a validação, usamos listeners com o objetivo de captar facilmente erros durante a análise da árvore.

- Método *validate*

O foco deste método e na verificação de erros e tratamento de exceções:

- Verifica se o listener registou algum erro.

Se houver erros, imprime uma lista detalhada e retorna false, de forma a indicar que a validação falhou.

Se não houver erros, imprime uma mensagem de sucesso e retorna true.

- Caso ocorra alguma exceção durante a leitura do ficheiro ou análise, a exceção é capturada. É impressa uma mensagem de erro e o método retorna false.

public static boolean validate(String filePath) {  
 try {  
 String content = Files.*readString*(Path.*of*(filePath));  
 DronesLexer lexer = new DronesLexer(CharStreams.*fromString*(content));  
 CommonTokenStream tokens = new CommonTokenStream(lexer);  
 DronesParser parser = new DronesParser(tokens);  
  
 SyntaxValidatorErrorListener errorListener = new SyntaxValidatorErrorListener();  
 parser.removeErrorListeners(); // disable default error printing  
 parser.addErrorListener(errorListener);  
  
 parser.program(); // entry point in your grammar  
  
 if (errorListener.hasErrors()) {  
 System.*out*.println("SYNTAX ERRORS in file: " + filePath);  
 errorListener.getErrors().forEach(e -> System.*out*.println(" - " + e));  
 return false;  
 }  
  
 System.*out*.println("SYNTAX OK in file: " + filePath);  
 return true;  
  
 } catch (Exception e) {  
 System.*err*.println("Failed to validate file: " + e.getMessage());  
 return false;  
 }  
}

Usamos o mesmo raciocínio de resolução nas restantes user stories, ficando por resolver a **US348 - Show generation**.