ENSISA 2012–2013

Projet Framework

Modélisation et méta-modélisation de systèmes d'inférences flous

Nicolas DEVENET & Valériane JEAN

Introduction

Une des principales préoccupations des éditeurs de logiciels est de générer du code rapidement et de ne pas avoir à refaire ce qui a déjà été fait auparavant. Un éditeur peut alors utiliser des modèles génériques qui serviront de base à l'ensemble de nouveaux projets. Ces modèles sont appelés *framework* : c'est une application semi-finie fournissant des cadres génériques spécialisables pour la réalisation de parties d'application ou d'applications.

L'objectif du projet est de réaliser un *framework* générique pour les systèmes d'inférence flous en langage C++, et de l'utiliser sur un exemple concret : « le calcul d'un pourboire ».

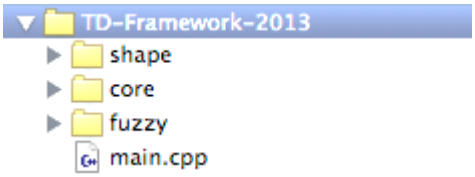
Le *framework* attendu se doit d'être flexible, extensible et fournir des opérateurs polymorphes (l'influence de la nature de chaque opérateur doit pouvoir être étudiée sans pour autant refaire un nouveau système).

Mise en œuvre

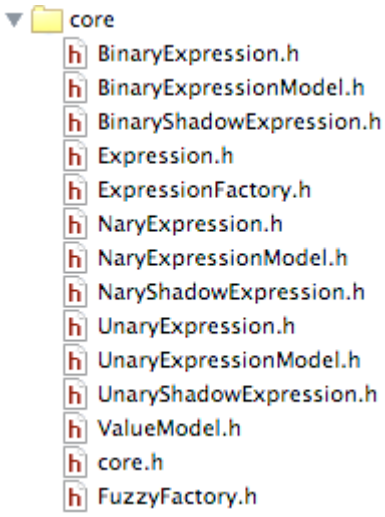
Pour rendre le *framework* flexible et extensible, le projet a été séparé en trois parties principales :

1. partie « core » : comprend l'ensemble des différentes expressions floues (unaire, binaire, naire) avec lesquelles les calculs sont réalisés ;
2. partie « fuzzy » : représente le modèle flou qui implémente la partie « core » ;
3. partie « shape » : contient les formes et intervalles nécessaires aux calculs.

Organisation générale du projet :



Partie « core »

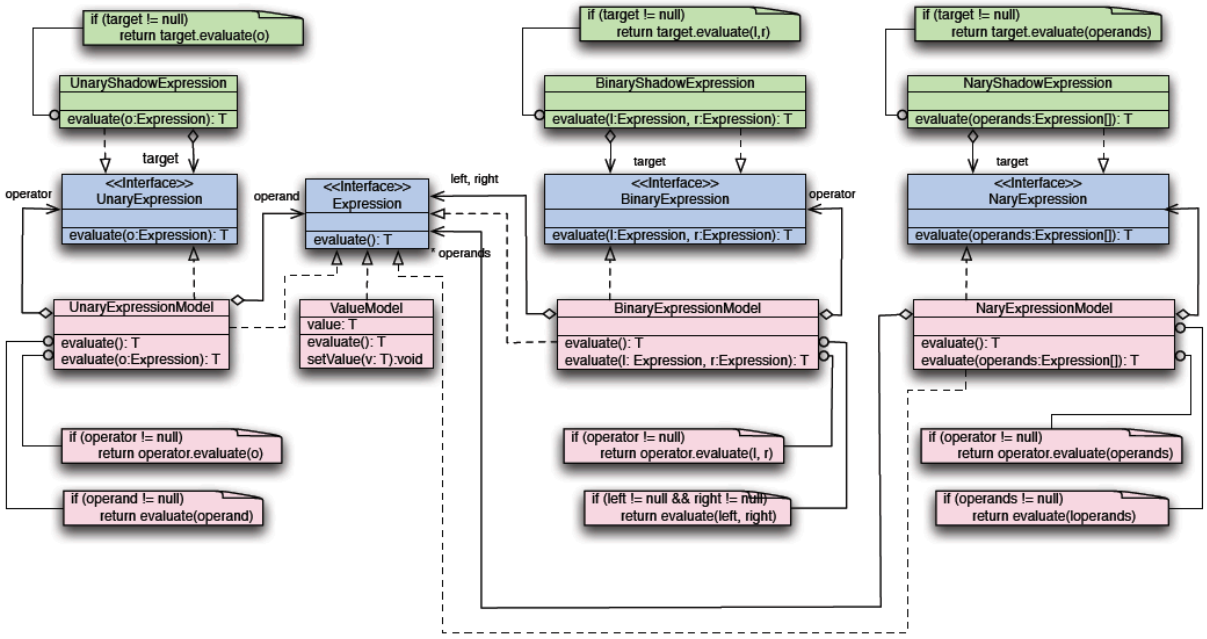


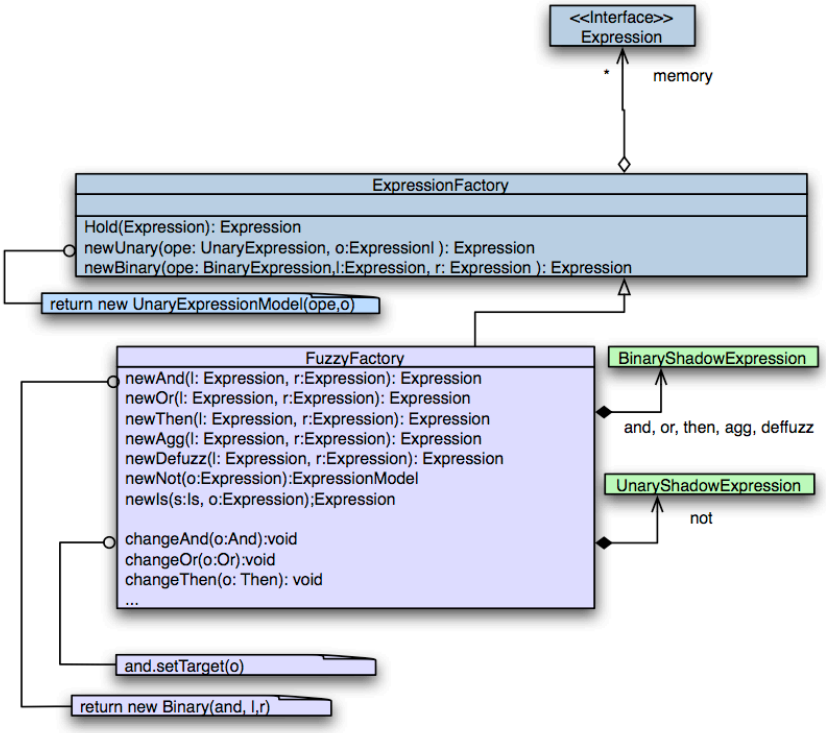
Cette partie doit fournir un système d'expressions logiques. Nous avons tout d'abord créé la classe des expressions unaires (leurs modèles permettant de définir l'opérateur pour évaluer l'expression), avant d'étendre les expressions à plusieurs expressions.

Les classes *shadow* permettent l'évolution polymorphique d'opérateurs. Elles jouent ainsi le rôle d'opérateurs en maintenant un lien vers l'opérateur réel. Cela permet de pouvoir modifier à tout moment l'opérateur initial en le remplacent par un autre. Ces classes ont été nécessaires pour fabriquer les *factories*.

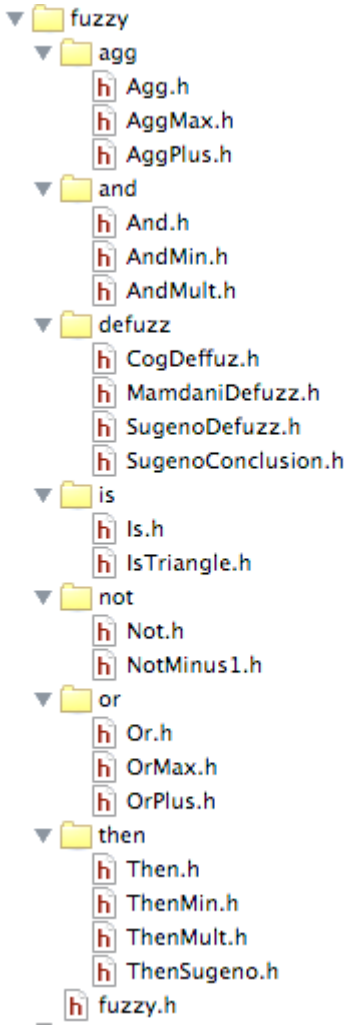
Pour améliorer le *framework* et répondre au cahier des charges, nous avons ensuite mis en place des *factories*. Une *factory* prend en charge le processus de cronstruction, et permet ainsi de masquer la complexité de l'assemblage. Elle détient aussi la sémentique d'assemblage du modèle.
C'est là qu'interviennent les classes *shadow* qui permettent de modifier le lien vers l'opérateur utilisé, sans devoir en redéfinir une nouvelle.

Une *factory* est nécessaire pour fabriquer les expressions (contenues dans la partie « core »), et une autre est nécessaire pour fabriquer les opérateurs de logique floue (contenus dans la partie « fuzzy »).





Partie « fuzzy »

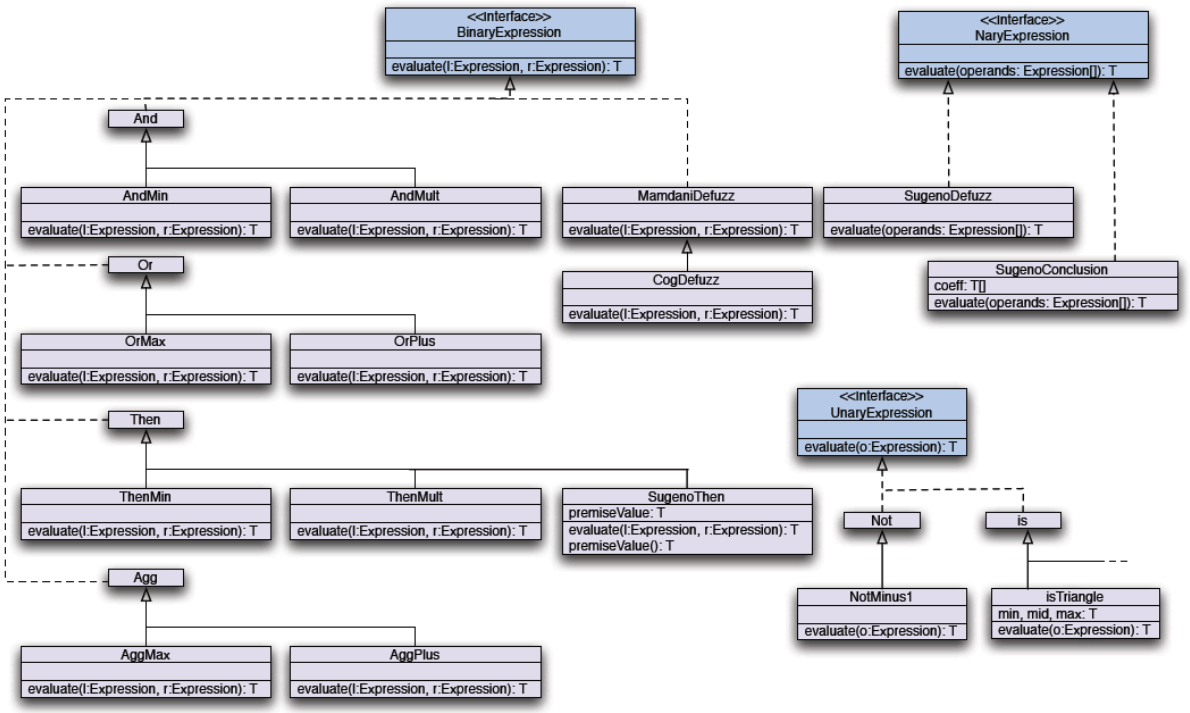


Cette partie permet d'implémenter la partie « core ».

Il a donc fallu définir les expressions caractérisant la logique floue. Pour une meilleure clarté, nous avons regroupé les opérateurs entre type.

Nous avons ensuite mis en place l'opérateur de défuzzification par centre de gravité : les classes « CogDefuzz » et « Mamdani Defuzz » se chargent du travail.
Les formes utilisées permettent de calculer leur centre de gravité ; nous obtenons ainsi un ensemble de points qui seront utilisés pour la défuzzification.

La dernière étape a été de construire l'opérateur de défuzzification de type Sugeno. La différence entre ces deux opérateurs réside dans l'implication et la méthode de défuzzification. « SugenoConclusion » permet d'affecter un coefficient à chaque valeur d'entrée et ainsi de donner plus ou moins d'importance à une entrée.
« SugenoThen » permet d'effectuer les opérations suivantes : il évalue d'abord une expression entrée en paramètre et la stocke en mémoire, évalue l'expression suivante qui n'est autre que « SugenoConclusion » et multiplie les deux résultats. Nous avons le même comportement qu'un « ThenMult » en gardant en mémoire le premier paramètre.



Mise en pratique

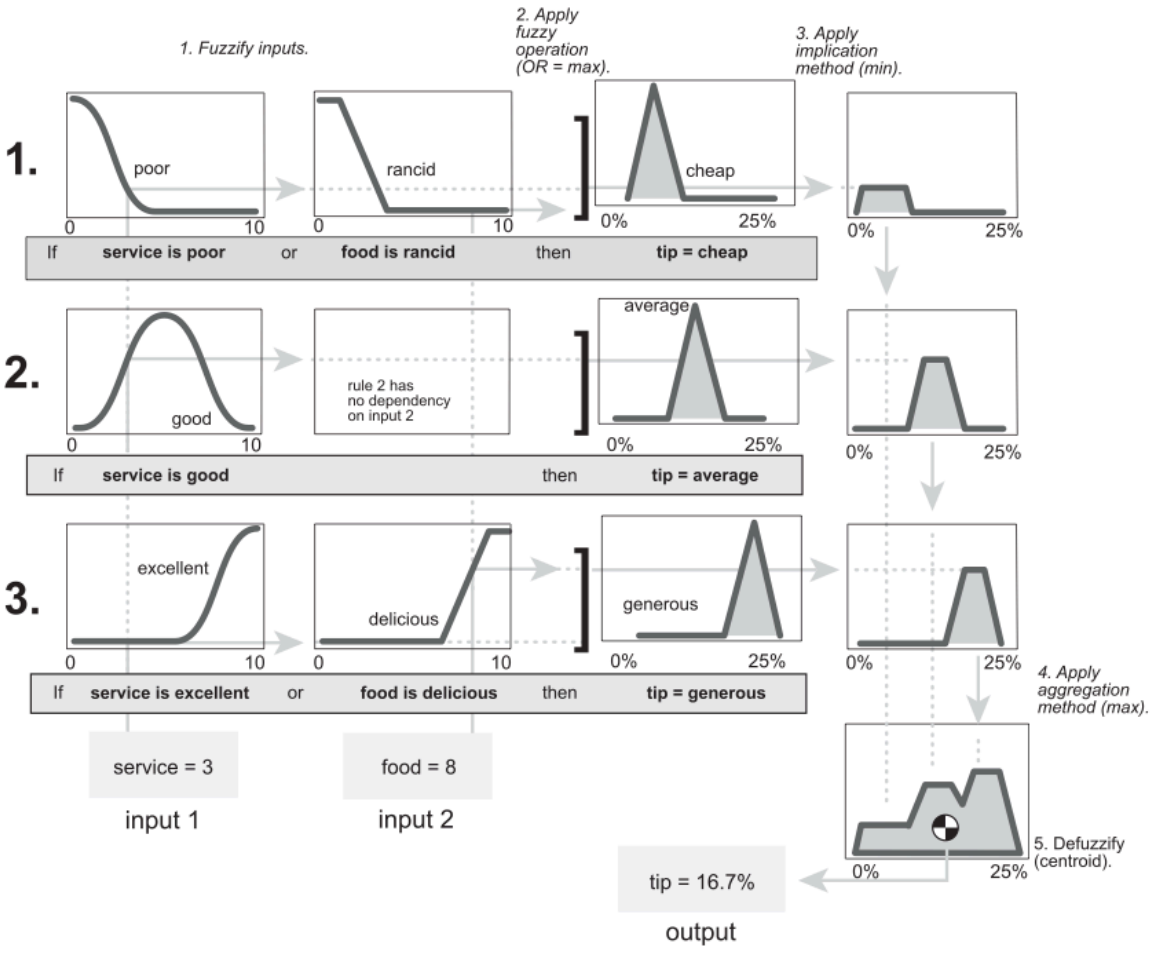
Une fois le *framework* fait, il fallait l'utiliser avec un exemple concret. Cet exemple devait permettre de calculer, en utilisant donc les règles des systèmes flous, le calcul d'un pourboire.

La principale difficulté a résidé dans le choix des intervalles et formes pour régler le système de logique floue. Il a ensuite fallu traduire les règles avec nos expressions et opérateurs.

Exemple de traduction des deux premières règles :

```
f.newAgg(
        f.newThen(
                f.newOr(
                        f.newIs(&service, &poor),
                        f.newIs(&food, &rancid)
                        ),
                f.newIs(&tips, &cheap)
                ),
        f.newThen(
                f.newIs(&service, &good),
                f.newIs(&tips, &average)
                )
        )
```

Nous avons évidemment utilisé une *factory* pour tirer profit de ses avantages.



En utilisant notre *framework* et avec notre configuration, nous obtenons un score proche de l'exemple avec l'utilisation de l'opérateur Mamdani de défuzzification. Ce score semble deux fois plus petit lorsque nous utilisons l'opérateur Sugeno deféuzzification.
Cela doit provenir des paramètres que nous avons utilisés pour « configurer » l'exemple.

All Output

```
/* mamdani */

service: 3
food: 8
tips > 15.15

service: 10
food: 12
tips > 23

service: -4
food: -2
tips > 6.58333

service:
```

All Output

```
/* sugeno */

service: 3
food: 8
tips > 7.11111

service:
```

Conclusion

La construction d'un *framework* générique pour les systèmes d'inférence flous s'est fait pas à pas, en testant au fur et à mesure les différents composants créés.
Le *framework* ainsi construit permet de le faire évoluer selon les besoins futurs.

L'utilisation du *framework* pour le calcul d'un pourboire a permis de mettre en pratique ce que nous avions fait.