Kaboom: une boîte à outils pour la programmation à la volée en C/C ++ .

Damien Marchal

Kaboom: une boîte à outils pour la programmation à la volée en C/C++ .

Damien Marchal
Université de Lille 1/CNRS
50 Avenue Halley
59650 Villeneuve d'Ascq
damien.marchal@lifl.fr

RÉSUMÉ
En tant que développeurs nous sommes des habitués du cycle édition-compilation-exécution. Pourtant même lorsqu'on développe en C/C++ d'autres approches plus interactives de la programmation sont possibles. C'est notamment le cas d'une pratique qu'on appelle programmation à la volée qui consiste à pouvoir modifier une application pendant son exécution. Lors de cette démo, nous présentons Kaboom, une boîte à outils que nous développons qui rend possible la programmation à la volée pour les développeurs C/C++ .

Mots Clés
Programmation à la volée; Développement logiciel interactif; Manipulation directe.

ACM Classification Keywords
H.5.m Information interfaces and presentation (e.g., HCI): Miscellaneous.; D.2.3 Programming environement: Interactive environement.

INTRODUCTION

Il n'est jamais aisé de quantifier la popularité d'un langage de programmation [2], cependant il ne fait aucun doute que le C et le C++ sont des langages majeurs et toujours d'actualité comme en témoignent les régulières mises-à-jours qu'ils subissent. Pourtant, bien que les langages évoluent, on peut constater que les pratiques des développeurs C/C++ continuent de s'appuyer sur un cycle de développement édition-compilation-exécution. Ce cycle, que David Ungar fait remonter aux mainframes des années 60 [6], est composé des étapes suivantes :

- édition du code source ;
- compilation ;
- liaison des symboles ;
- lancement de l'application ;
- mettre l'application dans l'état adéquat ;
- test de la modification.

Avec ce cycle est qu'il n'est pas rare que, entre la modification du code source et le test, plusieurs minutes ne ce soient écoulées. Ce problème a été identifié dès les travaux sur le langage Self [7] mais c'est depuis les communications de Bret Victor [8] qu'on observe un regain d'intérêt vis à vis de pratiques de programmation plus interactive, comme la programmation à la volée, où le développeur modifie l'application pendant son exécution.

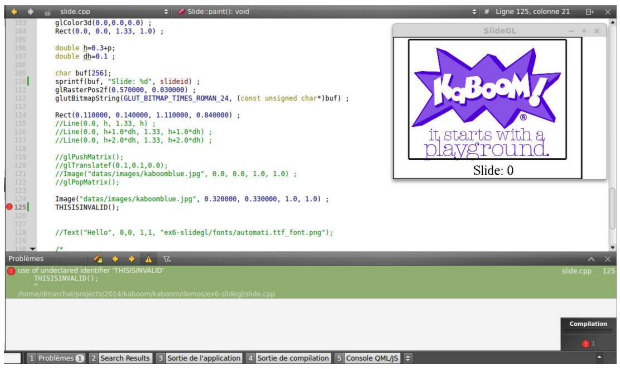

Figure 1. Développement d'une applicaton OpenGL avec QtCreator et Kaboom. A chaque modification du fichier source l'application, en cours d'exécution, est mise à jour.

Dans cet optique nous développons Kaboom, une boîte à outils pour la programmation à la volée pour le C/C++. Avec Kaboom notre objectif est d'explorer la tâche de programmation C/C++ comme une tâche interactive finalement assez proche de la manipulation directe tout en se contraignant à ce que les outils développés : n'imposent pas de changer de langage ; soient compatible avec les différents toolkits ; aient un impact minimal et contrôlable sur le code de l'application.

DESCRIPTION DE KABOOM

Kaboom est une boîte à outil pour la pratique de la programmation à la volée en C/C++. Elle permet de développer incrémentalement une application alors que celle-ci s'exécute.

Kaboom propose les fonctions suivantes :

- mise à jour de l'application lors du changement du code d'une fonction ou d'une méthode ;
- éditeur intégré et widgets pour la manipulation directe des constantes sans compilation (Figure 2) ;
- contrôle explicite de la portée de la mise à jour vis à vis de l'édition de liens ;
- contrôle explicite des points de mise à jours pour éviter les interférences avec les algorithmes ;
- fonctions exécutées lors de mises à jour ;

Figure 2. Utilisation du mode "live" de Kaboom. A gauche : l'éditeur intégré permet de changer dynamiquement les variables à l'aide de widgets dédiés. A droite : l'application en cours d'exécution.

— compatibilité avec les IDEs par l'usage de messages d'erreurs au même format que les compilateurs usuels ;
— récupération d'erreur de segmentation.

L'une des règles guidant le design de Kaboom est de laisser le développeur choisir les zones à mettre à jour ce qui lui permet de contrôler l'impacte de la programmation à la volée sur les performances de son application. Pour cela nous utilisons le principe des annotations du C/C++ . Trois annotations sont définies par Kaboom : *codegen_dynamic* ; *codegen_moduledecl* ; *codegen_onreload*. Elle sont ignorées par les autres compilateurs et le code annoté reste totalement compilable avec ceux-ci. Par contre lorsque l'application est compilée avec Kaboom ces annotations activent le support de la mise à jour du code pendant l'exécution.

En plus de la recompilation à la volée et de la manipulation directe des variables il est possible d'interagir avec l'application en utilisant le code source comme un langage de script. Pour cela on utilise l'annotation *codegen_onreload*. Une fonction ainsi annoté va être exécutées à chaque mise à jour du code source, ce qui revient à disposer d'une boucle REPL [3]. Le listing 1 illustre comment cette fonction peut être utilisée pour interroger l'état de l'application et pour ajouter dynamiquement des objets à une GUI.

```
1 codegen_onreload
2 void myscripting(int param){
3   printf("Obj Count: %d\n", getNumObject()
      );
4   app().addWidget(new QButton("Button OK")
      );
5 }
```

Listing 1. Utilisation de l'annotation *codegen_onreload* à la manière d'une console de script

OBJECTIFS DE LA DÉMO

Dans le cadre des démos IHM 2014 nous souhaitons présenter la première version publique de Kaboom. Nous pouvons illuster les usages de Kaboom sur différents examples en liens avec des tâches de développement que nous réalisons dans nos laboratoires. Par exemple :

— développement d'applications graphique (OpenGL, QT, etc...) ;
— développement d'applications interactive et plus particulièrement écriture d'automate d'interation ;
— utilisation de la boucle REPL et donc C/C++ comme un langage de script dynamique et facilement intégrable à une application ;
— utilisation de la programmation à la volée pour l'enseignement de la programmation à la fois par l'enseignant et par les étudiants.

En outre, Kaboom est un travail en cours auquel nous souhaitons ajouter les fonctions suivantes :

— la visualisation de l'historique de la session à la manière de [4] ;
— la modification à la volée des classes ;
— le remplacement de l'éditeur intégré par un éditeur externe à la manière de GNU Rocket [1] ;
— la manipulation directe d'une instance d'une classe à la manière de ce qu'on fait pour les constantes ;
— la sélection dans la fenêtre de l'application des des objets à manipuler dans le code source ;
— la programmation de machine à états et d'automates d'interactions. En s'appuyant sur l'analyse de [5] nous souhaitons pouvoir manipuler un même automate au travers d'une représentations graphique et d'un code source, les deux représentations étant interactives et synchronisées.

Dans cet optique nous espérons pouvoir échanger lors des séances démos sur limites et les avantages de la programmation à la volée à la fois comme une forme d'interaction riche et expressive et interactive, et plus concrètement comme pratique de développement généralisable et/ou à généraliser.

BIBLIOGRAPHIE

1. Gnu rocket. **http://rocket.sourceforge.net/**.
2. Measuring programming language popularity. **http://en.wikipedia.org/wiki/Measuring_programming_language_popularity** and **http://langpop.com/**.
3. Read–eval–print loop. **http://en.wikipedia.org/wiki/REPL**.
4. Chevalier F., Dragicevic P., Bezerianos A. & Fekete J.-D. Using text animated transitions to support navigation in document histories. In *CHI* (2010), 683–692.
5. Conversy S. Existe-t-il une différence entre langages visuels et textuels en termes de perception ? In *IHM 2013, 25ème conférence francophone sur l'Interaction Homme-Machine*, ACM (2013).
6. Ungar D. Self and self: Principles, meta-principles and personal lesson. Talk at Stanford Colloquim: **http://youtu.be/3ka4KY7TMTU**, 2009.
7. Ungar D. & Smith R. B. Self. In Proc. *HOPL III*, ACM (2007), 9–1–9–50.
8. Victor B. Inventing on principle. **http://worrydream.com**.