Outil de calcul d'indicateurs d'évaluation de modèles de simulations par la logique floue

Amine Ben Hammou, Nabil Zaini

Institut **S**upérieur d'**I**nformatique, de
Modélisation et de leurs **A**pplications

Campus des Cézeaux

24 avenue des Landais

BP 10125

63173 AUBIERE Cedex

UREP - INRA

5 chemin de Beaulieu 63 039

Clermont-Ferrand

Rapport d'ingénieur

Projet de 3ème année

F2 : Filière Génie Logiciel et Systèmes Informatiques

Outil de calcul d'indicateurs d'évaluation de modèles de simulations par la logique floue

Présenté par :

Amine BEN HAMMOU (F2)

Nabil ZAINI (F2)

Membres de jurys :

Claude MAZEL (ISIMA)

Raphael MARTIN (INRA)

Gianni BELLOCCHI (INRA)

Date de la soutenance : **09 / 03 / 2015**

Remerciements

A l'issue de ce projet, nous tenons tout d'abord à remercier Monsieur MARTIN Raphael et Monsieur BELLOCCHI Gianni pour le projet qui nous a été confié, et pour leurs précieux conseils et leur aide durant toute la période de notre projet ainsi que pour leurs bonnes explications qui nous ont éclairé le chemin et leurs collaborations avec nous dans l'accomplissement de ce projet.

Nos remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre projet en acceptant d'examiner notre travail, et de l'enrichir par leurs propositions. Nous ne manquerons pas d'adresser finalement nos remerciements à tous les employés de l'Institut national de la recherche agronomique et à toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Table des figures et illustrations

Figure 1: Objectif du projet 3
Figure 2 : Diagramme de Gantt 4
Figure 3 : Relation de calcul de la valeur floue d'un indice 6
Figure 4 : Exemple d'agrégation de deux indices 7
Figure 5 : Table des calculs (exemple de deux indices) 7
Figure 6 : Diagramme des cas d'utilisation 8
Figure 7 : Maquette de l'écran principal de l'outil 9
Figure 8 : Maquette présentant le menu Fichier 9
Figure 9 : Maquette du formulaire d'ajout d'un indice 10
Figure 10 : Maquette du formulaire d'ajout d'un module 10
Figure 11 : Diagramme de classes de la bibliothèque centrale 13
Figure 12 : Implémentation de Application::updateValues 15
Figure 13 : Implémentation de Index::updateValues 16
Figure 14 : fichier MiniButton.qml 18
Figure 15 : Les types de block 19
Figure 16 : La fonction Box.draw du javascript 20
Figure 17 : Résultat final 21

Résumé

L’évaluation rigoureuse des modèles agro-écologiques favorise leur acceptation et leur adoption par une large communauté d’utilisateurs. Pour cela, un inventaire d’indices est disponible. L’agrégation de plusieurs indices à travers la logique floue permet la formulation de critères flexibles et l’élaboration de concepts élargis d’évaluation, tout en intégrant des éléments différents tels que la comparaison entre données simulées et observées, la complexité des modèles et la stabilité des résultats de la modélisation dans des conditions diverses.

Ce document présente une application qu'on a développé en utilisant C++ et Qt. Cette application comporte une partie qui fait les calculs codée en C++, une deuxième partie qui correspond à l’interface graphique est développée en utilisant les composants QML du Qt qui sont des composants flexibles et plus légers que ceux codés en C++. La troisième partie constitue les tests unitaires de toutes les fonctionnalités de notre application.

Mots clés : indicateurs d’évaluation, modèles de simulation, logique floue, C++, Qt, QML, XML

Abstract

Rigorous evaluation of agro-ecological models supports acceptance and adoption by a large community of users. The aggregation of many indices using fuzzy logic allows the formulation of flexible criteria and the development of broader concepts evaluation with the ability to compare simulated and observed data, to handle models complexity and to ensure the stability of results under different conditions. An application with a simple user interface is needed to create and aggregate indices, and it should be able to save modules and use them in different indicators.

We developed an application using C++ and Qt framework. C++ was used to create the core library of the application and makes it able to handle complex models and to compute values fast. Instead of C++ coded widgets, the QML components of Qt were used to construct the user interface because they are cross platform and easily customizable. These components used JavaScript to handle user events and invoke corresponding functions on the core library. Unit tests of all features were written in a separate module using the unit testing framework of Qt.

The resulting application allows the user to handle an indicator by adding modules and indices to it. All input forms are validated before saving and errors are reported to the user immediately. Values of modules and the indicator are updated automatically. The user can save the indicator for future usages. He can also export it as an XML, CSV file or a PNG image. Modules can also be exported as an XML file and reused to build other indicators. The hierarchical structure of the indicator is displayed as a tree in the interface with the ability to hide and show the part under any module. This tool is useful to show indicator models and the core library can be linked to database containing auto generated models and values of indices to evaluate and compare different models of indicators.

Keywords: evaluation indicators, simulation models, fuzzy logic, C++, Qt, QML, XML

Table des matières

Remerciements i

Table des figures et illustrations ii

Résumé iii

Abstract iv

Table des matières v

Table des abréviations vii

Introduction 1

Présentation du projet 2

I.1. Présentation de l'INRA 2

I.2. L'objectif du projet 2

I.3. Diagramme de Gantt 3

Analyse et conception 5

II.1. Agrégation des indicateurs par la logique floue 5

II.1.1. La logique floue : 5

II.1.2. Application de la logique floue pour le calcul des indicateurs : 5

II.2. Les cas d'utilisation 8

II.3. Conception des interfaces graphiques 9

II.4. Les outils et technologies utilisées 11

II.4.1. C++ et Qt 11

II.4.2. QML et Javascript 11

II.4.3. Git et Github : 11

II.4.4. Editeur, compilateur et SE : 11

Réalisation et résultat 12
III.1. L'architecture de l'application 12
III.2. La partie centrale 12
III.3. Les tests unitaires 17
III.4. L'interface graphique 17
III.4.1. Composantes QML customisées 17
III.4.2. La gestion des événements en Javascript 20
III.5. Le résultat 21
Conclusion 22
Webographie viii
Annexe : Guide d'installation I

Table des abréviations

ISIMA	Institut Supérieur d'Informatique de Modélisation et de leurs Applications
INRA	Institut National de la Recherche Agronomique
UREP	Unité de Recherche sur l'Écosystème Prairial
XML	eXtensible Markup Language
QML	Qt Modeling Language

Introduction

Dans le cadre de notre formation en troisième année à l'ISIMA, nous avons réalisé un projet de fin d'année intitulé « Outil de calcul d'indicateurs d'évaluation de modèles de simulations par la logique floue », durant la période du 1er Novembre au 28 février 2015 à l'INRA au sein de l'Unité de Recherche sur l'Écosystème Prairial (UREP).

Comme le montre l'intitulé du projet, notre mission était d'analyser, concevoir et développer une solution ergonomique qui simplifie l'évaluation des modèles d'indicateurs. Ces modèles vont être décrits par l'utilisateur en ajoutant un ensemble d'indices et en les regroupant dans des modules de façon hiérarchique. Le calcul des valeurs d'évaluation des différents modules et de l'indicateur final s'appuie sur la théorie de la logique floue. Finalement on doit s'assurer du fonctionnement de l'application en faisant des tests unitaires de toutes les fonctionnalités.

Ce rapport présente ce que nous avons acquis et réalisé tout au long de notre projet. Son premier objectif est de décrire les différentes phases réalisées. Il s'organise en trois parties : la première partie comporte une présentation générale de notre projet et des étapes que nous avons suivies pour l'aborder ; la deuxième partie concerne l'analyse des besoins et la conception de la solution ; Enfin, la troisième partie est consacrée à la réalisation informatique des besoins identifiés.

Présentation du projet

I.1. Présentation de l'INRA

L'Institut National de Recherche Agronomique (INRA) est un organisme public Français de recherche fondé en 1946. Cet institut produit des connaissances scientifiques et accompagne l'innovation dans les domaines de l'alimentation, de l'agriculture et de l'environnement. C'est le premier institut de recherche agronomique en Europe et le deuxième mondial. Ses activités s'organisent au sein de treize départements et sont réparties dans toute la France sur 21 centres.

Notre projet est destiné au centre de Clermont-Ferrand, et plus précisément à l'Unité de Recherche sur l'Écosystème Prairial (UREP) qui est rattachée au département "Écologie des forêts, des prairies et des milieux aquatiques". L'UREP possède une expertise internationale dans le domaine de l'écologie prairial et plus particulièrement sur l'impact du changement climatique, les bilans de gaz à effet de serre, la séquestration de carbone, les cycles carbone et azote. L'unité a développé une démarche originale en se concentrant sur les interactions plantes-sol (microorganismes) et herbe-animal, et en considérant explicitement les effets des pratiques de gestion sur la dynamique prairiale.

I.2. L'objectif du projet

L'évaluation rigoureuse des modèles agro-écologiques favorise leur acceptation et leur adoption par une large communauté d'utilisateurs. Pour cela, un inventaire d'indices est disponible. L'agrégation de plusieurs indices à travers la logique floue permet la formulation de critères flexibles et l'élaboration de concepts élargis d'évaluation, tout en intégrant des éléments différents tels que la comparaison entre données simulées et observées, la complexité des modèles et la stabilité des résultats de la modélisation dans des conditions diverses.

Le but du projet est résumé par le schéma suivant :

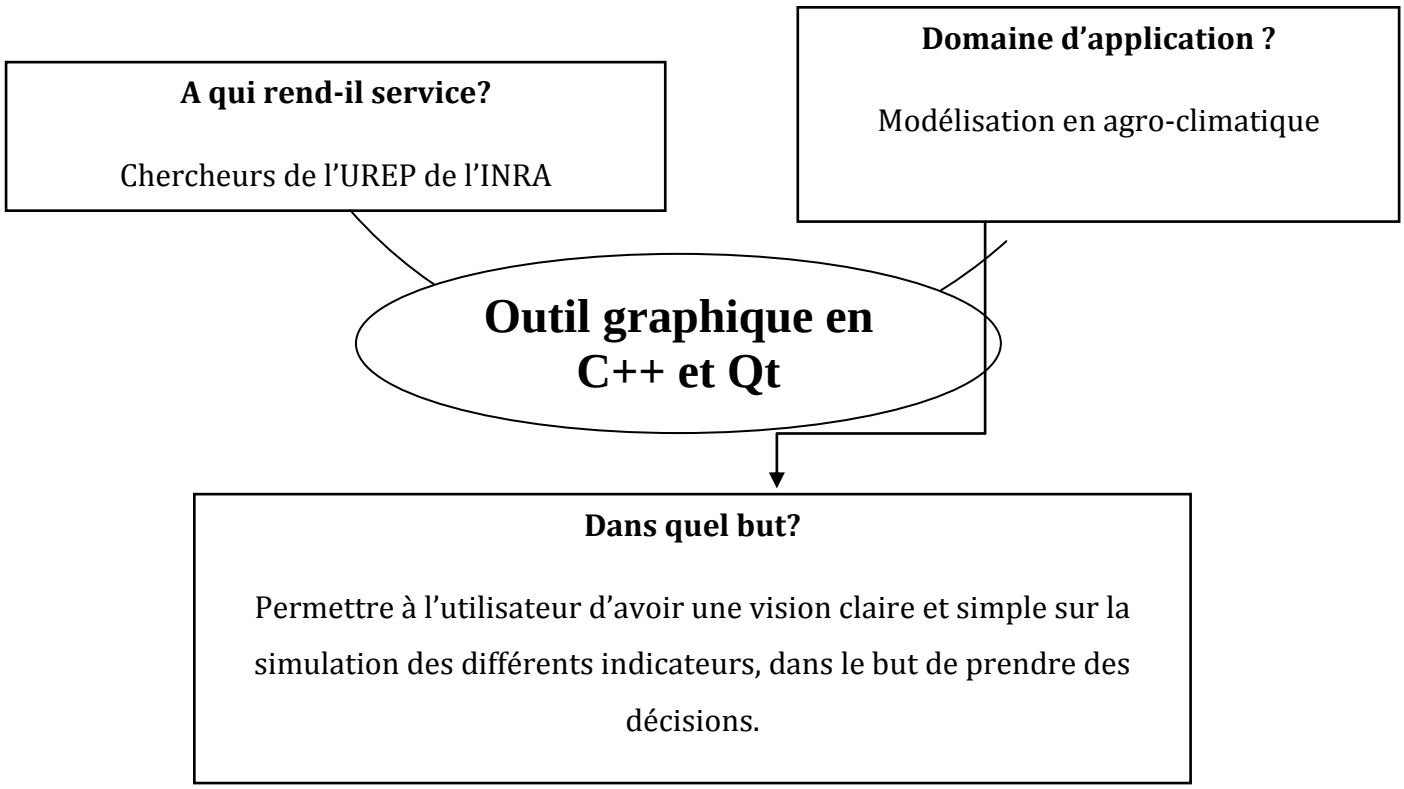

Figure 1: Objectif du projet

Cet outil graphique doit être compatible avec toute plateforme ; en particulier il doit offrir les mêmes fonctionnalités sur Linux et Windows. Il doit également offrir la possibilité d'exporter des modèles ou des parties de modèle sous format XML pour les importer dans d'autres modèles. On doit également pouvoir exporter les résultats sous formats XML, CSV et image.

I.3. Diagramme de Gantt

Afin d'avoir une bonne gestion du temps, nous avons réalisé un diagramme de Gantt au tout début du projet. Nous avons essayé de découper notre travail en différentes phases. Le diagramme de Gantt, présenté sur la figure suivante, donne une durée prévisionnelle pour chaque phase, ainsi que l'ordre dans lequel nous devions les traiter.

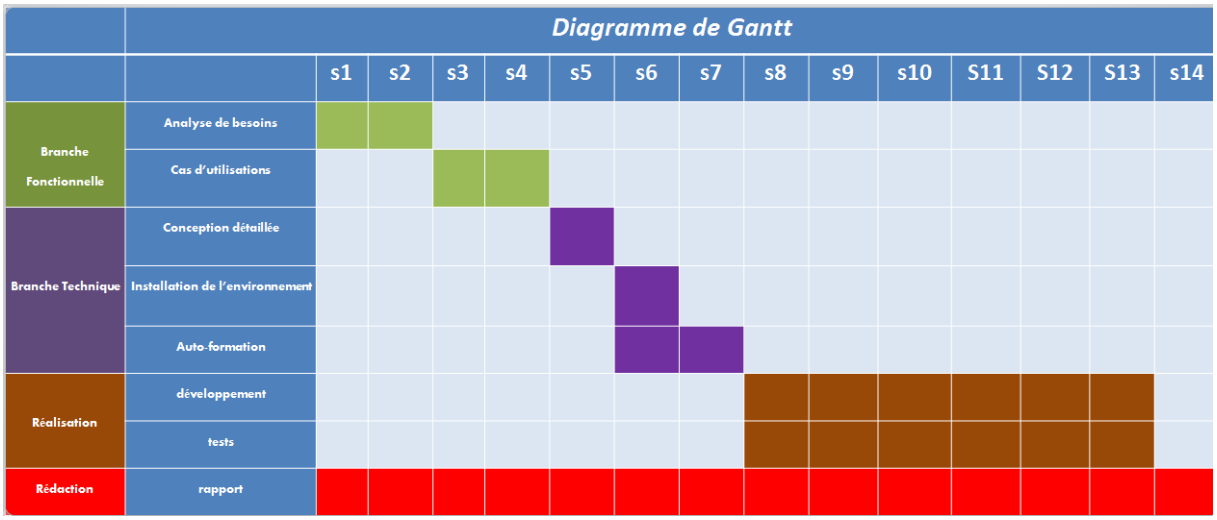

Figure 2 : Diagramme de Gantt

Nous avons identifié trois branches principales dans la réalisation de notre projet :

- la branche fonctionnelle qui s'appuie sur l'analyse et la collecte des besoins ;
- la branche technique, qui consiste à préparer l'environnement du travail en parallèle avec la découverte du Framework Qt et des autres outils qui ont été choisis au moment de l'analyse de la problématique et qui sont les mieux adaptés aux problèmes identifiés ;
- la branche de réalisation, correspondant au développement et à la réalisation des tests. Nous avons consacré la majorité de notre temps au développement.

Analyse et conception

Notre analyse du sujet a commencé par l'étude de la théorie de la logique floue, et de l'application de cette théorie dans le cadre du projet. Cette compréhension de la logique floue nous a aidés à comprendre les besoins en détails. Ainsi nous avons été capable d'établir le diagramme des cas d'utilisation et de mettre en place des prototypes de l'interface graphique de l'application.

II.1. Agrégation des indicateurs par la logique floue

II.1.1. La logique floue :

La logique floue est une extension de la logique binaire qui a été formalisée par Lotfi Zadeh en 1965. Les variables de la logique binaire peuvent prendre seulement l'un des deux états VRAI ou FAUX. Ils sont ainsi incapables de décrire les états intermédiaires où la condition n'est pas 100% vraie ou 100% fausse ; mais elle a un pourcentage de vérité. Une variable de la logique floue peut prendre n'importe quelle valeur entre 0 et 1 inclus, cette valeur reflète le niveau de sa vérité.

Afin de pouvoir profiter de ces variables floues, on doit définir les opérateurs logiques de base : **Négation**, **Conjonction** et **Disjonction.**

Soit **a** et **b** deux variables de la logique floue dont les valeurs sont respectivement **x** et **y**.

- La valeur de **NON a** est : **1 – x**
- La valeur de **a ET b** est : **min(x, y)**
- La valeur de **a OU b** est : **max(x, y)**

On voit bien que ces opérateurs s'appliquent également aux variables de la logique binaire et donnent les résultats attendus.

II.1.2. Application de la logique floue pour le calcul des indicateurs :

Notre objectif est d'agréger plusieurs indices dans un indicateur global qui donne une idée sur l'état du système. La première étape est donc d'attribuer à chaque indice de

base une valeur floue qui indique son état. Et comme les valeurs prises par les indices ne sont pas forcément incluses dans l'intervalle [0, 1], nous avons eu besoin d'une fonction qui, à une valeur prise par un indice, associe la valeur floue correspondante. Pour cela, il faut tout d'abord définir les bornes favorable et défavorable de l'indice. Ces bornes vont délimiter l'intervalle des valeurs de l'indice dont l'état est intermédiaire ou flou. A l'extérieur de cet intervalle, la valeur floue de l'indice va être 0 du côté de la borne défavorable et 1 du côté de la borne favorable.

Soient α la borne défavorable, γ la borne favorable, x la valeur prise par l'indicateur et

β = (α + γ)/2. Dans le cas où α ≤ γ, la valeur floue est donnée par la relation suivante :

$$S(x;\alpha;\gamma) = \begin{cases} 0 & x \le \alpha \\ 2\left(\dfrac{x-\alpha}{\gamma-\alpha}\right)^2 & \alpha \le x \le \beta \\ 1-2\left(\dfrac{x-\gamma}{\gamma-\alpha}\right)^2 & \beta \le x \le \gamma \\ 1 & x \ge \gamma \end{cases}$$

Figure 3 : Relation de calcul de la valeur floue d'un indice

On note cette valeur la valeur F (favorable) de cet indice, car elle indique à quel niveau la valeur prise par l'indice est favorable. De la même manière on définit la valeur U de l'indice qui est 1 – F.

Pour agréger les indices dans un indicateur, on commence par agréger les indices des modules. Ces modules peuvent être eux-mêmes agrégés dans d'autres modules. Jusqu'à un niveau supérieur où l'on agrège les modules dans l'indicateur final. Ce processus est récursif et le nombre des niveaux dépend de la modélisation de l'indicateur que l'on cherche à calculer.

Prenons par exemple le schéma suivant dans lequel on agrège les deux indice A et B dans le module M.

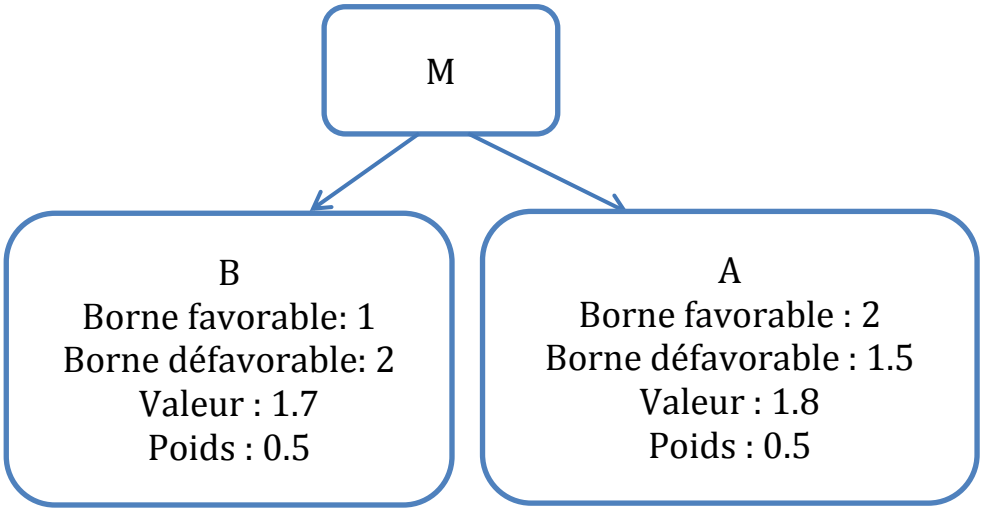

Figure 4 : Exemple d'agrégation de deux indices

Après le calcul des valeurs F et U des deux indices A et B, le calcul de la valeur du module M se fait en utilisant la table suivante :

B		A		Poids Somme des poids défavorables	Valeur de A ET B min des valeurs	Produit
F	0.18	F	0.68	0	0.18	0
U	0.82	F	0.68	0.5	0.68	0.34
F	0.18	U	0.32	0.5	0.18	0.09
U	0.82	U	0.32	1	0.32	0.32
				Somme	1.36	0.75
Résultat = Somme des produits / Somme des valeurs = 0.75 / 1.36 =						0.551471

Figure 5 : Table des calculs (exemple de deux indices)

Alors la valeur du module M est 0.551471. En prenant la borne favorable à 0 et la borne défavorable à 1 on peut utiliser les mêmes calculs pour agréger ce module dans un autre module ou indicateur.

II.2. Les cas d'utilisation

Après avoir compris la théorie des calculs, on a fait une étude des besoins et élaboré le diagramme des cas d'utilisations suivant :

Figure 6 : Diagramme des cas d'utilisation

II.3. Conception des interfaces graphiques

Avant de commencer le développement de notre outil, et comme l'ergonomie de l'application est l'un des besoins principaux du projet, nous avons mis en place des maquettes de l'interface graphique pour donner une idée du résultat final :

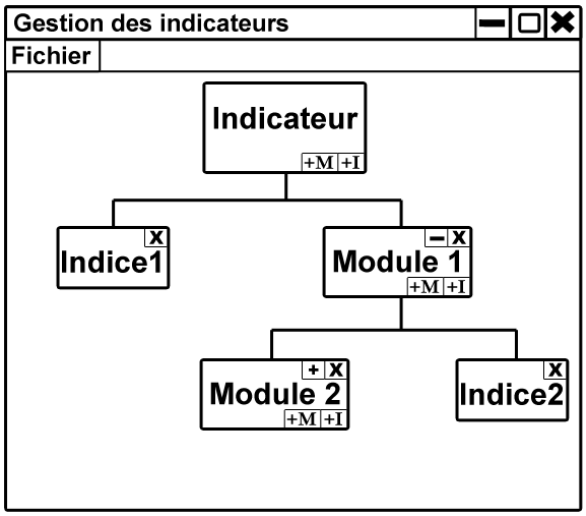

Figure 7 : Maquette de l'écran principal de l'outil

La figure ci-dessus représente la fenêtre principale de l'outil dans laquelle l'utilisateur peut construire un arbre des composantes graphiques dont la racine représente l'indicateur, les nœuds intermédiaires représentent les modules et les feuilles sont les indices.

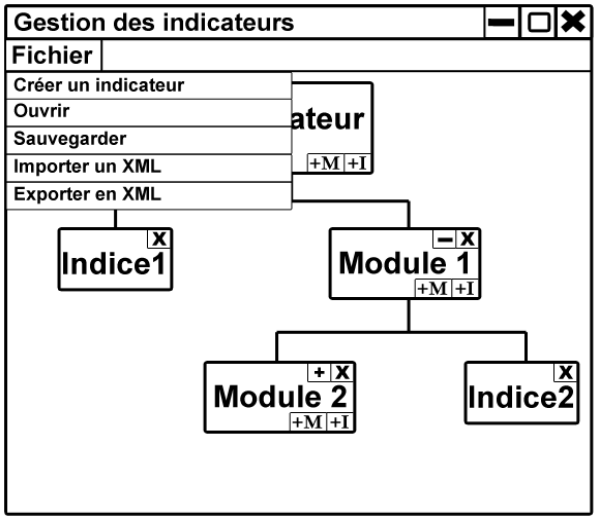

Figure 8 : Maquette présentant le menu Fichier

La figure 8 représente le menu contenant l'ensemble de fonctionnalités fournit par l'outil : La création d'un nouvel indicateur, l'ouverture d'un indicateur déjà définit, sauvegarde l'état actuel de l'indicateur, import et l'export sous format xml.

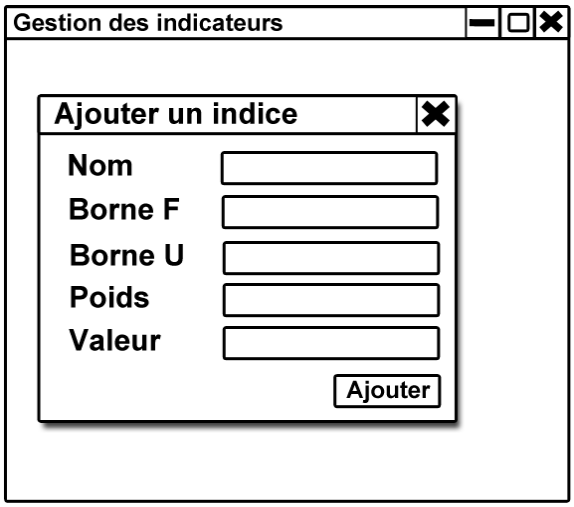

Figure 9 : Maquette du formulaire d'ajout d'un indice

La figure 9 représente le formulaire d'ajout un nouveau indice. Distingué par son nom, son poids, sa valeur et ses bornes favorable et défavorable.

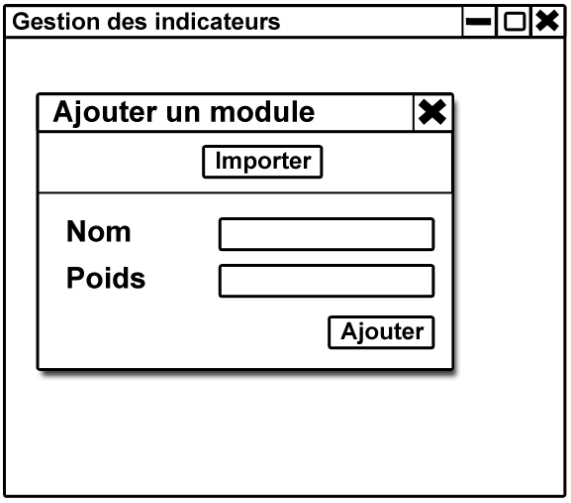

Figure 10 : Maquette du formulaire d'ajout d'un module

La figure 10 représente le formulaire d'ajout un nouveau module, avec la possibilité d'importer un module déjà défini.

II.4. Les outils et technologies utilisées

II.4.1. C++ et Qt

Le choix de C++ et Qt pour le développement de l'application fait partie du cahier des charges du projet. C++ est le meilleur choix pour ce type d'application qui demande beaucoup de calculs. Le framework Qt est le plus utilisé pour construire des applications à interface graphique en C++, il est flexible et s'adapte à n'importe quelle plateforme.

II.4.2. QML et Javascript

Dans les anciennes versions de Qt, les interfaces graphiques devaient être codées en C++. Ce n'est plus le cas dans les nouvelles versions où Qt a introduit un nouveau langage appelé QML (Qt Meta Language ou Qt Modeling Language) qui est un langage déclaratif pour construire les interfaces graphiques de façon simple et flexible.

Pour pouvoir gérer l'interaction de l'utilisateur avec l'interface graphique, le QML utilise du code Javascript. Le Javascript est un langage dynamique de scripting utilisé en général sur les pages web.

II.4.3. Git et Github :

Afin de pouvoir travailler en parallèle sur le même projet, et pour pouvoir garder traces de toutes les versions du code, nous avons utilisé l'outil Git en créant un dépôt sur le site Github.com.

II.4.4. Editeur, compilateur et SE :

On a utilisé l'éditeur **Sublime Text 3** pour l'écriture du code, le compilateur **g++** avec les deux commandes **make** et **qmake** pour la compilation du code, et la distribution **Ubuntu 14.04** de Linux comme système d'exploitation.

Réalisation et résultat

III.1. L'architecture de l'application

On a divisé l'application en trois parties :

- **Partie centrale** : le cœur de l'application qui contient les classes métiers et qui fait les calculs. Cette partie a été construite sous forme d'une bibliothèque statique indépendante du reste de l'application pour pouvoir l'intégrer facilement dans n'importe quel autre service ou plateforme au sein de l'INRA.
- **Partie des tests unitaires** : Cette partie s'appuie sur la partie centrale pour tester toutes les fonctionnalités de façon unitaire.
- **L'interface graphique** : l'interface graphique de l'outil est elle-même une application à part qui utilise la partie centrale comme étant une bibliothèque externe. On a choisi de séparer l'interface graphique et la partie centrale pour donner plus de flexibilité et de découplage à notre produit final.

III.2. La partie centrale

Le diagramme de classes de cette partie est le suivant :

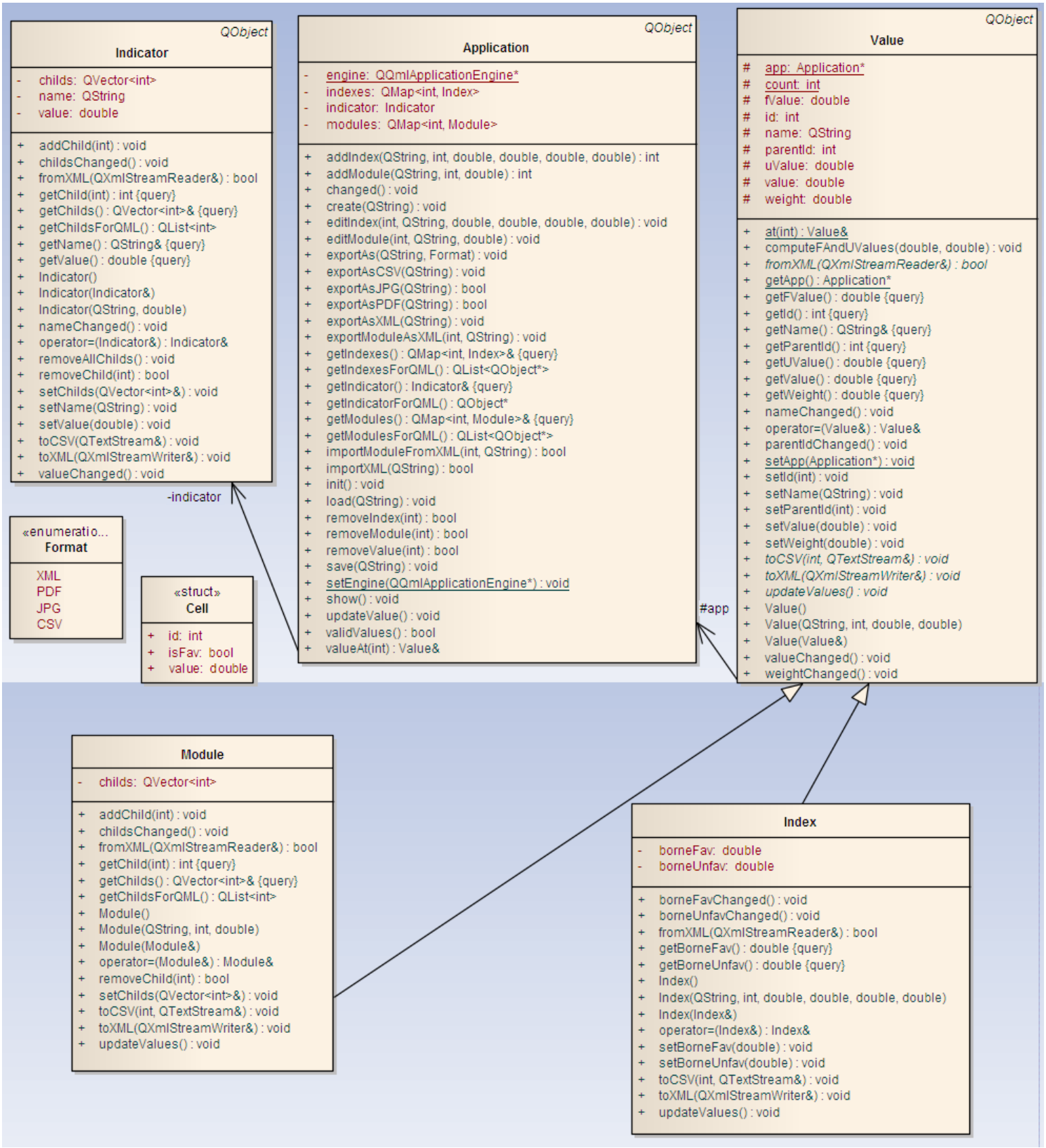

Figure 11 : Diagramme de classes de la bibliothèque centrale

Comme le montre le diagramme de classes, toutes les classes héritent de la classe QObject qui est la classe de base du framework Qt et utilisent des macros prédéfinis dans le framework pour déclarer les propriétés et les signaux qui vont nous aider à communiquer avec la partie d'interface graphique. Les principales classes de l'application sont :

- **Value** : Cette classe représente une valeur qui peut être un indice ou un module. Elle contient ainsi les attributs communs entre ces deux classes et des méthodes abstraites qui vont être implémentées de façons différentes dans les deux classes.
- **Index** : hérite de la classe Value et ajoute les deux attributs borne favorable et borne défavorable.
- **Module** : hérite de la classe Value et agrège une liste de valeurs.
- **Indicator** : C'est la classe qui représente un indicateur qui agrège un ensemble d'indices et/ou de modules.
- **Application** : Cette classe joue le rôle du contrôleur, c'est le point d'entrée pour utiliser la bibliothèque. Ses méthodes sont les implémentations des fonctionnalités de l'outil.

La méthode la plus importante de la classe Application est **updateValue**, c'est cette méthode qui lance le calcul des valeurs des modules et met à jour la valeur de l'indicateur. Son code est le suivant :

```
void Application::updateValue(){
    if(validValues()){ // Si les valeurs sont valides
        if(indicator.getChilds().size() == 0){ // Si l'indicateur n'a pas de fils
            indicator.setValue(-1);
        } else if(indicator.getChilds().size() == 1){ // Si un seul fils
indicator.setValue(Value::at(indicator.getChilds().first()).getFuzzyValue());
        } else {
            // Calculer la somme des poids des fils
            double totalWeight = 0;
            foreach(int id, indicator.getChilds()){
                totalWeight += valueAt(id).getWeight();
            }
            // Construction de la table des calculs
            int cols = indicator.getChilds().size(), rows = pow(2, cols), i, j, k, l;
            bool f;
            Cell ** table = new Cell* [cols];
            table[0] = new Cell[cols * rows];
            for(j = 1; j < cols; ++j)
                table[j] = table[0] + j * rows;
            // Remplir la table par les valeurs F et U
            j = 0; l = 1;
            foreach(int id, indicator.getChilds()){
                Value& v = valueAt(id);
                v.updateValues();
                i = 0; f = true;
                while(i < rows){
                    for(k = 0; k < l; ++k){
                        table[j][i].id = v.getId();
                        table[j][i].isFav = f;
                        table[j][i].value = ( f ) ? v.getFValue() : v.getUValue();
                        ++ i;
                    }
                    f = ! f;
                }
                ++ j; l = 2 * l;
            }
            // Calculer la valeur du module
            double truthValuesSum = 0, fuzzySolution = 0, w, B;
            for(i = 0; i < rows; ++ i){
                w = 1; B = 0;
                for(j = 0; j < cols; ++ j){
                    if(w > table[j][i].value)
                        w = table[j][i].value;
                    if(! table[j][i].isFav)
                        B += valueAt(table[j][i].id).getWeight() / totalWeight;
                }
                truthValuesSum += w;
                fuzzySolution += w * B;
            }
            indicator.setValue(fuzzySolution / truthValuesSum);
        }
    } else
        indicator.setValue(-1);
}
```

Figure 12 : Implémentation de Application::updateValues

Cette méthode appelle de manière récursive la méthode updateValues des fils. Si le fils est un module, il va faire à peu près le même traitement de la fonction ci dessus. S'il est un indice, l'implémentation est la suivante :

```
if( borneF < borneU ){
    if(value <= borneF)
        fValue = 1;
    else if(value <= ( borneF + borneU ) / 2.0)
        fValue = 1 - 2 * ((value - borneF) / (borneU - borneF)) * ((value - borneF) /
(borneU - borneF));
    else if(value <= borneU)
        fValue = 2 * ((borneU - value) / (borneU - borneF)) * ((borneU - value) /
(borneU - borneF));
    else
        fValue = 0;
    uValue = 1 - fValue;
} else if( borneU < borneF ){
    if(value <= borneU)
        uValue = 1;
    else if(value <= ( borneU + borneF ) / 2.0)
        uValue = 1 - 2 * ((value - borneU) / (borneF - borneU)) * ((value - borneU) /
(borneF - borneU));
    else if(value <= borneF)
        uValue = 2 * ((value - borneF) / (borneF - borneU)) * ((value - borneF) /
(borneF - borneU));
    else
        uValue = 0;
    fValue = 1 - uValue;
} else {
    // borneF == borneU !!
    if(value < borneF){
        fValue = 1;
        uValue = 0;
    } else {
        fValue = 0;
        uValue = 1;
    }
}
```

Figure 13 : Implémentation de Index::updateValues

III.3. Les tests unitaires

Dans cette partie on a utilisé le framework des tests unitaires **QTest** qui fait partie des modules de base du framework Qt et qui offre la possibilité de faire des tests unitaires des classes qui héritent de la classe de base **QObject**. Comme toutes nos classes héritent de cette classe, les tests unitaires de toutes les méthodes de la classe **Application** ont été possibles et nous ont bien dirigés pendant le développement de l'outil et la détection des erreurs.

III.4. L'interface graphique

Cette partie a pris plus du temps que les deux autres parties. Car il nous a fallu nous documenter sur le langage QML et sur l'utilisation du Javascript pour gérer les évènements d'utilisateur.

III.4.1. Composantes QML customisées

Pour construire l'interface graphique comme prévue sur les maquettes, on avait besoin de construire nos propres composants QML. On a commencé par la construction du composant MiniButton qui présente un petit bouton qui change de couleur lorsque la souris passe dessus et qui lance un traitement lorsqu'il est cliqué. Son code QML est le suivant :

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Window 2.2
import QtQuick.Dialogs 1.2
import QtQuick.Layouts 1.1

Rectangle {
    id: btn
    property string normalColor: '#1852B7'
    property string hoverColor: '#5C8CE8'
    property alias text: txt.text
    property alias area: area

    color: normalColor
    width: 30
    height: 30
    border.width: 1
    border.color: "#000000"

    MouseArea {
        id: area
        anchors.fill: parent
        hoverEnabled: true
        onEntered: parent.color = hoverColor
        onExited: parent.color = normalColor
    }
    Text {
        id: txt
        anchors.centerIn: parent
        text: 'X'
    }
}
```

Figure 14 : fichier MiniButton.qml

Comme le montre la figure 14, le fichier QML commence par des **import**s qui incluent les bibliothèques QML dont nous avons besoin. Puis la définition de l'élément graphique en se basant sur les éléments prédéfinis dans les bibliothèques que nous avons importé. Dans note cas, le MiniButton est un **Rectangle** pour lequel nous définissant le couleur de fond, la largeur, la hauteur et la bordure, et dans lequel nous mettons un espace cliquable **MouseArea** et du texte. Le changement de la couleur du fond lorsque la souris entre ou sorte du rectangle est assuré par les événements **onEntered** et **onExited** du **MouseArea**.

Ensuite nous avons utilisé cette composante avec d'autres composantes de QML pour construire la composante Block qui représente un indicateur, indice ou module sur notre interface graphique, et ensuite construire les formulaires sur lesquels l'utilisateur va saisir les données pour créer les indices et des modules.

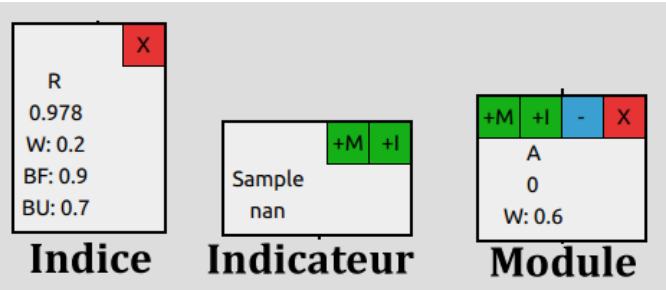

Figure 15 : Les types de block

III.4.2. La gestion des événements en Javascript

Les actions faites par l'utilisateur sur l'interface graphique déclenchent des fonctions Javascript qui traitent ces actions et appellent les méthodes appropriées de la classe Application de la partie centrale de l'application. En particulier à chaque changement, l'arbre graphique qui représente la hiérarchie des indices et redessiné. La fonction qui fait le dessin de l'arbre est la suivante :

```
Box.draw = function(){
       // Construction des blocks
       Box.makeComponents();
       // Calcul des dimensions de la fenêtre
       var dim = Box.computeWH();
       content.width = Math.max(content.width, dim.w);
       content.height = Math.max(content.height, dim.h);
       // Dessiner le block de la racine (l'indicateur)
       // au center haut de la feêntre
       Box.blocks[0].drawAt((content.width - dim.w) / 2, 5);
       content.visible = false;
       content.visible = true;
}
```

Figure 16 : La fonction Box.draw du javascript

La fonction drawAt dessine le block aux coordonnées indiquées et dessine ces fils. En plus de la fonction du dessin, le fichier Javascript contient les fonctions de validation des formulaires et les fonctions de mise en relation avec la partie centrale.

III.5. Le résultat

Une capture d'écran de l'application finale est comme suite :

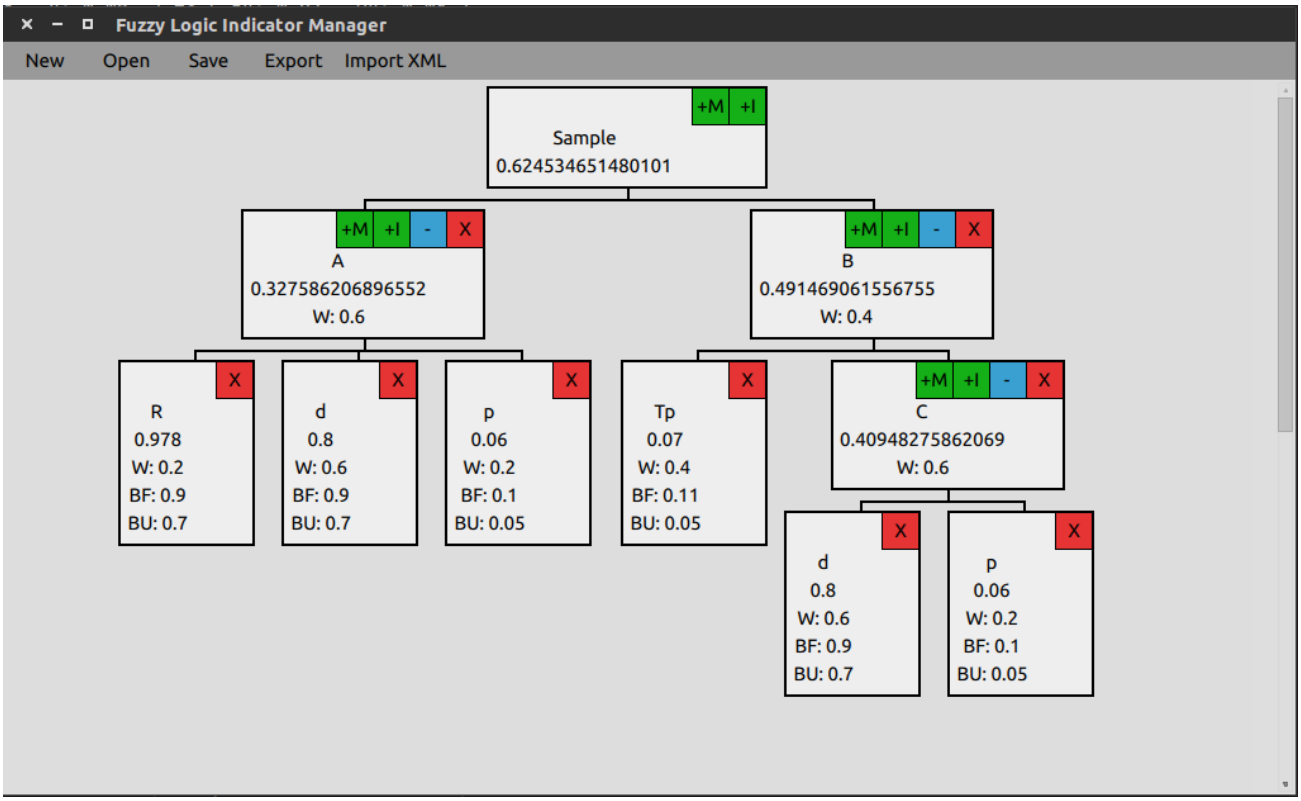

Figure 17 : Résultat final

Conclusion

Dans le cadre de nos études en 3ème année à l'ISIMA, nous avons réalisé un projet intitulé « Outil de calcul d'indicateurs d'évaluation de modèles de simulations par la logique floue ». Cet outil doit être utilisé par les chercheurs de l'INRA pour évaluer les modèles des indicateurs. Nous avons réalisé cet outil en C++ et Qt en nous basant sur la théorie de la logique floue et en élaborant une interface graphique ergonomique à l'aide de QML et du Javascript. Cet outil comportait trois modules : un module central écrit en C++ sous la forme d'une bibliothèque indépendante qui fait les calculs ; un deuxième module des tests unitaires ; et un troisième module constitué par l'interface graphique de l'outil.

L'outil ainsi créé a été livré et installé sous Windows sur les postes de Monsieur MARTIN Raphael (responsible scientifique à l'UREP) et Monsieur BELLOCCHI (chercheur en sciences agricoles et directeur de l'UREP), ainsi que sur d'autres postes sous Linux. L'étape suivante va être de l'intégrer dans une plateforme développée par l'INRA afin de pouvoir l'utiliser dans un serveur pour analyser et évaluer des modèles qui vont être générés et passer automatiquement à cet outil.

Webographie

[Logique floue] http://fr.wikipedia.org/wiki/Logique_floue (10/01/2015)

[Documentation officielle de Qt] http://doc.qt.io (15/02/2015)

Annexe : Guide d'installation

1- Télécharger et installer la version 5.4 de Qt depuis le site: www.qt.io/download
2- Ajouter les liens DOSSIER_INSTALLATION/5.4/COMPILATEUR/bin et
   DOSSIER_INSTALLATION/5.4/Tools/QtCreator/bin dans votre PATH
   DOSSIER_INSTALLATION : l'endroit où vous avez installé Qt
   COMPILATEUR : le compilateur que vous avez choisi d'installer avec Qt
3- Lancez QtCreator et ouvrez le projet avec
4- Choisissez le dossier build/ comme endroit des exécutables qui vont être générés
5- Compiler la partie core, puis la partie gui
6- Lancer la partie gui