Domain Specific Languages

Quiz

Benzarti Zied

Destefanis Marc

Lozach Maxime

Le 16/02/2016

Énonce de notre projet de Quiz

Notre projet consiste en la création d'un DSL qui permettrait de réaliser un quiz.

Le code généré utilise un écran LCD qui va afficher les informations à l'écran comme les questions, le score. Deux boutons qui permettent au joueur de donner une réponse 'Oui' ou 'Non'. Deux LEDs qui notifient le joueur à la suite de la réponse à une question.
L'utilisateur de notre DSL doit avoir la possibilité de choisir les pins sur lesquels brancher les boutons, les LEDs et l'écran LCD. Il peut ajouter autant de questions/réponses qu'il le souhaite et une question doit être accompagnée de sa réponse True ou False.

Choix de conception du meta-modèle

Pour réaliser notre DSL, nous avons préféré continuer sur MPS en réalisant un DSL externe.
Au cours de la réalisation du projet, le meta-modèle a connu plusieurs aspects.
Dans un premier temps, nous sommes restés sur la base donnée par le tuto avec une application à la racine de l'arbre et deux enfants possibles : les bricks et les états. Afin de pouvoir réaliser les 4 situations de base, le modèle des états a évolué afin de pouvoir accepter plusieurs transitions, afin qu'un état puisse changer différemment en fonction des conditions exprimées. Les transitions ont elles aussi été améliorées pour pouvoir définir un ensemble d'action à réaliser simultanément pour réaliser le changement d'état.
Avec ces modifications, les quatre cas de base pouvaient être modélisés avec notre DSL, ce qui validait pour nous, la possibilité de commencer le projet sans risque d'être bloqué dans le futur.
Malheureusement ce fut le contraire. Pensant que notre modèle qui permettait de réaliser une machine à état était le bon, nous avons continué l'amélioration du modèle en ajoutant l'intégration du LCD, en tant que sous-modèle de Brick. C'est à partir de ce moment-là avec le début de l'évolution du modèle pour ajouter les questions et les réponses que nous avons commencé à sérieusement remettre en question l'ensemble du modèle. Pour la réalisation du quiz, la machine à état était toujours la même, avec l'affichage des questions, l'écoute des boutons, le calcul du score et l'état final. C'est pourquoi nous avons commencé à réfléchir à ajouter un niveau d'abstraction au modèle et dissimuler complètement la notion de machine à état. Celle-ci étant fixe, elle serait générée lors de l'interprétation du modèle.
Partant de ce postulat, nous sommes repartis de zéro (ou presque) pour redéfinir notre modèle. Il est apparu que ce dernier s'imposait moins comme un langage de programmation qu'un langage de configuration. Nous avons donc désigné le modèle en gardant l'application à la racine, mais cette fois avec autant de noeuds enfant qu'il y avait de champ à configuré sur notre DSL. De cette manière, nous exposions pour l'utilisateur seulement les éléments où il était nécessaire qu'il y ait une action en l'abstrayant de la réalisation de la machine à état, dont l'exposition était devenu inutile.

Liste des extensions et implémentations

Dans le cadre de notre sujet, plusieurs extensions ont été implémentées. Nous avons, dans un premier temps, la génération de quiz via notre DSL. L'implémentation se faisait à la base

de manière séparée à ArduinoML, dans l'idée de séparer la construction de la machine à état de la construction du quiz. Seulement, après la révision du modèle, les questions sont devenues un des paramètres d'ArduinoML. Afin de s'assurer qu'il y ait le même nombre de questions et de réponses, et pour faciliter l'association de ces deux éléments, les questions et les réponses sont définies par paire. Les questions sont représentées par une phrase exprimée par l'utilisateur, qui ne doit pas être entourée par des guillemets et qui peut contenir n'importe quel caractère. Les guillemets étant convertis pour ne pas être interprétés. Les réponses se modélisent quant à elles, soit par "true" soit par "false".
L'implémentation des boutons et des LED que nous avons réalisés durant le , ont été gardée, mais légèrement modifiée afin d'abstraire le nom des éléments, car ceux-ci sont toujours fixes. Seul le pin, sur lequel ils peuvent être branchés, est laissé au choix de l'utilisateur.
Nous avons également ajouté la gestion du LCD afin d'afficher les questions. L'utilisateur peut choisir sur quel BUS brancher le LCD, sans donner de nom ici aussi, étant donné qu'il n'y a qu'un seul LCD pour le quiz. Afin de donner la possibilité à l'utilisateur de changer la vitesse d'écriture du texte, nous avons mis à disposition trois vitesses au choix. Nous ne voulions pas laisser une notion brute et plutôt abstraire cette notion afin que l'utilisateur ne perde pas trop temps sur une configuration secondaire. Le LCD que nous avons implémenté est capable d'afficher des questions sur plusieurs lignes avec le défilement de l'écran tout en prenant en compte les évènements de clic sur les boutons de réponse à une question. Il s'agit de la partie la plus complexe de notre projet et elle nous a demandé de revoir notre code à plusieurs reprises afin de la mettre en place comme nous le souhaitions.

Création de scénarios

Le sujet s'est construit sur un scénario de base : l'utilisateur veut créer un quiz se définissant par un ensemble de questions où l'on peut répondre par "oui" ou par "non".
L'utilisateur part alors du langage de base avec une configuration donnée par défaut et doit simplement formuler ses questions, ses réponses, configurer les boutons, les LEDs et l'écran LCD.

Nous avons fait en sorte que la syntaxe concrète soit la plus simple possible à écrire, à comprendre et à maintenir. Par exemple nous avons pris soin de rapprocher la réponse de sa question en mettant ces deux éléments sur la même ligne, de ce fait si l'utilisateur de notre DSL a écrit une grande quantité de questions et qu'il veut en supprimer une, il supprime la réponse sans avoir à la rechercher.

Ci-dessous la syntaxe concrète pour un scénario d'exemple qui contient 3 questions/réponses et dans lequel la vitesse de défilement de la question sur l'écran LCD est 'normale' :

```
Quiz :
  Buttons:
      yes: plugged in D9
      no: plugged in D10
  Leds:
      green: plugged in D11
      red: plugged in D12
  LCD:  plugged in BUS1 with text speed NORMAL
  Questions:
    Le cameleon a une tres courte langue --> False
    Le celadon designe la couleur vert pale --> True
    Un charme est un arbre --> True
```

Ci-dessous un autre scénario d'exemple qui ne contient qu'une question et dans lequel la vitesse de défilement de la question sur l'écran LCD est lente :

```
Quiz :
  Buttons:
      yes: plugged in D9
      no: plugged in D10
  Leds:
      green: plugged in D11
      red: plugged in D12
  LCD:  plugged in BUS1 with text speed SLOW
  Questions:
    Einstein a un lien de parente avec Frankeinstein --> False
```

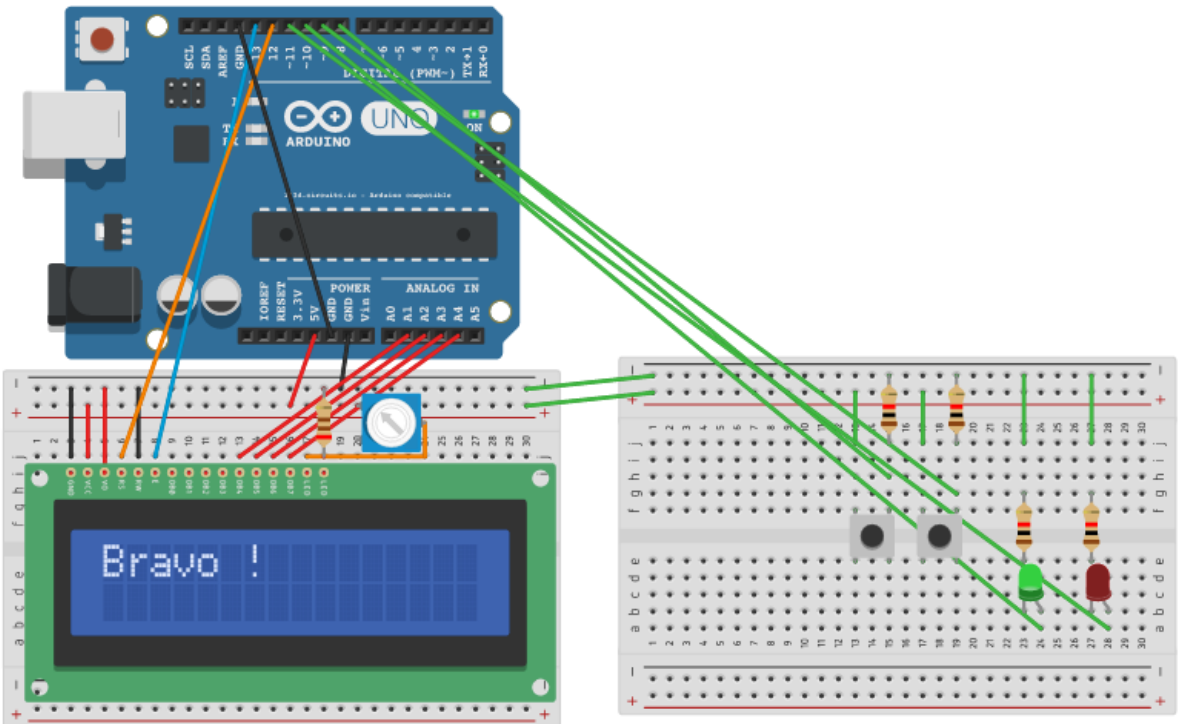
Analyse critique

1. L'implémentation du DSL en respectant le cas d'utilisation d'ArduinoML

Notre DSL dans son état final ne respecte plus le cas d'utilisation de base d'ArduinoML, à savoir la définition d'états et de leurs transitions. Afin de correspondre à la philosophie des DSL qui est d'abstraire au langage tout ce qui est superflu pour ne pas laisser à l'utilisateur des leviers explicites et pertinents pour modéliser son application, nous avons retiré la notion de machine à état qui devenait de plus en plus secondaire pour notre langage. Nous avons fait en sorte que notre langage soit utilisable par le plus grand nombre d'utilisateurs, notamment des utilisateurs n'ayant aucune notion en programmation.
Les principales lacunes de notre langage sont dues au manque de compréhension de l'outil qu'est MPS. Nous aurions aimé pouvoir empêcher l'exécution du DSL si les contraintes n'étaient pas respectées, au lieu d'avoir simplement des erreurs affichées dans l'éditeur. Malgré ça, l'utilisateur garde assez peu de possibilités de se tromper et devra faire preuve de beaucoup de mauvaise volonté pour avoir une génération fausse.

2. Choix technologiques

Afin de réaliser le code Arduino que nous comptions créer avec notre DSL nous avons utilisé l'application '123D Circuits' pour pouvoir travailler en dehors des séances de travaux dirigés. Après nous être documentés, nous avons réalisé le montage suivant et nous avons pu travailler sur le code afin d'obtenir le jeu de quiz que nous voulions.



Le projet '123D Circuits' est disponible en cliquant sur le lien suivant :
https://123d.circuits.io/circuits/1615129/edit

Pour nous, il était absolument essentiel de pouvoir coder à la maison, car les TPs ne suffisaient pas. Nous n'avions jamais fait auparavant de code Arduino et le fait de choisir la réalisation d'un DSL de quiz nous a entraînés dans un code complexe qui nécessitait d'être fréquemment testé.

Suite à l'étude que nous avons réalisée dans le rapport précèdent, nous avons choisi un DSL externe afin de gérer au mieux la taille de notre domaine. Sans entrer de nouveau dans le détail, notre choix s'est arrêté sur MPS, car c'est un DSL pour les DSL qui permet de s'abstraire de la partie code pour se concentrer sur l'essentiel du travail, à savoir : l'abstraction du langage et c'est celui que nous trouvons le plus efficace.

Organisation du travail

Maxime

Mise en place du projet.
Refactor vers le nouveau modèle.
Simplification de la génération des templates des états
Contraintes sur le langage

Zied
Implémentation du jeu de quiz en Arduino
Gestion du défilement de texte sur l'écran LCD sur Arduino
Prise en main et mise en place de notre projet sur l'émulateur '123D Circuits'
Refractor code MPS après l'ajout de différents états dans l'Arduino

Marc
Implémentation du jeu de quiz en Arduino
Prise en main et mise en place de notre projet sur l'émulateur '123D Circuits'
Gestion de la vitesse de défilement du texte sur l'écran LCD dans le DSL
Gestion des LEDs dans le DSL

Conclusion

Nous sommes satisfaits de notre travail, mais avec plus de temps nous aurions pu mettre en place de nouveaux principes dans notre quiz et d'offrir de nouvelles configurations comme par exemple un message personnalisé lorsque le joueur a juste ou faux à une question ou même l'ajout d'éléments comme un buzzer qui sonne lorsque le joueur répond faux à une question.
Nous avons essayé de mettre au point un DSL qui permettrait à un utilisateur d'écrire un court programme, facile à comprendre, à écrire et qui recouvrirait au mieux le domaine de notre sujet sans en sortir.
Nous ne regrettons pas notre choix technologique, l'utilisation de MPS fut enrichissante et nous a permis d'écrire un DSL qui répond à nos attentes et on l'espère à celles des futurs utilisateurs.