# Assignment 1: Developing a Language for Ordering Game Console Packs

2IMP20 - DSL Design

May 12, 2023

## 1 Introduction

The goal of this assignment is to design and implement—using a language workbench—a Domain-Specific Language (DSL) for ordering game console packs (console with controllers and included game) from a vendor. This assignment must be implemented using the Rascal language workbench. Make sure to first read and follow the separate document with instructions on how to install the Rascal language workbench.

Then,

- Get the assignment's project skeleton from the template repository.
- Import the assignment's project into the current workspace (File → Import → Existing Projects into Workspace). Then, select the location where the project's repository was cloned.

## 2 Assignment

The first assignment is meant to get you acquainted with the basics of defining languages using Rascal. The goal of the assignment is to build a DSL called ConCL (Console Configuration Language) for defining game console packs (to order from a vendor). One of the tasks in this assignment is to develop a concrete syntax (using Rascal's grammar formalism for describing languages) for ConCL, which can be used to parse ConCL programs.

In the template repository you will find an Eclipse project with the skeleton of the assignment.

*NB: When opening this project, it may get stuck on the Rascal builder, which can not execute because of necessary Maven updates which in turn may be blocked by the Rascal builder. In that case, stop the Rascal builder in the progress bar of Eclipse, and restarting Eclipse now seems to fix the problem.*

Take a look at the `src` folder within Eclipse. This folder contains the language modules and each language module contains the instructions for the exercises.

ConCL allows users to define hardware resources by defining instances, and each instance has different properties. We will use ConCL to demonstrate the various aspects of language design

using a language workbench (Rascal). Hereafter, we introduce more details about the ConCL language.

## 2.1 Grammar

A game console pack **must** have a label and contains a list of components. Each component provides information describing the console, controllers, and included game involved. It can thus be of one of three types: *console* or *controller* or *game*.

A *console* component has storage and a display.

A *storage* component has a *size*. The storage's size is defined by an integer value that ranges from 32 until 1024 GB.

A *display* component is defined by its diagonal size (in inches), type (either LED or OLED) and an indicator of its number of pixels (one of HD, Full-HD, 4K, 5K).

The *controller* is standardized, but has a colour: black, white, red, blue, gold, silver, or green.

As for *game* included with the console pack: either no game is included, or a game called Hedwig the Hedgehog, or a game called Link's Resolution.

## 2.2 Well-formedness Resources

In order to have a valid ConCL console pack definition, some requirements have to be satisfied. Some of these requirements can not be expressed in context-free grammars. The following conditions ensure the well-formedness of a ConCL hardware definition.

- Every console pack must have at least one controller and at most four controllers included.
- It is not allowed to have multiple controllers of the same color included in a game pack.
- If the game included in a game pack is Link's Resolution, then this game pack must at least have a green controller included.
- If the game included in a game pack is Hedwig the Hedgehog, it must at least have a blue controller included, and must not include a green one.
- Display type must be valid. In other words, the type and pixel indicator have to be one of the two and four types respectively mentioned in Section 2.1.
- Do not accept duplicate components with the exact same configuration and different labels.
- The language supports positive integers and reals.

Figure 1 shows a valid simple hardware definition using the ConCL language.

# 3 Deliverable

The first assignment consists of four parts:

- Define a concrete syntax of ConCL using Rascal's grammar formalism (module `Syntax.rsc`).

```
console_pack my_console_pack {
    console {
        storage: 1024 GB,
        display {
            diagonal: 30 inch,
            type: LED,
            resolution: Full-HD
        }
    },
    controller {
        colour: blue
    },
    controller {
        colour: black
    },
    game {
        name: Hedwig the Hedgehog
    },
}
```

Figure 1: Example of a ConCL program specifying a console pack with a console, two controllers, and a game.

- Define a parse function for ConCL. The name of the function is `parserConCL(...)`. It gets a location (`loc`) as parameter and it returns a concrete ConCL resource (module Parser.rsc).
- Define an abstract syntax for ConCL (module `AST.rsc`).
- Define the function `cst2ast(...)`, which takes a parse tree of a ConCL resource as parameter and returns an abstract syntax tree as described in the AST (module `CST2AST.rsc`). **Obviously, you are not allowed to use Rascal's built in *implode* function.**
- Specify a well-formedness checker for ConCL.
  Define the function `checkHardwareConfiguration(...)`, which takes the AST of a resource as parameter and verifies that all well-formedness checks pass (module `Check.rsc`).

# 4 Testing

The assignment's skeleton contains a module called `Plugin.rsc`. This module registers the ConCL into Eclipse. This means that Eclipse will recognize files with extension `.concl`, and it will call the ConCL's parser. If the program is syntactically correct, you should observe syntax highlighting on it. If that is not the case, it is highly probable that the program contains a parse error. To activate such functionality you have to open Rascal's REPL, import the module `concl::Plugin`, and call the `main()` function. Likewise, this module also contains a function called `checkWellformedness()` that receives as a parameter the path of a ConCL program and returns a Boolean value. If the program is correct, it returns true; it returns false if the program is not well-formed.

Now, you can create your first `myfirst.concl` file. Open it and enter your first hardware

specification using ConCL. Observe what happens if you write a syntactical correct and incorrect ConCL specification.

# 5 Submission

You have to submit a zip file containing:

- Your ConCL language solution including all modified files.
- The test programs demonstrating the correct parsing of non-trivial ConCL specifications.
- Test programs containing well-formedness check errors.

It is also important to add comments to the files explaining the modifications/extensions you have made. You have to submit this zip file *as a Canvas group of two students* before 23:59 of Monday 29th of May via Canvas.