Rapport Fuzzy Framework

Objectif : Etablir un Framework générique pour les systèmes d'inférence flous

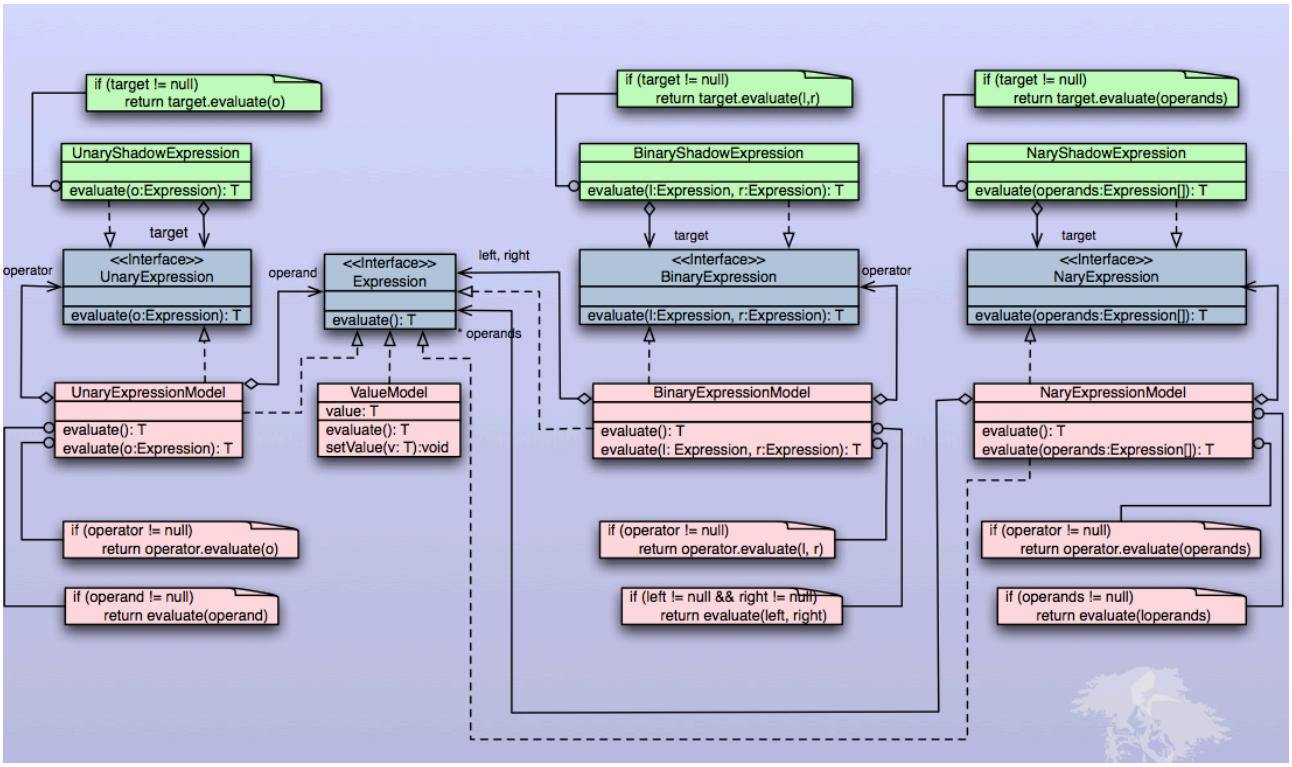
Dans le cadre du cours sur les frameworks (encadré par M. Perronne), nous avons pour objectif de réaliser un framework de calcul générique en C++, nous l'utiliserons par la suite dans le domaine de la logique floue.

1. Le «Core» qui permet de manipuler des expressions et des opérateurs qui nous permettrons d'effectuer nos calculs
2. «Fuzzy» quand à lui est l'utilisation de la logique floue appliquée au «Core»
   Il permet de définir l'ensemble des éléments de la logique floue afin de pouvoir l'utiliser aisément par la suite directement dans notre main.cpp

Nous verrons ensemble chaque partie et leur fonctionnement.

Le métamodèle conceptuel : «Core»

Le métamodèle core permet de créer et manipuler des expressions peu importe leur type.
Dans notre cas il sera utilisé pour le domaine des mathématiques, plus particulièrement de la logique floue.



Dans le «Core» nous manipulons des pointeurs d'«Expression».

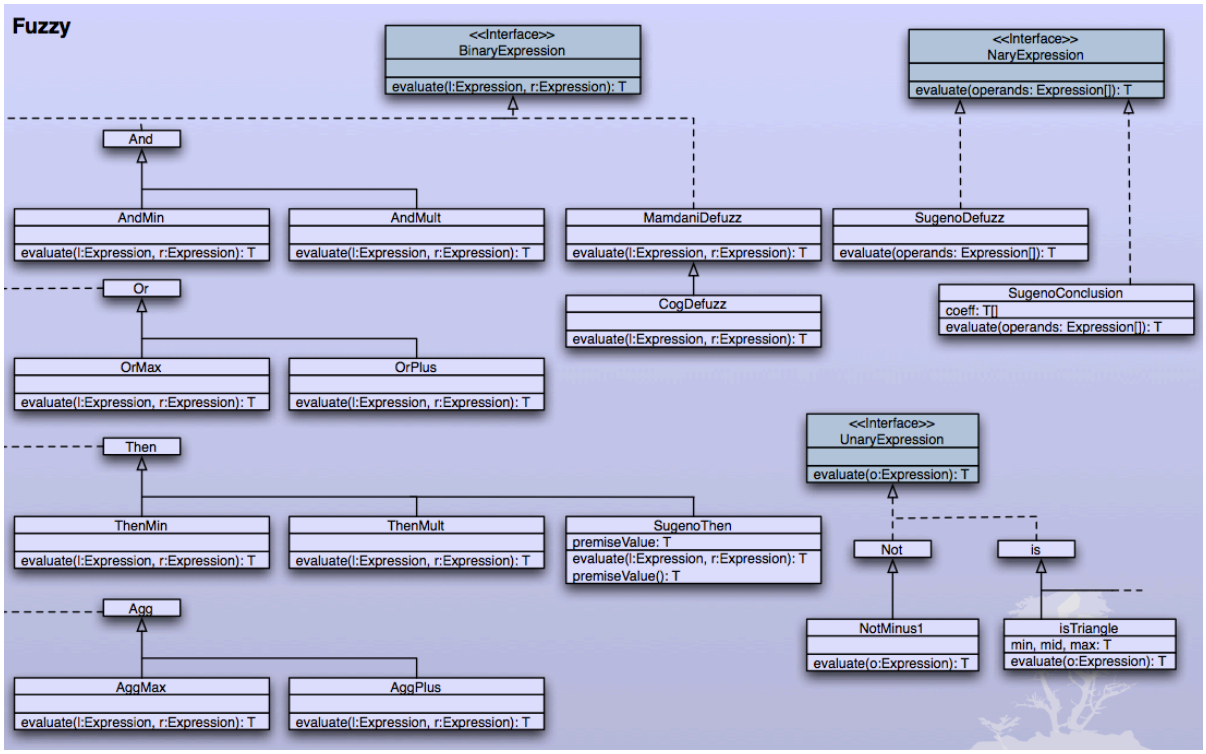
Les expressions sont des objets qui possèdent qu'une seule méthode : Evaluate(). On compose les Expressions comme on composerait un arbre, les ValueModel étant les feuilles de l'arbre (C'est la seul classe qui comporte un attribut utile (de type T)). Ainsi, il existe différentes sous-classes d'Expression :

1. Unaire (UnaryExpression), elle évalue une «Expression» à une seule opérande (non, carré...).
2. Binaire (BinaryExpression), elle évalue une «Expression» à deux opérandes (x + y, max (x, y)...).
3. N-aire (NaryExpression), elle évalue une «Expression» à n opérande grace à une liste (sugeno...).

Par la suite nous avons fait évoluer le Core en y ajoutant les «Shadow» :
Elle se contente de recopier une «Expression» et modifier son opérateur grâce à un setTarget(), elle a le même comportement qu'une «Expression». On pourra donc recopier une nouvelle «Expression» sans modifier l'objet «Shadow» au final.

Une «ExpressionFactory» qui permettra d'utiliser le concept de «Factories» pour créer ou modifier des arbres d'expressions.

Le métamodèle complété «Fuzzy»



Le métamodèle «Fuzzy» permet de compléter le «Core» en créant tous les opérateurs nécessaires à la manipulation de système flou.

1. Pour les unaires : not,is.....
2. Pour les binaires : and, ou, implique et agrégation
3. Pour les n-aires tous ce qui concerne Sugeno

Comme pour la partie «Core» une «Factory» est mise en place pour faciliter la création des arbres d'évaluation, les «ValueModel» étant toujours en bout d'arbre, la tête de l'arbre correspondant quant à elle au résultat final de l'opération.

Le cas Mamandi

```
core::Expression<double> *res =
    f.NewAgg(
        f.NewAgg(
            f.NewThen(
                f.NewOr(
                    f.NewIs(&poor,&service), //On affirme que le service est mauvais
                    f.NewIs(&rancid,&food)   //OU que la nourriture est mauvaise
                ),
                f.NewIs(&cheap,&tips)        //DONC le pourboir sera médiocre
            ),f.NewThen(
                f.NewIs(&good,&service),     //De la même manière on affirme que le service est bon
                f.NewIs(&average,&tips)      //DONC le pouboir sera moyen
            )

        ),f.NewThen(
            f.NewOr(
                f.NewIs(&excellent,&service),//Le service est parfait
                f.NewIs(&delicious,&food)    //OU que la nourriture est delicieuse
            ),
            f.NewIs(&generous,&tips)         //DONC on sera très généreux !!
        )
    );

core::Expression<double> *defuzz = f.NewMamdani(&tips, res); //On cherche le centre de
                                                             //gravité du résultat
```

Ce procédé défuzzifie par la méthode du centre de gravité, cela va revenir à faire une agrégation de toutes les règles pour au final en extraire le centre.

Pour cela des «Shape» sont créées par la classe de défuzzification, les «Shape» étant un échantillonnage de la forme de l'entrée(gaussien,triangle,trapèze,créneau...), les «Shape» sont donc fabriquées en évaluant point à point l'«Expression».

Les centres de gravité seront ensuite calculés avec une formule de mathématiques classique.

Le cas Sugeno :

```
fuzzy::SugenoThen<double> opThenSugeno;
f.ChangeThen(&opThenSugeno); //On redéfini THEN pour Sugeno
                             //Evalue et memorise left
```

```
core::Expression<double> *res =
    f.NewAgg(
        f.NewAgg(
            f.NewThen(
                f.NewOr(
                    f.NewIs(&poor,&service), //On affirme que le service est mauvais
                    f.NewIs(&rancid,&food)   //OU que la nourriture est mauvaise
                ),
                f.NewIs(&cheap,&tips)        //DONC le pourboir sera médiocre
            ),f.NewThen(
                f.NewIs(&good,&service),     //De la même manière on affirme que le service est bon
                f.NewIs(&average,&tips)      //DONC le pouboir sera moyen
            )

        ),f.NewThen(
            f.NewOr(
                f.NewIs(&excellent,&service),//Le service est parfait
                f.NewIs(&delicious,&food)    //OU que la nourriture est delicieuse
            ),
            f.NewIs(&generous,&tips)         //DONC on sera très généreux !!
        )
    );

core::Expression<double> *defuzz = f.NewMamdani(&tips, res); //On cherche le centre de
                                                             //gravité du résultat
```

Il met en place la défuzzification par Sugeno, elle doit pouvoir admettre n paramètres comme Sugeno («NaryExpression») le préconise, pour cela il a fallu redéfinir deux opérateurs et une conclusion qui permet de coéfficienter la valeur d'entrée.

La classe «SugenoThen», permet d'évaluer une expression face à un «SugenoConclusion», elle agit en quelque sorte comme n'importe quel «Then» à la premiseValue près.

Vient ensuite la classe «SugenoDefuzz» qui va évaluer un à un les «SugenoThen» pour ensuite sortir la valeur finale grâce à la méthode Sugeno.

$$\text{Final Output} = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i}$$

Grégory GABUTTO
Clément RAUSSIN
2A Informatique & Réseaux
ENSISA 2013