



Rapport de projet final - IDM

Un Environnement de Calcul Domaine-Spécifique

GROUPE : L34-04

MEMBRES:

SALOUANI ABDELMOUNAIM
TENE FOGANG ZACHARIE IGOR
OUDARD VERON
MEKKAOUI OSSAMA MOUSSA

Table de matière

| | |
|--|-----------|
| 1. Introduction..... | 4 |
| 2. Cahier de charge..... | 4 |
| 3. End User..... | 5 |
| 3.1 Conception et Architecture des Metamodèles..... | 5 |
| 3.2. Metamodèle des tableaux..... | 5 |
| 3.3. Metamodèle d'algorithme..... | 6 |
| 3.4. Metamodèle d'expression..... | 7 |
| 4. Les syntaxes concrètes graphiques(SIRIUS)..... | 7 |
| 4.1. Interface pour le métamodèle de tableau..... | 7 |
| 4.2. Interface pour les métamodèles de algorithm et expression..... | 8 |
| 5. Les contraintes java sur les différents modèles..... | 9 |
| 5.1. Contraintes sur les tableaux..... | 9 |
| 5.2. Contraintes sur les algorithmes..... | 10 |
| 5.3. Contraintes sur les expressions..... | 10 |
| 6. Les transformations de modèle à texte..... | 10 |
| 6.1. Transformation de expression vers du code python(Définition des scripts de Calcul)..... | 10 |
| 6.2. Transformation d'un modèle de tableau vers un fichier JSON..... | 10 |
| 7.Librairie..... | 11 |
| 7.1. Conception et pensée derrière la Librairie..... | 11 |
| 7.2. Étapes de génération de la Librairie..... | 11 |
| 7.3. Fonctionnalités implémentés de la Librairie..... | 12 |
| 8. Explication pour l'utilisation des différents outils du end user..... | 12 |
| 8.1. Générer les modèles de table, algorithme et expression..... | 12 |
| 8.2. Vérifier la conformité avec les contraintes java..... | 12 |
| 8.3. Transformer les modèles via acceleo..... | 13 |
| 8.4. Créer la librairie pour le end user target..... | 13 |
| 9. End user Target..... | 13 |
| 9.1. Interface utilisateur client..... | 13 |
| 9.2. Fonctionnalités..... | 14 |
| 9.2.1. Importation de données et visualisation de données..... | 14 |
| 9.2.2. Lancement du calcul automatique..... | 15 |
| 9.2.3. Modification des données..... | 16 |
| 9.2.4. Valider le schéma de table (vérifier les contraintes)..... | 17 |
| 9.2.5 Consulter la documentation et les informations sur les colonnes..... | 18 |
| 9.2.6. Exporter les données au format csv..... | 19 |
| 10. Conclusion..... | 20 |
| 11. Description du rendu..... | 21 |

Tables des figures

| | |
|--|----|
| Figure 1. Diagramme UML du métamodèle de Tableau (tableProject.ecore)..... | 6 |
| Figure 2. Diagramme UML du métamodèle de Algorithme (algorithmProject.ecore)..... | 6 |
| Figure 3. Diagramme UML du métamodèle de Expression (ExpressionProject.ecore)..... | 7 |
| Figure 4 Interface sirius pour créer des modèles de schémas de table..... | 7 |
| Figure 5. Interface sirius pour créer des modèles d'algorithme | 8 |
| Figure 6. Interface sirius pour créer des modèles d'expressions..... | 9 |
| Figure 7. Interface utilisateur..... | 14 |
| Figure 8. Illustration du fichier CSV..... | 14 |
| Figure 9. Démonstration de la fonctionnalité d'import..... | 14 |
| Figure 10. Démonstration de la fonctionnalité de visualisation..... | 15 |
| Figure 11. Interface de commande pour le lancement des calculs automatiques..... | 15 |
| Figure 12. Illustration du résultat après lancement du calcul de variation..... | 16 |
| Figure 13. Interface édition de cellules..... | 16 |
| Figure 14. Interface d'ajout de lignes..... | 17 |
| Figure 15. Illustration après ajout d'une ligne..... | 17 |
| Figure 16. Interface pour supprimer une ligne..... | 17 |
| Figure 17. Illustration de la fonctionnalité de vérification..... | 18 |
| Figure 18. Bouton lancer documentation / Consulter information..... | 18 |
| Figure 19. Interface de présentation de la documentation..... | 18 |
| Figure 20. Interface de description de la colonne..... | 18 |
| Figure 21. Boîte de dialogue d'exporter..... | 19 |

1. Introduction

L'objectif principal de ce projet est de fournir à un utilisateur (end user) un ensemble d'outils pour lui permettre de créer des schémas de table incluant des colonnes, des relations et, si nécessaire, des contraintes associées. Par la suite, il doit être capable de définir également grâce à cet ensemble d'outils des algorithmes, scripts et calculs mathématiques simples.

Enfin, cela sera fourni à un utilisateur final (end-user target), qui aura la possibilité d'importer des tables en format csv, vérifier des contraintes, lancer des calculs, visualiser et manipuler des données.

2. Cahier de charge

D'après le sujet, qui fait office de cahier des charges, nous notons que le projet se découpe en deux parties. La première a pour objectif la création d'un outil permettant à un ingénieur de concevoir une structure de données appropriée et personnalisée pour des spécialistes d'un domaine particulier, par exemple, celui du calcul scientifique. En outre, il s'agit de concevoir un ou un ensemble d'outils permettant lui-même de créer un outil générique et adaptable à une utilisation précise. La deuxième partie a pour but de permettre aux spécialistes d'utiliser cette structure de données préalablement définie afin d'importer des données et d'appliquer un certain nombre de traitements dessus.

3. End User

Cette rubrique présente la conception des outils développés pour permettre à l'ingénieur de créer une structure de données adaptée aux besoins des spécialistes. La suite d'outils débute par la définition d'un métamodèle associé à cette structure, se poursuit par le développement d'un outil graphique pour instancier ce métamodèle, et se termine par l'exportation du modèle créé.

3.1 Conception et Architecture des Metamodèles

Après avoir lu le sujet, et pour répondre aux fonctionnalités attendues F1, F2 et F3, nous avons choisi d'aborder le problème en posant trois métamodèles, (avec tableModel pour répondre à F1, algorithmProject pour répondre à F2 et expressionProject pour F3).

3.2. Metamodèle des tableaux

D'après notre métamodèle(voir Figure 1), un tableau est une classe qui possède un nom(name) et qui se compose de colonnes. L'une des colonnes qui compose la table est référencée comme la colonne de référence(colonneDeLignes), c'est la colonne qui permet d'identifier chaque ligne du tableau. La classe colonne est celle qui permet de modéliser les colonnes. Chaque colonne possède donc un nom, un type, un identifiant, elle peut posséder 0 à 2 contraintes prédéfinies(des contraintes créer à titre d'exemple) ainsi que 0 ou 1 ou plusieurs contraintes que le end user peut réaliser et importer.

Enfin, via la référence colonneType, une colonne peut importer un algorithme(ImportationAlgorithm) ou un tableau(ImportationTableau). Une colonne ne peut importer qu'un seul tableau ou un seul algorithme(ou ne rien importer). Ainsi ImportationTableau a comme attribut une référence au tableau d'où l'on importe la colonne et l'identifiant de la colonne importé et ImportationAlgorithm a comme attribut l'algorithme importé. On remarque donc que le métamodèle du tableau importe également le métamodèle de l'algorithme(On le voit également via DataType qui est une énumération dans le métamodèle de algorithme).

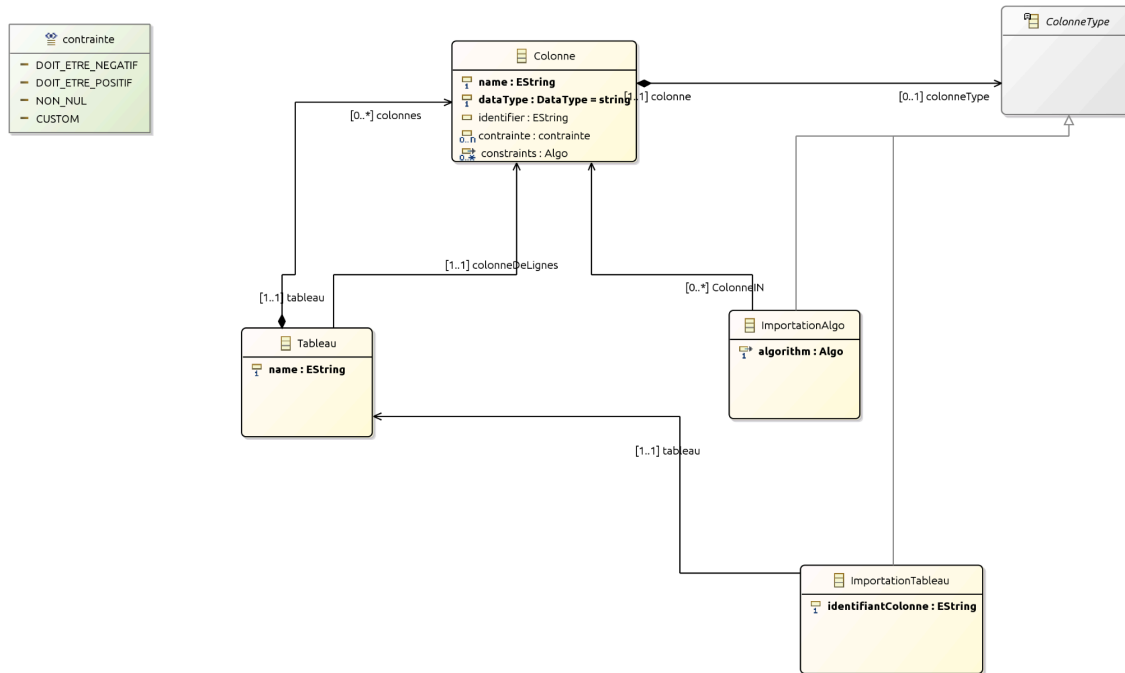


Figure 1 - Diagramme UML du métamodèle de Tableau (tableProject.ecore)

3.3. Metamodèle d'algorithme

Le métamodèle de l'algorithme (Figure 2) contient la classe Algorithm qui possède un nom et un booléen qui nous permet de savoir si l'algorithme est un algorithme quelconque qui peut modifier la colonne ou s'il est une contrainte sur une colonne qui ne fait que vérifier que les éléments sont conformes à une contrainte définie. De plus Algorithm se compose de la classe EntreeSortie pour définir les entrées et les sorties de l'algorithme qui auront un certain type (type) et une certaine valeur (value). Ainsi, c'est via les références input et output que l'on différencie les entrées et sorties de l'algorithme. Algorithm se compose également de la class Ressource qu'il doit obligatoirement avoir et qui possède un path vers l'algorithme qui sera utilisé par la colonne.

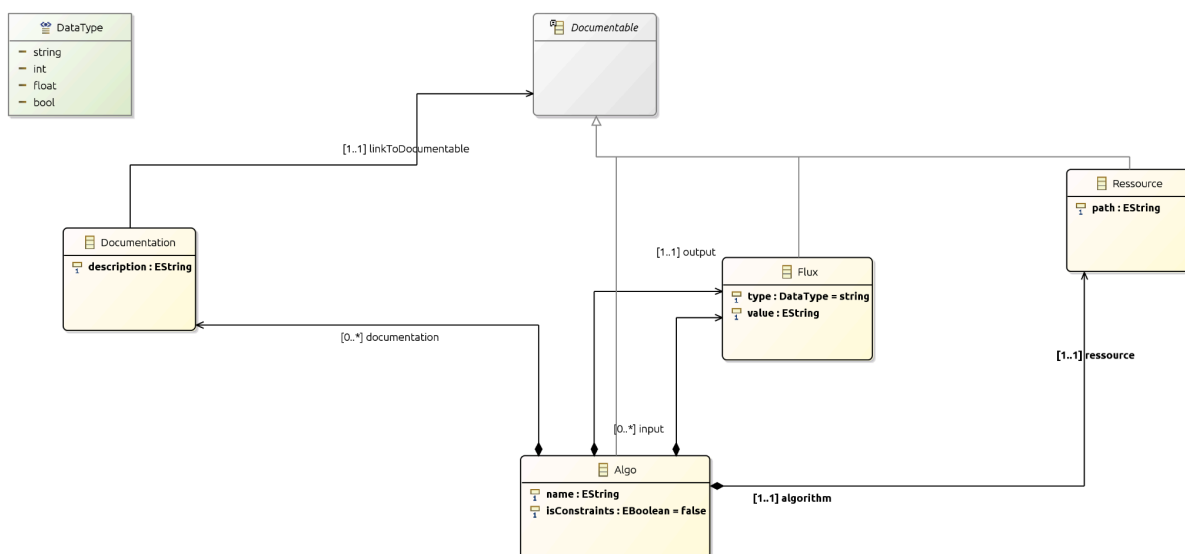


Figure 2 - Diagramme UML du métamodèle de Algorithme (algorithmProject.ecore)

3.4. Metamodèle d'expression

Le métamodèle de l'expression (Figure 3) contient la classe Expression, composée d'une ExpressionElement qui est représentable par des classes abstraites pour distinguer les comportements : une FonctionBinaire (exemple : (a,b)-> Min(a,b)), une FonctionUnaire (x->-x), une FonctionConstante (x-> value) ou un flux (EntreeSortie) qui représente les colonnes en entrée ou en sortie.

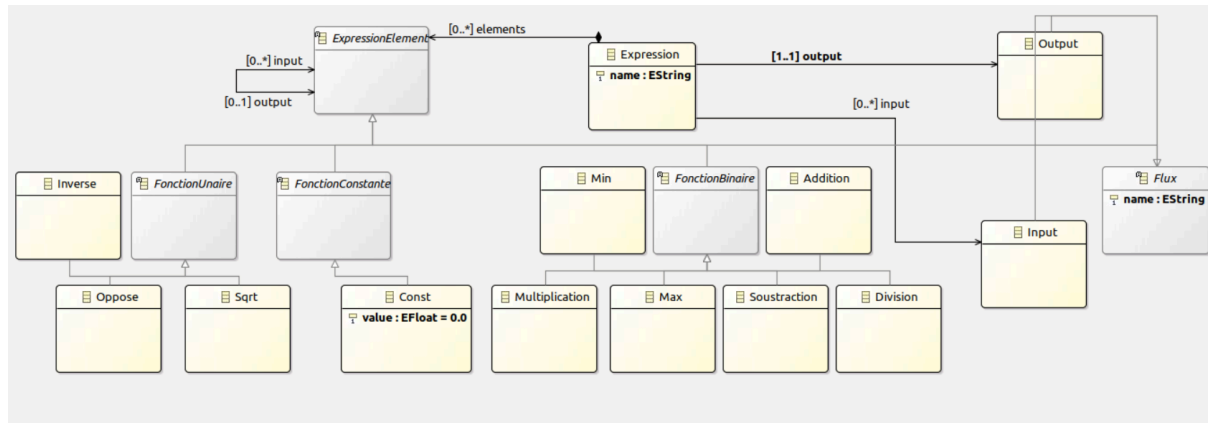


Figure 3 - Diagramme UML du métamodèle de Expression (ExpressionProject.ecore)

4. Les syntaxes concrètes graphiques (SIRIUS)

Afin de faciliter le travail de l'ingénieur concepteur, nous proposons deux interfaces graphiques dans l'éditeur Sirius inclus dans Eclipse, dont l'objectif est d'instancier les métamodèles tableau (schéma de table) et Expression avec une certaine ergonomie. Une fois les expressions de calcul et le schéma de table composés, il est possible d'exporter une instanciation de ces métamodèles sous format XMI.

4.1. Interface pour le métamodèle de tableau

Pour répondre à F1, nous avons mis à disposition de l'utilisateur une interface sirius qui lui permet de composer, sauvegarder et consulter des schémas de table (Figure 4).

| | Identifiant | Contrainte | Algorithme de contrainte | Nom | Type | Import: Table | Algorithm : Algorithme | Import: identifiant | Algorithm : Input |
|------------------------|--------------------------|---------------------|--------------------------|----------------|-------|----------------------------|------------------------|---------------------------|------------------------|
| Tableau | | | | banque finance | | | | | |
| Colonne 1 [*] | banque.finance.date | [] | | Date | float | Tableau banque finance | | | |
| Colonne 2 | banque.finance.ouverture | [DOIT_ETRE_POSITIF] | | Ouverture | float | Tableau banque finance | | | |
| ✦ Imported Colonne 3 | banque.finance.min | [DOIT_ETRE_POSITIF] | | Min | float | Tableau banque raw finance | | banque.finance.raw.open | |
| ✦ Imported Colonne 4 | banque.finance.max | [DOIT_ETRE_POSITIF] | | Max | float | Tableau banque raw finance | | banque.finance.raw.low | |
| ✦ Imported Colonne 5 | banque.finance.fermeture | [DOIT_ETRE_POSITIF] | | Fermeture | float | Tableau banque raw finance | | banque.finance.raw.high | |
| ✦ Imported Colonne 6 | banque.finance.volume | [DOIT_ETRE_POSITIF] | | Volume | float | Tableau banque raw finance | | banque.finance.raw.close | |
| ✦ Imported Colonne 7 | banque.finance.moyenne | [] | | Moyenne | float | Tableau banque raw finance | | banque.finance.raw.volume | |
| ✦ Algorithme Colonne 8 | banque.finance.variation | [] | | Variation | float | Tableau banque finance | calcul_moyenne | | [Min, Max] |
| ✦ Algorithme | | | | | | | calcul_variation | | [Fermeture, Ouverture] |

Figure 4 - Interface sirius pour créer des modèles de schémas de table (SchemaTable.odesign)

La colonne en orange est la colonne spéciale pour les identifiants des lignes.

Donc l'utilisateur peut définir/supprimer des colonnes à l'aide d'un clic droit + **Nouvelle Colonne/Delete Line**. Puis il peut saisir un identifiant, nom, et type. Il peut ensuite définir quelle contrainte s'applique sur cette colonne, tout en important l'algorithme qui l'implante (en double cliquant sur Algorithm de contrainte).

Il peut ensuite désigner qu'une colonne est importée d'un autre tableau en clic droit et en ajoutant la sous ligne Imported (**Import Column**), il peut donc définir de quel tableau cette colonne provient (Import : Table).

De même, avec une clique droit (**Call Algorithm**), il peut indiquer que la colonne est résultat d'un Algorithm (Algorithm : Algorithm) tout en ajoutant les colonnes que cet algorithme prend en entrée (Algorithm : Input).

4.2. Interface pour les métamodèles de algorithm et expression

Pour répondre à F2, nous avons mis à disposition de l'utilisateur une interface sirius qui lui permet de donner la ressource qui réalise l'algorithme (Figure 5), de définir des entrées et sorties qui sont reliées à l'interface relative à la ressource et de documenter chaque élément, pour identifier ce que fait chaque entrée, décrire l'algorithme, etc.

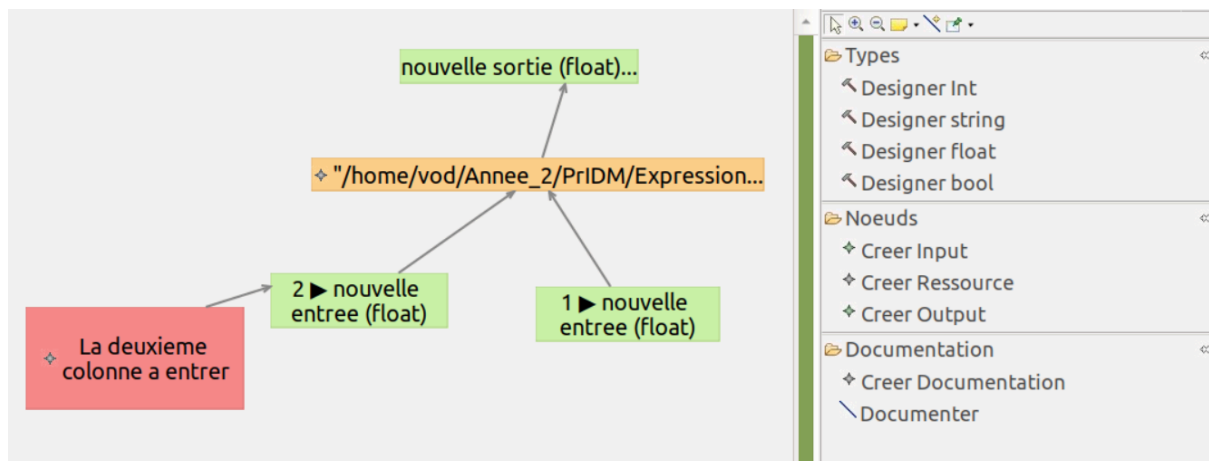


Figure 5 - Interface sirius pour créer des modèles d'algorithme (algo.odesign)

Enfin, pour répondre à F3, pour que l'utilisateur soit capable de spécifier les calculs, nous avons mis à sa disposition une troisième interface (Figure 6) pour répondre à ce besoin.

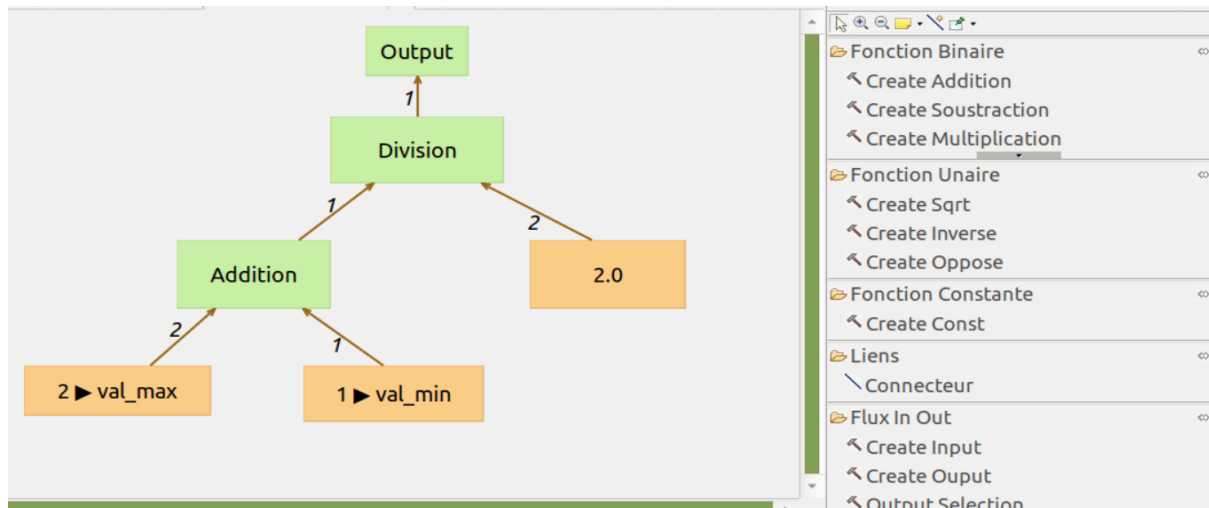


Figure 6 - Interface sirius pour créer des modèles d'expressions

À l'aide de la palette, l'utilisateur peut donc spécifier les opérations à réaliser à l'aide de blocs disposant d'entrées et de sorties (**Create Division par exemple**) des Fonction Binaire, Unaires et la fonction Constante. Il peut ensuite déclarer les entrées (**Create Input**) et les sorties (**Create Output et Output Selection**). Il peut définir des fonctions classiques mathématique avec Create Sqrt, Inverse et Opposé.

5. Les contraintes java sur les différents modèles

Certaines contraintes nécessaires au fonctionnement des modèles et des modules qui les utilisent ne peuvent pas être représentées via la modélisation avec les diagrammes UML. Ainsi, il est nécessaire de définir des contraintes directement en Java pour pouvoir empêcher certaines actions indésirables sur les modèles, ce qui risqueraient de créer des problèmes. Cependant ces contraintes sont statiques, c'est à dire que c'est au end user de lui même utiliser les code java mit à sa disposition pour vérifier que les modèles qu'il a créé sont conformes aux contraintes, la vérification n'est pas faite automatiquement.

Toutes les contraintes créées peuvent être vérifiées une par une via des modèles .xmi créés pour chaque métamodèle et qui mettent en évidence que les contraintes ne sont pas ignorées (Il suffit quand on lance par exemple ValidateTableModel.java en run as "Java Application" de préciser le chemin du modèle à vérifier qui sont dans models/contrainteTest)

5.1. Contraintes sur les tableaux

Pour le métamodèle du tableau, nous avons imposé des contraintes sur le nom du tableau, ce dernier ne peut être null, vide ou composé uniquement d'espace. De même pour les colonnes, les contraintes écrites empêchent une colonne d'avoir un identifiant et un nom null, vide ou composé seulement d'espace. De plus, pour les colonnes nous vérifions également que deux colonnes n'aient ni le même nom ni le même identifiant.

5.2. Contraintes sur les algorithmes

Pour le métamodèle de algorithme, nous avons également imposé que l'algorithme doit avoir un nom qui n'est pas null, vide ou composé uniquement d'espace. De plus, si un algorithme est une contrainte (isConstraints = true) alors il faut vérifier que le type de retour de l'algorithme est un DataType.bool. En ce qui concerne les ressources, nous avons contraint le path de la ressource à finir par ".py" car nous ne gérons que des algorithmes en python. Enfin nous avons ajouté une contrainte qui impose au **end user** de remplir les documentations qu'il a créées (On ne peut pas laisser une documentation vide).

5.3. Contraintes sur les expressions

En ce qui concerne les expressions, nous avons mis les mêmes contraintes sur le nom pour les expressions, les inputs et les outputs. De plus, nous avons également imposé qu'une fonction constante ne puisse pas recevoir d'input, qu'une fonction unaire doit avoir un input et qu'une fonction binaire doit avoir deux input. Certaines contraintes n'ont cependant pas pu être faites par manque de temps, notamment celles qui empêchent au end user de définir un attribut sortie pour un objet de type sortie ou de définir un attribut entré pour un objet de type entrée.

6. Les transformations de modèle à texte

6.1. Transformation de expression vers du code python (Définition des scripts de Calcul)

Après avoir défini une interface qui permet de construire des calculs mathématiques, il est important d'avoir la possibilité de la transformer en script (ressource) à importer dans un algorithme. On a donc choisi l'outil de transformation M2T pour transformer un calcul (expression) vers un code python.

La transformation acceleo génère donc les fonctions qu'on a définies avant dans notre méta modèle (inverse, addition...). La fonction finale générée est construite récursivement et récupère les entrées pour chaque fonction.

Ainsi, si nous testons sur moyenne.xmi la fonction finale sera :

```
def calcul(val_min ,val_max ):
    return calc_division(calc_addition(val_min,val_max),2.0)
```

6.2. Transformation d'un modèle de tableau vers un fichier JSON

Pour pouvoir manipuler les modèles de tableaux .xmi que nous avons créés, nous avons décidé de les transformer en données JSON. Nous avons estimé que les données seraient plus simples à manipuler sous cette forme plutôt que sous la forme .xmi. En effet, nous connaissions le format JSON, nous avons pensé qu'il serait plus facile de manipuler les modèles de tableaux depuis ce format. En python Cependant, il est probable que garder le format xmi nous aurait fait perdre moins de temps car nous n'aurions pas eu à réaliser la transformation.

Ainsi, nous transformons le tableau en un objet json qui contient un nom, la colonne de référence (la colonne qui référence les lignes) et une liste de colonne, cette liste de colonne contient plusieurs objets qui sont chacun des colonnes. Une colonne possède un nom, un type et un id obligatoirement. Par contre elle peut ne pas avoir de contraintes ou d'algorithmes (elle n'en importe pas), de même elle peut ne pas avoir de tableau (elle n'en importe pas) et dans ce cas la clé est mise à null. Si elle importe une contrainte, un algorithme ou une colonne alors ces éléments sont réécrit en json comme ce qui a été fait pour Tableau ou colonne.

La transformation est une bijection, on ne perd pas d'information, vous pourrez trouver un exemple dans le répertoire exemple du projet.

7. Librairie

7.1. Conception et pensée derrière la Librairie

La librairie que nous avons développée vise à répondre aux exigences exprimées dans la fonctionnalité F4. Son but est de fournir un ensemble d'outils modulaires et réutilisables, permettant de traiter des données structurées selon un schéma spécifique, défini par l'utilisateur. Nous avons conçu un générateur de librairie en suivant une approche orientée modèle, afin de maximiser la flexibilité et la maintenabilité.

7.2. Étapes de génération de la Librairie

- **Point de Départ : Schéma JSON**

Le schéma JSON sert de base pour définir les colonnes, les types, les contraintes et les relations entre les données. Nous avons inclus des fonctionnalités pour :

- a. Importer les données depuis un fichier CSV (F4.2).
- b. Valider les données en vérifiant les contraintes spécifiées dans le schéma (F4.3).
- c. Effectuer des calculs sur les colonnes dérivées, en respectant les dépendances (F4.4).
- d. Consulter la documentation et certaines informations sur les colonnes.

- **Génération Automatique de Code**

La librairie Python est générée automatiquement à partir du schéma JSON via un module python (**générateur.py**) que nous avons nous même développé. Ce processus garantit que la structure et les fonctionnalités de la librairie sont toujours alignées avec le schéma défini. Pour assurer la compatibilité et bénéficier des dernières fonctionnalités, nous avons utilisé **Python 3.12.3**. Cette version nous a permis d'exploiter des améliorations récentes du langage, notamment en termes de gestion des fichiers, des exceptions, et de définition de format texte formatés.

- **Modularité et Extensions**

La librairie est conçue pour être modulaire. Par exemple, les colonnes dérivées peuvent être calculées à l'aide de scripts externes en Python ou d'autres langages (F4.4). Cette modularité permet aux utilisateurs de personnaliser les calculs selon leurs besoins.

7.3. Fonctionnalités implémentés de la Librairie

- Importation de Données (F4.2)

La méthode *import_csv* permet de charger des données en respectant le schéma, avec une gestion stricte des colonnes obligatoires et des références croisées. En effet, pour que l'import se fasse sans encombre, il est nécessaire que le premier schéma de table conçu par le end user respecte la règle suivante: le nom des colonnes et celui des identifiants des colonnes qu'il définit pour ce schéma doit être rigoureusement identique. Par contre pour le schéma de table final (celui à obtenir) , cette règle ne s'impose pas. En effet, nous nous sommes rendu compte l'omission du nom dans notre métamodèle schéma lorsqu'on définit qu'une colonne fait référence à une autre colonne.

- Validation des Données (F4.3)

La méthode *verify_constraints* identifie les violations de contraintes, comme les valeurs nulles ou les plages non respectées. Cela permet de garantir la qualité des données.

- Calculs Automatiques (F4.4)

Les colonnes dérivées sont calculées automatiquement en utilisant des algorithmes définis. Nous avons intégré une logique de dépendances pour exécuter les calculs dans le bon ordre.

- Exportation de Données (F4.6)

La méthode *export_csv* permet de sauvegarder les données transformées dans un fichier CSV, assurant leur portabilité.

- Gestion des Références Croisées (F4.5)

La librairie gère automatiquement les colonnes liées à d'autres tables, en utilisant les identifiants uniques définis dans le schéma.

- Edition des données

La librairie permet de gérer l'ajout d'une ligne de donnée, de modifier une donnée, de supprimer une ligne de données.

- Consulter la documentation ou des informations sur des colonnes

En effet, la librairie permet également d'obtenir des informations comme le type de donnée d'une colonne, son identifiant et voir la documentation de celle-ci si elle a été fournie.

8. Explication pour l'utilisation des différents outils du end user

8.1. Générer les modèles de table, algorithme et expression

Se référer à la **partie 3** qui explique de manière exhaustive comment créer des modèles en utilisant sirius et les syntaxes concrètes graphiques réalisées.

8.2. Vérifier la conformité avec les contraintes java

Une fois les modèles .xmi d'expression, de table et d'algorithme créés, il est utile avant de les transformer pour les rendre utilisable au end user target de les vérifier grâce aux contraintes java

présentes dans chaque projet. Il faut ainsi importer le .xmi dans le projet qui lui correspond (un .xmi de table dans le projet où se situe le métamodèle de table par exemple).

Ainsi nous pouvons par la suite utiliser le fichier java “Validate....java” présent dans src/<nom du métamodèle>/<nom>.validation, en faisant clic droit dessus puis run as, run configuration puis dans java application il reste à mettre le chemin du modèle à vérifier.

8.3. Transformer les modèles via acceleo

Une fois les modèles vérifiés, on peut les transformer via les transformations acceleo réalisées. Selon les objectifs de l’end user, plusieurs transformations peuvent être à réaliser. Ainsi il faut transformer les algorithmes en code python en utilisant le .mtl présent dans le projet expressionToPython (clic droit sur le fichier, run as, run configuration puis dans la zone pour les projets acceleo on entre le chemin du modèle à transformer). Il faut également transformer le modèle de schéma de table en un fichier json en utilisant le fichier ToJSON.mtl dans le projet TableToJson de la même manière que pour expressionT FonctionnalitésPython.

8.4. Créer la librairie pour le end user target

Pour générer la librairie à partir du schéma table au format json, il est impératif que le module python soit lancé avec une version de python supérieure ou égale à la **3.12.0**. En effet, une utilisation d’une version de python inférieure à celle-ci pourrait ne pas correctement fonctionner comme prévu et pourrait échouer à lancer le script lors de son exécution. Notre générateur de librairie est constitué de deux modules (**generateur.py** et **tools.py**). Le module *generateur.py* est un script lançant le module *tools.py* en utilisant le paramètre qui lui est passé en argument lors de son exécution.

Pour générer la librairie, il suffit d’exécuter le module python (*generateur.py*) en passant en argument lors de l’exécution le chemin menant au fichier json (obtenu après transformation du schéma de table élaboré). Nous avons fourni un exemple *TAB_banque_finance.json* (obtenu après transformation du *banque_finance.xmi*), dans le répertoire **exemple du projet** déposé sur gitlab. La librairie créée est un module python nommé **generated_table.py**.

9. End user Target

9.1. Interface utilisateur client

L’interface utilisateur a été créée en utilisant *python* et son module *tkinter*. Nous avons utilisé cette technologie car nous avons pensé que comme la plupart des outils que nous manipulons (librairie, scripts de calculs, et algorithmes) sont en python, il est donc judicieux et plus simple de continuer dans cette lancée pour l’interface utilisateur.

L’interface utilisateur est définie par le module python **interface.py**. Cette dernière doit se trouver obligatoirement dans le même répertoire que la librairie générée. En effet, cette dernière fait appel à celle-ci pour son fonctionnement. Le nom de la librairie doit être *generated_table.py*.

9.2. Fonctionnalités

Nous allons décrire dans cette section l'ensemble des fonctionnalités dont dispose le End User Target.

Pour utiliser l'interface, il faut lancer l'interface en utilisant la commande **python interface.py**.

Une fois l'application lancée, l'interface ressemble à cela :

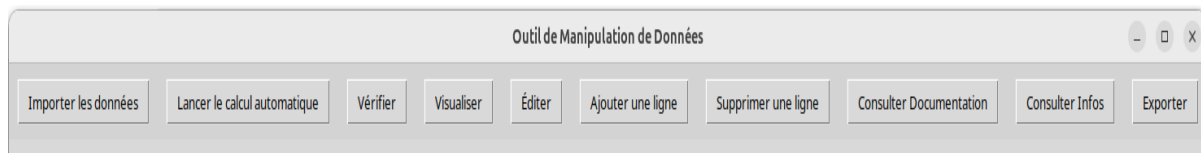


Figure 7 - Interface utilisateur

9.2.1. Importation de données et visualisation de données

Dans cette démonstration, nous utilisons le fichier `banque_finance_raw.csv` (repertoire exemple du projet sur gitlab) dont une illustration du contenu est la suivante :

| | Standard | Standard | Standard | Standard |
|---|-------------------------|-------------------------|-------------------------|------------------------|
| 1 | banque.finance.raw.date | banque.finance.raw.open | banque.finance.raw.high | banque.finance.raw.low |
| 2 | 20230918 | 137.800003 | 138.500000 | 136.860001 |
| 3 | 20230919 | 136.759995 | 136.820007 | 134.160004 |
| 4 | 20230920 | 134.380005 | 136.919998 | 134.380005 |
| 5 | 20230921 | 134.679993 | 135.440002 | 133.339996 |
| 6 | 20230922 | 133.000000 | 134.380005 | 132.619995 |
| 7 | | | | |

Figure 8 - Illustration du fichier csv

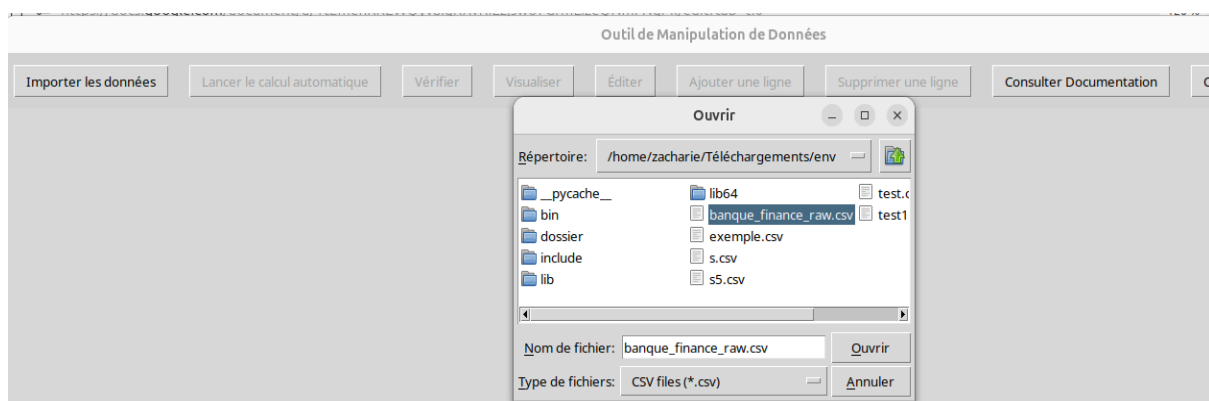


Figure 9 - Démonstration de la fonctionnalité d'import

| Outil de Manipulation de Données | | | | | | | |
|----------------------------------|------------|------------------------------|------------|---------------------|-------------------------|-----------------|-------------------|
| Importer les données | | Lancer le calcul automatique | | Vérifier | Visualiser | Éditer | Ajouter une ligne |
| | | | | Supprimer une ligne | Consulter Documentation | Consulter Infos | Exporter |
| Date | Ouverture | Min | Max | Fermeture | Volume | Moyenne | Variation |
| 20230918 | 137.800003 | 136.860001 | 138.5 | 137.179993 | 806069 | None | None |
| 20230919 | 136.759995 | 134.160004 | 136.820007 | 134.539993 | 908349 | None | None |
| 20230920 | 134.380005 | 134.380005 | 136.919998 | 136.199997 | 936370 | None | None |
| 20230921 | 134.679993 | 133.339996 | 135.440002 | 133.679993 | 1457225 | None | None |
| 20230922 | 133.0 | 132.619995 | 134.380005 | 133.240005 | 1440069 | None | None |

Figure 10 - Démonstration de la fonctionnalité de visualisation

Comme on peut le constater, lors de l'importation, le schéma du fichier csv est directement mappé avec celui du schéma de table défini par le end user . Les colonnes qui doivent être obtenues par dérivation ont leur valeur par défaut à None. Si on tente d'importer un fichier csv ne respectant le schéma défini , une erreur est lancée signalant le problème.

9.2.2. Lancement du calcul automatique

Lorsque le end user target clique sur Lancer le calcul automatique, il a la possibilité de choisir laquelle des colonnes dérivées, il souhaite obtenir. Il dispose également de l'option "All" pour lancer tous les calculs en une seule fois.

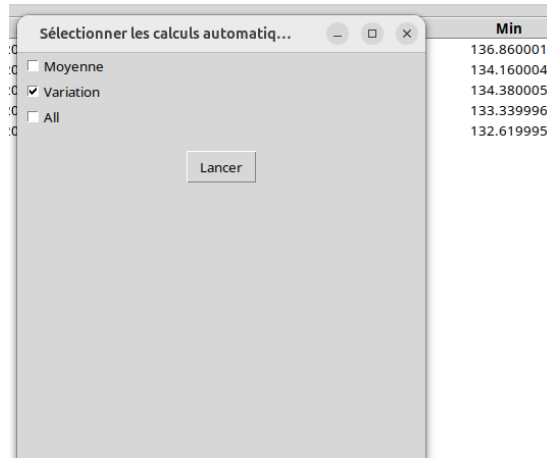


Figure 11 - Interface de commande pour le lancement des calculs automatiques

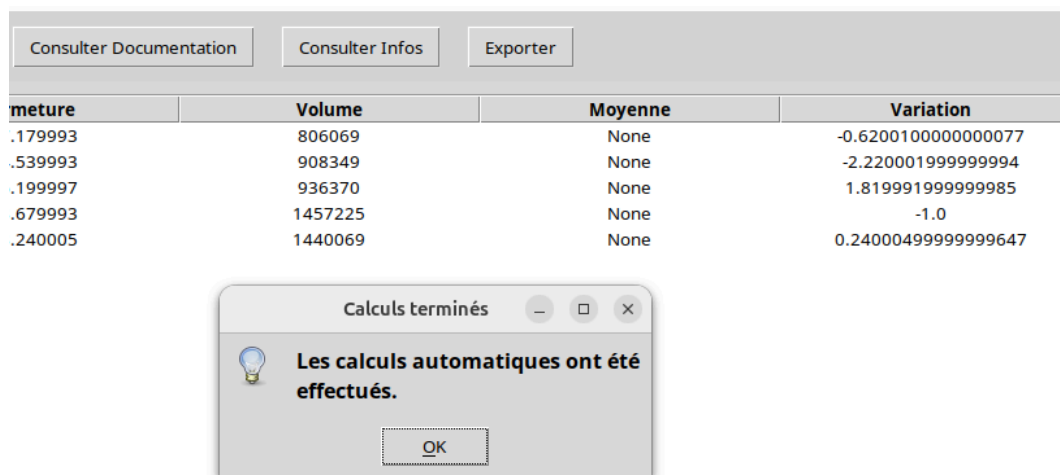


Figure 12 - Illustration du résultat après lancement du calcul de variation

9.2.3. Modification des données

L'interface permet à l'utilisateur d'éditer ces données. Nous rappelons que celui-ci ne peut pas manipuler de colonnes (rigides). Il ne peut qu'ajouter/modifier/supprimer des lignes. Nous allons ici détailler l'utilisation de ces 03 fonctionnalités. :

- **Editer une donnée**

Lorsque l'utilisateur clique sur Éditer, un premier menu suivant s'affiche où il est appelé à saisir le numéro de la ligne (L'indexation commence à 0) puis un second menu (juste après avoir renseigné le premier menu et que celui ci s'est fermé) où il est invité à saisir le nom de la colonne:

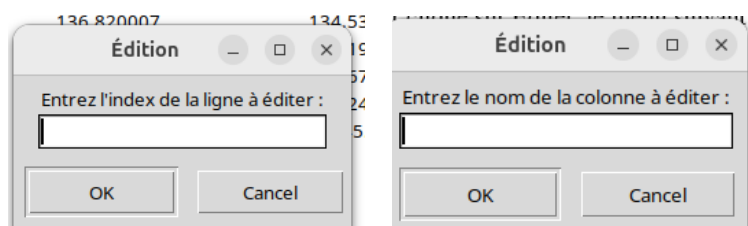


Figure 13 - Interface edition de cellules

- **Ajouter une ligne**

Lorsque l'utilisateur clique sur Ajouter une ligne, le menu suivant s'affiche

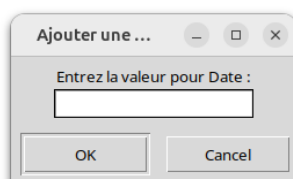
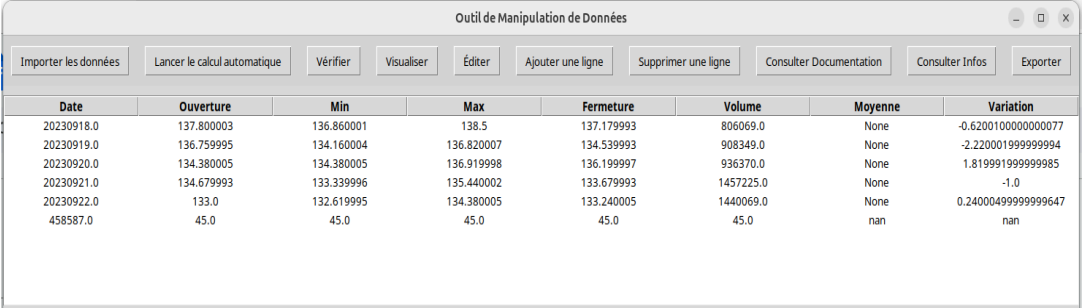


Figure 14 - Interface d'ajout de lignes

Il est appelé à renseigner la valeur conformément au type de données attendues par la colonne en question (ici Date), lorsqu'il finit et clique sur ok ou entrer, celle-ci se ferme et laisse place à une autre fenêtre pour l'entrée de la ligne correspondant à la prochaine colonne. Par défaut, l'on ne peut pas ajouter de données pour les colonnes qui doivent être obtenues par calcul car elles sont censées être déduites par calculs.

Pour ces colonnes, la valeur par défaut dépend du type de la colonne.



| Date | Ouverture | Min | Max | Fermeture | Volume | Moyenne | Variation |
|------------|------------|------------|------------|------------|-----------|---------|---------------------|
| 20230918.0 | 137.800003 | 136.860001 | 138.5 | 137.179993 | 806069.0 | None | -0.6200100000000077 |
| 20230919.0 | 136.759995 | 134.160004 | 136.820007 | 134.539993 | 908349.0 | None | -2.2200019999999994 |
| 20230920.0 | 134.380005 | 134.380005 | 136.919998 | 136.199997 | 936370.0 | None | 1.8199919999999985 |
| 20230921.0 | 134.679993 | 133.339996 | 135.440002 | 133.679993 | 1457225.0 | None | -1.0 |
| 20230922.0 | 133.0 | 132.619995 | 134.380005 | 133.240005 | 1440069.0 | None | 0.24000499999999647 |
| 458587.0 | 45.0 | 45.0 | 45.0 | 45.0 | 45.0 | nan | nan |

Figure 15 - Illustration après ajout d'une ligne

La valeur nan affichée pour les colonnes dérivées est tout à fait normale, il s'agit de la façon dont les modules python utilisés (pandas en l'occurrence) pour la manipulation gère certaines valeurs non renseignées.

- **Supprimer une ligne**

Lorsque l'utilisateur clique sur Supprimer une ligne, le menu suivant s'affiche

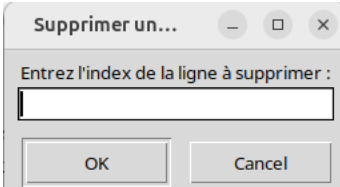


Figure 16 - Interface pour supprimer une ligne

Il suffit de renseigner le numéro de la ligne à supprimer. L'indexation commence à 0.

9.2.4. Valider le schéma de table (vérifier les contraintes)

Lorsqu'il clique sur vérifier, le processus de test des contraintes est lancé. Une fois ce dernier finit: un menu s'affiche pour présenter le résultat. Dès le premier test faux, ce dernier s'arrête:

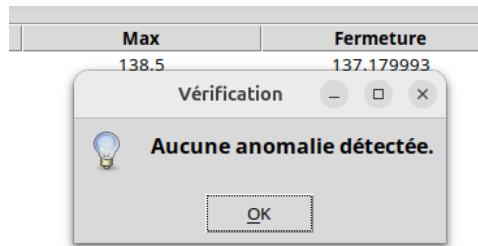


Figure 17- illustration de la fonctionnalité de vérification

9.2.5 Consulter la documentation et les informations sur les colonnes

Nous donnons la possibilité au end user target de pouvoir consulter la documentation d'une colonne (Figure 19) et consulter certaines informations sur les colonnes (Figure 20):

Lorsqu'il clique sur consulter la documentation , l'interface suivante s'affiche pour lui demander d'entrer le nom de la colonne dont il souhaite obtenir la documentation:

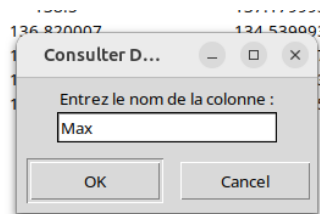


Figure 18 - bouton lancer documentation /Consulter information

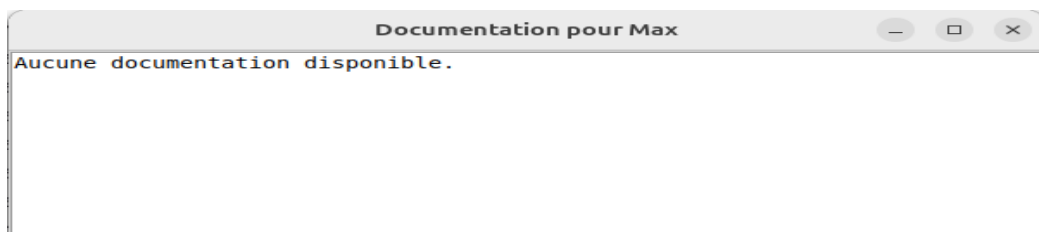


Figure 19 - Interface de présentation de la documentation

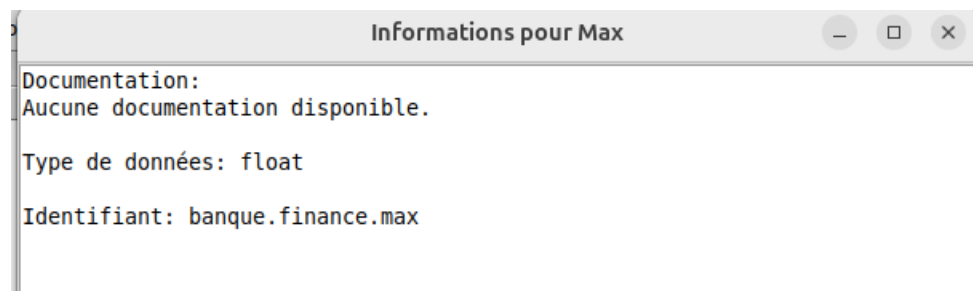


Figure 20 - Interface de description de la colonne

9.2.6. Exporter les données au format csv

Lorsqu'il clique sur exporter, un menu déroulant s'affiche alors, il clique alors sur CSV et choisit avec la boîte de dialogue l'endroit où il souhaite le sauvegarder.

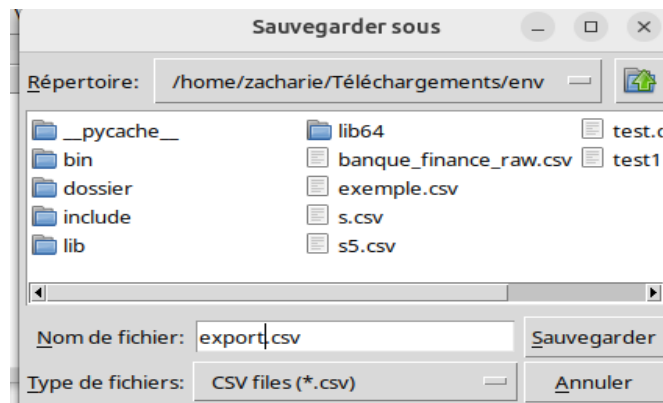


Figure 21 - boîte de dialogue d'exporter

10. Conclusion

Ce projet a permis de concevoir et de réaliser un environnement de calcul répondant aux fonctionnalités spécifiées. En s'appuyant sur des métamodèles distincts pour les tableaux, les algorithmes et les expressions, nous avons pu structurer de manière efficace la solution. Les outils graphiques développés avec Sirius offrent une interface intuitive pour composer, consulter et manipuler ces éléments, tandis que les transformations de modèles Acceleo assurent une génération cohérente de code Python et JSON.

Les défis principaux ont résidé dans la conception des métamodèles qui ont constamment été changés lors de la réalisation du projet pour les adapter à l'évolution de notre travail et dans la réalisation de la librairie pour pouvoir récupérer tous les fichiers générés et les utiliser dans l'interface graphique python. De plus, comme expliqué dans le rapport, il serait intéressant de se questionner sur la réelle utilité de la transformation en json par rapport à l'utilisation directe du xmi.

A l'avenir, il serait possible d'améliorer encore l'ergonomie des outils et d'augmenter les formats de code supportés. Malgré cela, l'outil développé constitue une base fonctionnelle et modulaire, permettant à un utilisateur non expert en informatique de concevoir et d'exploiter des schémas de données et des calculs spécifiques à son domaine.

11. Description du rendu

Notre rendu sur gitlab se compose de plusieurs dossiers:

- Le dossier acceleo contient les deux fichiers acceleo ExpressionToPython.mtl et toJSON.mtl
- Le dossier contrainte contient trois dossiers qui les fichiers java des contraintes sur les tables, les expressions et les algorithmes
- Le dossier exemple contient l'exemple de la banque décrit dans le sujet du projet avec notamment le fichier csv avec les données, le modèle .xmi utilisé pour représenter la table de la banque et la transformation acceleo du fichier .xmi.
- Le dossier metamodelle contient les trois métamodèles algorithmProject.ecore, expressionProject.ecore et tableProject.ecore
- Le dossier outil contient les trois fichiers python dont deux generateur.py ,tools.py utilisés par le end user et un, interface.py utilisé par le end user target.
- Le dossier ressource sirius contient les trois syntaxes graphiques sous sirius pour générer les modèles : algo.odesign, expression.odesign, schemaTable.odesign
- Le dossier ressource contient trois fichiers .png qui sont les représentations graphiques de chaque métamodèle.
- Le dossier workspace contient le dossier eclipse-workspace dans lequel il y a les métamodèles, les contraintes java et les transformations acceleo. De plus ce dossier contient également le dossier runtime-EclipseApplication dans lequel il y a les outils Sirius pour générer des modèles.