Modélisation & Framework

Création d’un Framework de logique floue en C++

Sommaire

[Objectifs 2](#_Toc358054096)

[Conception 2](#_Toc358054097)

[Réalisation 2](#_Toc358054098)

[Outils de développement 2](#_Toc358054099)

[Implémentation du concept d’expressions 2](#_Toc358054100)

[Extension du modèle d’expressions appliqué à la sémantique floue 3](#_Toc358054101)

[Opérateurs 3](#_Toc358054102)

[Evaluation d’expressions 3](#_Toc358054103)

[Défuzzification 3](#_Toc358054104)

[Création de systèmes avec le pattern Factory 3](#_Toc358054105)

[Principe 3](#_Toc358054106)

[Concept de « ShadowExpression » 3](#_Toc358054107)

[Cas d’utilisation 4](#_Toc358054108)

[Pistes d’améliorations 5](#_Toc358054109)

# Objectifs

Le but de ce projet est de réaliser un Framework générique adapté à la résolution de systèmes d’inférence flous. Le Framework se doit d’être portable, puissant, adapté tout en permettant la manipulation de nouveaux aspects et extensions de la logique floue.

Les pré-requis essentiels de ce Framework sont la flexibilité et l’extensibilité tout en fournissant des opérateurs polymorphes.

# Conception

Pour réaliser les objectifs, nous avons suivi une démarche de développement par les modèles. En partant de la structure de la logique floue, un méta-modèle a été conçu puis raffiné et étendu pour définir la sémantique à atteindre.

Nous avons tiré des concepts généraux des variables floues (fonctions d’appartenance et opérateurs) et différencié plusieurs types de systèmes flous (Mamdani et Sugeno). Les différents types de systèmes reposent sur les mêmes variables et seule la méthode de défuzzification varie.

Le concept principal est celui des expressions : les règles floues sont des expressions floues semblables au modèle des expressions arithmétiques.

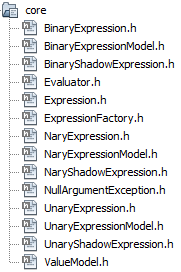
# Réalisation

## Outils de développement

Nous avons développé dans des environnements similaires et les mêmes logiciels afin de réaliser chacun le projet dans son intégralité. Cela a permis à chacun d’apprécier l’intégralité des aspects du développement du projet en permettant une entraide efficace.

Nous avons choisi de travailler sous *NetBeans 7.2.1* en utilisant le compilateur de *Cygwin*. Pour partager et archiver facilement le travail et les avancés, nous avons utilisé *Git-Hub* en mode privé.

## Implémentation du concept d’expressions

Nous avons défini une entité appelée expression le plus généralement possible : elle est définie par un template C++ définissant un contrat : elle peut être évaluée et retourner ainsi sa valeur. L’expression a été étendue plus spécifiquement en expression unaire, binaire et n-aire.

Différents types de modèles d’expressions, appliquant ce contrat, on été définies. On parle ainsi d’*UnaryExpressionModel*, *BinaryExpressionModel*, *NaryExpressionModel*. Pour définir les entrées des systèmes, une dernière classe de modèle d’expression a été nécessaire et encapsule les valeurs brutes : *ValueModel*.

## Extension du modèle d’expressions appliqué à la sémantique floue

### Opérateurs

Le concept des opérateurs a été adapté à la logique floue. Des opérateurs, basés sur les différents types d’expression (unaires, binaires et n-aires) ont été implémentés. On retrouve les opérateurs courants tels que les *And*, *Or*, *Not* et *Then*, ainsi que les opérateurs d’appartenance *Is* (étendus suivant plusieurs formes classiques : trapèzes, triangle, gaussiennes). Pour agréger des ensembles de règles floues, des opérateurs d’agrégation *Agg* ont été nécessaires.

Chacun de ses opérateurs définit un comportement spécifique lui étant propre et changeant la façon d’évaluer l’expression.

### Evaluation d’expressions

Afin d’avoir une évaluation échantillonnée d’une expression, un évaluateur a été développé et intégré au cœur du projet. Celui-ci évalue une expression sur une plage de données et permet une utilisation ultérieure par les opérateurs de défuzzification.

### Défuzzification

Pour manipuler les deux types principaux de systèmes flous identifiés (Mamdani et Sugeno) deux opérateurs de défuzzification ont été réalisés, implémentant dés résolutions différentes.

Dans le cas de Sugeno, un *Then* particulier a été mis en place afin de pouvoir accéder aux prémisses (valeurs calculés avant l’application du *Then*).

## Création de systèmes avec le pattern Factory

### Principe

La conception du système de règles floues étant une opération laborieuse, répétitive et sujette aux erreurs d’inattentions, une *Factory* a été mise en place. Celle-ci s’initialise avec les opérateurs utilisés dans les règles (spécifications des *And*, *Or*, *Not*, *Agg* et *Then*) ainsi que les opérateurs de défuzzification.

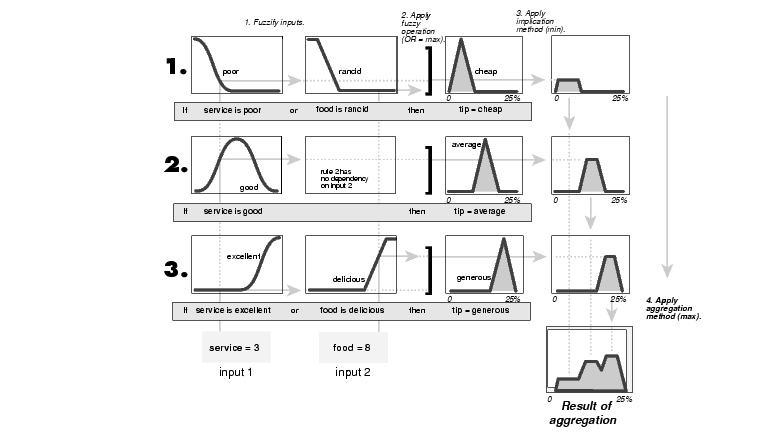
La création de système est ainsi facilitée et les arbres de règles floues sont plus simples à concevoir.

### Concept de « ShadowExpression »

Pour permettre un changement d’opérateur à la volée dans les systèmes générés avec la Factory, des expressions d’encapsulation ont été crées. Des *ShadowExpressions* ont été réalisés pour chaque type de modèle d’*Expression* et miment le comportement d’évaluation de l’*Expression* qu’elles encapsulent. Elles agissent comme des proxies et le changement de l’opérateur encapsulé change le comportement de la *ShadowExpression*.

# Cas d’utilisation

Nous avons utilisé notre Framework dans le problème classique du pourboire :

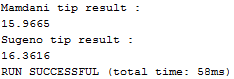


La configuration des paramètres en entrée ainsi que les formes utilisées (particulièrement les gaussiennes) sont des valeurs proches de l’exemple donné, mais par manque d’informations précise seul des approximations sont utilisées.

Nous avons utilisé les paramètres suivants :

* **Gaussiennes :**
* **Tips :** 0 à 26
* **Service :** 0 à 10
* **Food :** 0 à 10
* **Echantillonnage d’évaluation :** 0 à 26 (pas : 0.1)
* **Coefficients Sugeno :** [1.6 ; 2.1]

Nous obtenons le résultat suivant :



Le résultat escompté étant de 16,3. Les erreurs constatées sont faibles et dues aux paramètres définis par approximation.

# Pistes d’améliorations

Un Framework évolutif a été créé. Il est donc possible de l’étendre pour lui ajouter des fonctionnalités. On peut penser notamment à ajouter d’autres méthodes de défuzzification, d’autres formes voir d’autres opérateurs.

Il est aussi possible d’utiliser le Framework dans des interfaces utilisateur en console ou graphiques pour faciliter la prise en main et son utilisation par des non développeurs.

Pour obtenir des systèmes plus fidèles, il serait intéressant d’imaginer un modélisateur qui réaliserai une série d’approximations afin de trouver les paramètres idéaux du système, voir même de permettre l’utilisation de réseaux de neurones.