UNIVERSITÉ CÔTE D'AZUR
SI5/M2 INFO
2025-2026

Julien Deantoni

Domain Specific Language

SlideDeckML

Project delivery

This project concerns the second delivery of the DSL course. It will include the implementation of a DSL focused on the specification of slide deck definition, in *either* an internal *or* an external DSL. Deliveries are expected by email (to Julien Deantoni: firstname.lastname@univ-cotedazur.fr, with [DSL] as object prefix) followed by "team X lab 2" where X is the name of your team on slack. The delivery is expected before the 18th of January 2026 at 10:00PM Paris Time. The delivery is expected as a PDF report (please let your report be succinct and rigorous).
The report must contain :

- the name of the members of your team
- a link to the code of your DSL
- a description of the language proposed:
  - the domain model represented as a class diagram;
  - the concrete syntax represented in a BNF-like form;
  - a description of your language and how it was implemented;
  - a simple description on how you wrote the compiler to obtain executable code
- a set of relevant scenarios implemented by using your language(s) (internal or external);
- a critical analysis of (i) DSL implementation with respect to the ImpressML use cases and (ii) the technology you chose to achieve it;
- responsibility of each member in the team with respect to the delivered project.
- a (set of) example slide decks realized with your DSL to demonstrate its usage and **all** of its features. The associated script and materials will be provided.

Objectives: Define the SlideDeckML language

Your objective is to define a DSL that supports the creation of a wide range of slide decks, including potentially complex ones. Core functionalities should include all the basic features of main stream tools like slide definition, template usage, media insertion, and customization options (e.g., text color and style). However, the DSL may also provide additional modern features such as transition effects, slide animations, non-linear navigation, slide annotations, and any other functionality that could enhance contemporary presentations.

The intended users of this DSL are computer science students and teachers. Although they are familiar with web technologies, the goal is to enable them to build web-based slide decks quickly and easily, without having to write HTML5 directly. Additionally, it is important that the DSL supports offline usage (e.g., when a presentation is delivered abroad, where internet access may not be reliable).
From your programs written in the *SlideDeckML* language, you will generate directly usable presentations. The choice of target language and/or framework is up to you, but it must be web-based and free to use, in order to ensure easy deployment and sharing across different operating systems and platforms.
A number of libraries already exist for slide-deck rendering, such as the well-known impress.js and reveal.js, as well as other solutions (e.g., those referenced in curated lists of HTML presentation frameworks like here: `https://www.jqueryscript.net/blog/best-html-presentation-framework.html`). These may serve as inspiration, but you are free to select any suitable technology that aligns with the project requirements.

Basic scenarios

1. **Scenario 1**

   A student wants to prepare slides for apprenticeship following this path :

   (a) import a presentation template imposed by her company. It contains at institutional logos, the company name and use specific font and colours;

   (b) add a title slide with the name of her project, current status and a picture of the site where she works.

   (c) add an outline slide with either ordered or unordered list of items,

   (d) add different slides with various animations (appearance of text and pictures),

   (e) add a slide with a video that demonstrate the work done in situ.

   (f) add conclusion slide

   (g) export the slide deck as a web-based presentation usable offline.

2. **Scenario 2**

   A teacher wants to prepare slides for an introduction to programming in python. The slide deck is like this :

   (a) after the addition of usual introductory slides;

   (b) add other slides containing some code automatically highlighted, with animation so that it appears line by line. Jointly with the lines revealed, a picture evolves according to the code.

   (c) add slides mixing text and pictures with possibly to choose their position

Commmon Parts

The following parts must be available in your DSL:

- **Domain Model**:

  The domain model (a.k.a. abstract syntax) should be clearly identified in the delivered code. It will be provided as a class diagram, together with explanation about the main choices you did.

- **Concrete syntax**:

  The concrete syntax (external or embedded), in a BNF form, must be clearly identified and used by a relevant set of scenarios. The syntax must leverage the tool chosen to implement it to make it clear and easy to use.

- **Validation**:

  Support your end-user by checking all possible arguments for type and range.

- **Code generation**:

  Provide a generator producing a web based slide deck. The target language and framework is your choice.

- **DSL Addons for User Acceptance** The acceptance of a language by users is usually not only due to the language itself but also on the extra services provided by the languages. These services can be related to the language itself (like the borrow checker in Rust) or used to help the user in the definition of its programs (like for instance a graphical representation of the program control flow or a very precise and didactic error spotting). You will provide at least one extra service for the SlideDeckML DSL.

"À la carte" features

The remainder of this document describes some extensions that you can implement on top of your language. Each extension is defined by a short description and at least one acceptance scenario.
Choose the feature(s) to introduce in your project. Feel free to propose other features you would like to develop.

Support for Live Programming Session

This extension makes it possible to embed a live programming environment directly into a slide. It may, for example, contain an interactive notebook cell. The editor should support at least syntax highlighting, and an enhanced output stream should also be available.
Scenario: A student wants to demonstrate the simplicity of using a new API and therefore decides to conduct a live coding session. The API facilitates the retrieval of process monitoring data. To make the presentation more appealing, the student intends to visualize the retrieved data using a graphical plot.

Support for Slide Annotations

This extension enables the presenter to annotate any slide in real time. It provides a virtual pen tool that allows freehand drawing, highlighting, or marking directly on the slide content. Annotations should be temporary by default, with the option to save them if needed.
Scenario: During a lecture on system architecture, the instructor wants to emphasize the flow between components on a diagram. Instead of creating multiple pre-drawn slides, he uses the virtual pen to draw arrows, circles, and notes live, guiding the audience's attention interactively. This approach makes the presentation more dynamic and easier to follow, especially when explaining complex relationships.

Interactive Quizzes and Live Polling

This extension allows the presenter to engage the audience through embedded quizzes and live polls directly on the slides. Quiz questions can be multiple-choice or short-answer, with instant feedback displayed. Simultaneously, live polling collects audience responses in real time, aggregating and displaying the results as charts or word clouds. It should be possible to show results only on demand.
Scenario: During a workshop on software performance, the instructor presents a short quiz about sorting algorithms. Attendees select their answers on their devices, and the slide instantly shows which choices were most popular. At the same time, a live poll asks the audience which programming language they prefer for performance-critical tasks, and a dynamic bar chart updates in real time. This interactive setup keeps participants engaged, provides immediate feedback, and visually highlights trends in the audience's understanding.

Support for Non-Linear "Zoomable" Presentations

This extension allows the presenter to organize slides on a 2D or 3D canvas and navigate them non-linearly using pan and zoom transitions. Each slide or section can be positioned anywhere on the canvas, and the presenter can move fluidly between topics in any order, creating a storytelling effect similar to Prezi or impress.js. While previously mentioned libraries allow many different navigation which may be very complex to specify, this extension should identify interesting ones and be simple to enact/edit.
Scenario: During a lecture on network architectures, the presenter places each subsystem on a canvas as a separate node. Rather than following a fixed sequence, they zoom into the "routing" node first, pan to "firewall" next, and then zoom out to show the overall architecture. This approach allows them to adapt the narrative dynamically, emphasize connections visually, and maintain audience engagement without being constrained by linear slides.

DSL Extension for Image Annotation

This extension allows the DSL author to declare images that can be annotated directly within a slide. It supports specifying shapes such as rectangles or arrows, applying visual styles, and optionally defining sequences of steps to create simple animations. The annotation is expressed declaratively in the DSL, enabling the presenter to highlight or emphasize specific areas of an image in a controlled and reusable manner.
Scenario: During a lecture on object detection, a student creates a slide showing a traffic scene. They use the DSL to specify rectangles around cars and pedestrians and define a sequence highlighting each element in turn. The annotations allow the audience to clearly follow the explanation without requiring the presenter to draw manually during the session.

DSL Extension for Image and Text Placement

This extension enables the DSL author to precisely position images and text zones on a slide. It supports specifying coordinates, layering, and relative positioning, allowing the creation of complex layouts and overlays in a declarative way. The DSL syntax should make it easy to describe where each visual element should appear without requiring manual manipulation of low-level styling or HTML structures.
Scenario: In a lecture on neural networks, a student designs a slide showing multiple layers as images, with labels and explanatory text positioned on top. Using the DSL, they specify the placement of each image and text zone, ensuring a clear and visually appealing layout that conveys the relationships between components effectively.

DSL Extension for Mathematical Equations

This extension allows the DSL author to include mathematical equations directly within slides using a LaTeX-like syntax. Equations can be specified inline with text or as separate blocks, providing a clear and readable way to present formulas and scientific notation. The DSL syntax abstracts away the underlying framework details, making it easy to write, maintain, and reuse equations across multiple slides.
Scenario: In a physics lecture, a student creates a slide explaining kinetic energy. They declare the formula for kinetic energy directly in the DSL. During the presentation, the equation is displayed correctly on the slide, clearly formatted and readable, allowing the audience to focus on the concept rather than on formatting issues.

DSL Extension for Interactive Controls

This extension allows the DSL author to leverage standard web interactivity features such as hover effects, checkboxes, dropdowns, and sliders. These elements can be used to switch between examples, toggle annotations, or adjust parameters of an animation directly from the slide. The DSL abstracts away the underlying HTML and JavaScript, making it simple to add dynamic, interactive behaviors to slides declaratively.

Scenario: In a lecture on sorting algorithms, a student creates a slide showing multiple sorting visualizations. Using checkboxes and a dropdown declared in the DSL, he can select which algorithm to display, while a slider adjusts the animation speed. Hovering over array elements highlights them to emphasize key operations.

DSL Extension for Interactive Plot Elements

This extension allows the DSL author to embed plots with built-in interactivity, such as highlighting points or displaying tooltips when hovering over elements. The DSL can declaratively specify the data and plot type, while the generated slide supports direct exploration of the visualization by the audience. This makes it possible to convey additional information without cluttering the slide and allows viewers to inspect values or relationships dynamically.
Scenario: During a statistics lecture, a student creates a scatter plot showing test scores versus study hours. Using the DSL, the slide automatically shows the value of each point when the audience hovers over it. This lets the audience quickly see details for individual data points while keeping the visualization clean and focused on overall trends.

DSL Extension for Reusable Components

This extension allows the DSL author to define reusable components or macros that can be instantiated multiple times across a slide deck. Components can encapsulate combinations of images, text zones, plots, annotations, or interactive elements, making it easy to maintain a consistent style and reduce repetitive declarations. The DSL abstracts the instantiation logic, allowing authors to focus on content rather than boilerplate.
Scenario: During a lecture on sorting algorithms, a student defines a reusable component representing a labeled array with optional highlighting. They can then instantiate this component on multiple slides, specifying only the array values or highlighted elements each time. This ensures a consistent layout and style while saving time and keeping the DSL concise.