Du code aux modèles, des modèles au code: enseigner les langages dédiés (DSL)

Laure Gonnord, Sébastien Mosser

▶ **To cite this version:**

Laure Gonnord, Sébastien Mosser. Du code aux modèles, des modèles au code: enseigner les langages dédiés (DSL). CIEL 2018 : 7ème Conférence en IngénieriE du Logiciel, Jun 2018, Grenoble, France. pp.1-4. hal-01816239

HAL Id: hal-01816239
https://hal.science/hal-01816239

Submitted on 15 Jun 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Du code aux modèles, des modèles au code: enseigner les langages dédiés (DSL)

Laure Gonnord[1] and Sébastien Mosser[2]

[1] Univ Lyon, Université Claude Bernard Lyon 1,
LIP, CNRS, ENS de Lyon, Inria,
F-69342, LYON Cedex 07, France Laure.Gonnord@end-lyon.fr
[2] Université Côte d'Azur, CNRS, I3S, France mosser@i3s.unice.fr

Résumé

Cet article relate notre expérience d'enseignement des langages spécifiques aux domaines (DSLs), notamment via l'utilisation d'un énoncé de Travaux Pratiques guidé permettant d'expérimenter les différents concepts et niveaux d'abstraction mis en oeuvre lors de la conception de tels langages. Le domaine d'illustration choisi est celui des systèmes embarqués réactifs, avec une application aux microcontrôleurs Arduino.

1 Introduction

Enseigner les langages dans l'enseignement supérieur est une activité ardue, qui à notre sens ne peut et ne doit pas se résumer à un catalogue de concepts accompagnés d'exemples jouets. Nous proposons d'attaquer ce problème via l'enseignement des langages spécifiques aux domaines, de leur outillage théorique et pratique, en joignant nos compétences en compilation, conception de langages, sémantique des langages de programmation et génie logiciel.

Nous proposons un cours « brique de base » sur 8 semaines, qui pourra servir de fondation à des cours plus avancés via des extensions. L'objectif est de permettre aux étudiants d'acquérir des connaissances et compétences langages indispensables comme la (méta) modélisation, la notion d'abstraction, les choix de conception, et l'outillage théorique et pratique pour la génération de code. Le cours sera également l'occasion d'aborder des thématiques plus spécialisées comme la vérification formelle et la génération de code avancée.

Ce cours a été mis en œuvre deux fois en 2017-2018 :
— A l'ENS de Lyon, dans le cadre du M2 Informatique Fondamentale. Nous avons alors donné au cours une dimension recherche, et moins insisté sur les points de vue implémentation.
— À Polytech Nice, en 5ème année d'école d'ingénieur, avec une orientation ingénierie, et où la dimension recherche a été plus réduite.

2 Objectifs et mise en oeuvre

Les objectifs du cours sont les suivants :

1. illustrer comment abstraire du code en modèles ;
2. identifier les différents contextes d'utilisation des outillages « compilation » et « métamodélisation » selon les caractéristiques finales désirées (facilité de développement, types d'utilisateurs, expressivité...) ;
3. étudier les outillages autour des différents concepts ;

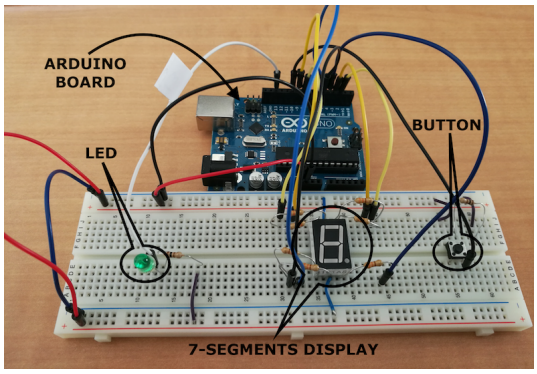

FIGURE 1 – Plateforme d'expérimentation : Arduino et *breadboard*

4. appliquer les connaissances précédentes pratiquement, via un TP permettant d'approfondir l'expérience sur un cas pratique non jouet.

Nous proposons un cours en deux phases. Tout d'abord, les étudiants travaillent à la réalisation de plusieurs versions d'un langage spécifique à la programmation de microcontrôleurs. À travers neuf réalisations différentes d'un langage dédié au même problème (réalisations à différents niveaux d'abstractions ou à l'aide de technologies alternatives), les étudiants explorent en quatre semaines le champ des possibles. Ensuite, ils choisissent une des technologies étudiées pour mener à bien la réalisation d'un nouveau langage, dans une autre étude de cas.

3 Un énoncé de travaux pratiques à plusieurs niveaux

Nous développons ici la première phase. Le contexte que nous choisissons est celui des systèmes embarqués. Ce domaine applicatif est à la fois suffisamment simple pour que le code à écrire ou générer ne demande pas (trop) d'expertise en programmation, et suffisamment expressif pour être représentatif d'un domaine dédié non trivial. Nous choisissons de réaliser un TP de bout en bout, sur de vraies plateformes matérielles autour de la carte Arduino (cf. Figure 1).

Nous utilisons des Arduino UNO [1], technologie open-source, ainsi qu'une plateforme "*breadboard*" contenant un bouton poussoir (capteur), une led et un afficheur 7 segments (actionneurs), pour un coût d'environ 40 euros par plateforme. Les plateformes sont données par les enseignants pour permettre au cours de mettre l'accent sur les langages, mais le montage électronique associée est simple. Il est de plus possible d'acheter du matériel précablé (Arduino dispose d'un système de *shield* emboîtable avec des montages prédéfinis), pour un coût supérieur.

Déroulement Ce TP est une suite de neuf étapes avec du code fourni sous licence libre [2], chaque étape étant construite selon le schéma suivant :

1. Un exemple minimal où la led clignote à une fréquence d'$1Hz$ est fourni aux étudiants, dans le langage et ou la technologie introduite dans l'étape. Les étudiants peuvent compiler et exécuter cet exemple sur la plateforme. Ils sont alors invités à critiquer la solution en terme de performance, utilisabilité, lisibilité...

1. https://en.wikipedia.org/wiki/Arduino
2. https://github.com/mosser/sec-labs/tree/master/lab_1

2. Ensuite, les étudiants doivent modifier l'exemple de façon non triviale, représentative du domaine. Ici, nous proposons d'utiliser un afficheur 7-segments pour compter le temps, ainsi qu'un bouton poussoir pour le remettre à zéro. Cet exemple requiert d'introduire la notion d'état mémoire dans le code ou l'outil de modélisation.

Les étapes de notre TP permettent d'explorer différents niveaux d'abstraction, de discuter leurs avantages et leurs inconvénients, de comparer différentes formes de (méta) modélisation et les outillages autour de la génération de code. Voici plus précisément les étapes que nous proposons :

— À la première et deuxième étapes, la programmation C Arduino, la chaîne de compilation et de déploiement sont étudiés. Nous travaillons au niveau des registres du microcontrôleur, et démontrons le passage d'un code C générique à une API dédiée, même à bas niveau.
— Les étapes trois et quatre illustrent comment une approche à base de modèles (ici des automates de contrôle, FSM) permettent à l'utilisateur du langage de travailler à un niveau d'abstraction plus proche de son métier, puis d'obtenir du compilateur le code de plus bas niveau à déployer sur le microcontrôleur. On peut commencer à vérifier des propriétés sur les automates, ce qui était difficile à faire au niveau du code.
— Les étapes 5 et 6 posent les questions de l'utilisation des bonnes abstractions. Les FSMs souffrent d'un problème d'explosion combinatoire quand il s'agit de les composer, or dans le cas fil rouge, les étudiants doivent composer l'application de contrôle de la LED avec celle réinitialisant le compteur temporel. On explore dans cette étape la définition d'un système réactif, à travers le langage dédié Lustre[3], et aussi à travers la réalisation d'un métamodèle réactif. Ce changement d'approche de modélisation (FSM vers système réactif) permet de montrer qu'à technologie identique, le choix des bonnes abstractions a un fort impact sur l'expressivité et les capacités fournies par le langage. Ces deux étapes sont des étapes clefs dans l'avancée, elles prennent donc un temps substanciel. La prise en main d'un sous-ensemble de Lustre est relativement aisé à ce stade, grâce aux concepts vus auparavant, et au fait que le langage a un très petit nombre de constructions syntaxiques.
— Les étapes 7, 8 et 9 n'ont plus pour but d'explorer des niveaux d'abstraction mais portent sur la réalisation de langages dédiés, en utilisant trois technologies différentes : langage embarqué dans un langage hôte, langage externe, et édition projectionelle.

Les étudiants sont invités à rédiger un rapport (non noté) comportant les étapes clefs et les choix de conception de leurs différentes solutions. Ces conclusions seront ensuite mises en œuvre dans la deuxième phase du cours.

La seconde phase du cours porte sur la définition d'un langage répondant à un appel d'offre. Ce langage, plus ambitieux, permet d'exploiter les compétences acquises dans les étapes 4 à 9 pour que les étudiants choisissent un paradigme et l'appliquent de manière plus poussée. Parmi les études de cas traitées, on peut citer par exemple la réalisation d'un langage de définition de workflow scientifique (avec projection sur l'environnement d'exécution Taverna[4]), ou encore la réalisation d'un langage de simulation de capteurs pour une "*Smart City*", avec une pile technologique InfluxDB/Grafana[5] pour collecter les données produites et les afficher.

3. http://www-verimag.imag.fr/The-Lustre-Programming-Language-and?lang=en
4. https://taverna.incubator.apache.org/
5. https://www.influxdata.com/, https://grafana.com/

4 Conclusion

L'expérience d'enseignement décrite dans ce papier permet de démontrer une mise en oeuvre sur une courte durée d'un enseignement dédié aux langages et à la modélisation, au niveau M. Même si ce cours n'a été monté qu'au niveau M2, ses pré-requis (programmation objet et modélisation) sont compatibles avec une mise en oeuvre au niveau M1. Ce cours demandant une prise de recul sur des codes existants et l'appréhension de paradigmes différents, il nous paraît ambitieux de le reprendre en l'état au niveau L. Nous prévoyons de réaliser une étude plus fine des retours d'étudiants dans un futur proche. L'étape suivante est de développer ce TP et de diffuser le code plus largement, en définissant et documentant mieux les étapes pour qu'elles soient reproductibles sans les instructeurs. Nous prévoyons aussi de développer notre collaboration autour d'un TP plus avancé, notamment sur les aspects compilation et vérification formelle. D'autres pistes de collaboration sont envisageables pour étendre ce travail, comme par exemple des collaborations avec des groupes de travail du GDR GPL sur différents thèmes (ingénierie des exigences, méthodes formelles, sécurité, ...) pour élargir le spectre des notions abordées.