



L'ENSEEIHT

RAPPORT DU PROJET IDM

---

## Rapport Projet IDM - Un Environnement de Calcul Domaine-Spécifique

---

*L'équipe :* AKKAR KHADIJA, ABOUMEJD WISSAL, BELAHRACH SAFAE, FARHAT OTMAN

*Tuteur :* M. Le PANTEL MARC  
Génie Logiciel L1

1<sup>er</sup> Janvier 2024 — 22 Janvier 2024

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Étapes de la réalisation du projet</b>	<b>3</b>
<b>3</b>	<b>Création des Métamodèles Ecore</b>	<b>4</b>
3.1	Le métamodèle Schema . . . . .	4
3.1.1	Première version . . . . .	4
3.2	Le métamodèle Algorithme . . . . .	5
3.2.1	Première version . . . . .	5
3.2.2	Deuxième version . . . . .	6
3.2.3	Troisième version . . . . .	6
3.3	Les contraintes OCL du métamodèle schema . . . . .	8
3.3.1	Explication des contraintes OCL du métamodèle schema . . . . .	8
3.3.2	Explication des contraintes OCL du métamodèle Algorithme . . . . .	9
3.3.3	Lien entre les métamodèles Schema.ecore et Algorithme.ecore . . . . .	10
<b>4</b>	<b>Documentation des algorithmes en HTML par outil acceleo</b>	<b>11</b>
<b>5</b>	<b>Représentation des données d'un schème de tables dans un Tableau HTML en utilisant Python</b>	<b>12</b>
<b>6</b>	<b>Représentation des données d'un schème de tables en utilisant un script gnuplot</b>	<b>12</b>
<b>7</b>	<b>Les Transformations : Java - La librairie</b>	<b>13</b>
7.1	Transformation : CSV - Modèle de schéma . . . . .	13
7.2	Transformation : Modèle de schéma - CSV . . . . .	15
<b>8</b>	<b>Manipulation du calcul des résultats</b>	<b>16</b>
<b>9</b>	<b>Transformation à Xtext</b>	<b>17</b>
<b>10</b>	<b>L'éditeur graphique : Sirius</b>	<b>18</b>
<b>11</b>	<b>Entraves et Challenges</b>	<b>19</b>
<b>12</b>	<b>Gestion Dynamique du Projet</b>	<b>21</b>
12.1	Sprint 1 : Metamodèles et Contraintes OCL . . . . .	21
12.2	Sprint 2 : Documentation HTML des Algorithmes avec Acceleo . . . . .	22
12.3	Sprint 3 : Représentation des Données en Tableau HTML avec Python . . . . .	22
12.4	Sprint 4 : Représentation des Données avec gnuplot . . . . .	22
12.5	Sprint 5 : Transformations CSV - Modèle de Schéma et Vice Versa en Java . . . . .	23
12.6	Sprint 6 : Transformation à Xtext et Éditeur Graphique Sirius . . . . .	24
12.7	Outils utilisés . . . . .	24
<b>13</b>	<b>Conclusion</b>	<b>28</b>

## Table des figures

1	Le modèle de Schéma de tables . . . . .	4
2	Le modèle 1 des Algorithmes . . . . .	5
3	Le modèle 2 des Algorithmes . . . . .	6
4	Le modèle final des Algorithmes . . . . .	7
5	L'éditeur final des Algorithmes . . . . .	7
6	Contraintes OCL Schema . . . . .	9
7	Les contraintes 1 OCL sur les algorithmes . . . . .	9
8	Les contraintes 2 OCL sur les algorithmes . . . . .	10
9	La liaison entre Schéma/Algorithmme . . . . .	11
10	Permettre à l'utilisateur de générer un script gnuplot pour faire des graphiques . . . . .	13
11	Le modèle de Schéma de tables créé par la transformation JAVA à partir du fichier .csv . . . . .	15
12	Le génération du Java '.csv' . . . . .	15
13	Extrait de la transformation modèle2CSV . . . . .	16
14	Extrait du calcul ATL . . . . .	16
15	Transformation Xtext . . . . .	17
16	Editeur graphique Sirius . . . . .	18
17	Des erreurs d'ouverture - 'xmi' . . . . .	19
18	Des erreurs de compilation/exécution JAVA . . . . .	20
19	Des erreurs de Sirius . . . . .	20
20	Extrait de nos activités sur Github . . . . .	25
21	Extrait 1 de nos activités sur Trello . . . . .	26
22	Extrait 2 de nos activités sur Trello . . . . .	26
23	Extrait 2 de nos activités sur Trello . . . . .	27

## 1 Introduction

L'objectif fondamental de ce projet est de concevoir une série d'outils intégrés au sein de la plateforme Eclipse, exploitant les technologies EMF. Cette suite a pour finalité de permettre à l'utilisateur de définir des schémas de données et d'automatiser des opérations de calcul liées à ces schémas.

Afin de gérer les grandes quantités de données et d'effectuer des calculs complexes, on utilise des logiciels de la famille des tableurs, tels qu'Excel, ainsi que des systèmes de gestion de bases de données comme MySQL. Cependant, la manipulation de ces logiciels peut s'avérer difficile pour ceux qui ne sont pas à l'aise avec les domaines de l'informatique. C'est pourquoi on utilise l'ingénierie dirigée par les modèles pour faciliter la création d'outils de gestion des données par des utilisateurs experts dans leur domaine, mais pas nécessairement en informatique.

## 2 Étapes de la réalisation du projet

Notre projet de modélisation des systèmes se déploie à travers un processus méthodique :

- **Analyse des Besoins** : Comprendre les besoins spécifiques des utilisateurs en termes de modélisation de données et d'opérations de calcul implique des discussions approfondies entre notre groupe et les parties prenantes afin de définir les fonctionnalités requises ;
- **Distribution des Tâches** : Attribution judicieuse des responsabilités aux membres en fonction de leurs compétences, assurant ainsi une collaboration fluide et une progression efficace du projet. La clarté des rôles garantit une mise en œuvre harmonieuse des fonctionnalités définies dans le modèle ;
- **Conception du Modèle** : Développer des modèles conceptuels pour représenter les schémas de données et les algorithmes de calcul. Cela englobe la définition des entités, des relations et des opérations de calcul associées. Utiliser les concepts de l'ingénierie dirigée par les modèles afin d'assurer une flexibilité et une évolutivité optimales ;
- **Développement des Outils** : Mettre en œuvre les fonctionnalités définies dans le modèle en utilisant les technologies EMF au sein de la plateforme Eclipse. Cela implique la création d'outils intégrés permettant aux utilisateurs de définir de manière intuitive des schémas de données ainsi que des fonctions de calcul ;
- **Tests et Validation** : Réaliser des tests approfondis pour s'assurer que nos outils fonctionnent correctement et répondent aux besoins spécifiques des utilisateurs. Impliquer les utilisateurs finaux pour des retours d'expérience précieux .

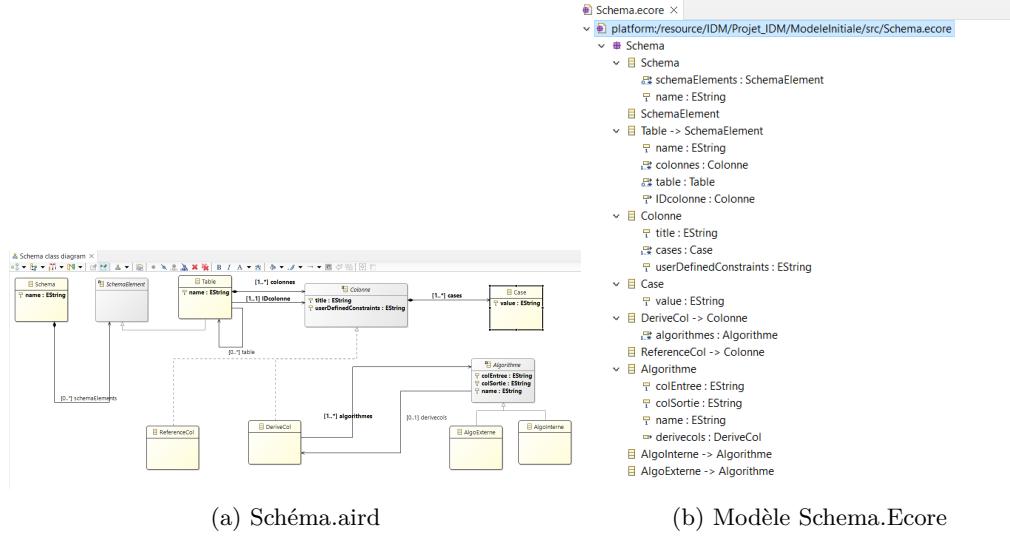


FIGURE. 1 – Le modèle de Schéma de tables

### 3 Crédation des Métamodèles Ecore

#### 3.1 Le métamodèle Schema

##### 3.1.1 Première version

La première itération de notre modèle *Schema.ecore* répond aux exigences de la fonctionnalité *F1*, où l'utilisateur peut composer, sauvegarder et consulter des schémas de table au sein d'une interface conviviale. Chaque table, identifiable par un nom spécifique (attribut **name** de type **EString**), est définie dans une structure hiérarchique avec une relation d'héritage entre la classe "**Table**" et la classe abstraite "**SchemaElement**". Ces tables sont composées de colonnes, chacune ayant des titres/identifiants (attribut **title** de type **EString**), avec une relation d'agrégation entre la classe "**Table**" et la classe abstraite "**Colonne**". Les colonnes, quant à elles, englobent des cases caractérisées par des valeurs (attribut **value**) de type **EString**, établissant ainsi une relation d'agrégation entre les classes "**Colonne**" et "**Case**".

Dans cette structure, les colonnes sont classées en deux catégories distinctes, répondant aux fonctionnalités *F1.1* et *F1.3* : les colonnes dérivées, calculées à l'aide d'algorithmes, et les colonnes références provenant d'un fichier externe (les classes "**DeriveCol**" et "**ReferenceCol**" héritent de la classe "**Colonne**"). Les colonnes dérivées sont associées à des algorithmes, qu'ils soient internes ou externes, ce qui concorde avec la déclaration de la fonction *F1.3*.

Cette version offre ainsi une intégration entre les aspects structuraux et les entités du modèle *Schema.ecore* et les fonctionnalités spécifiées par la *feature F1*, garantissant une expérience utilisateur cohérente dans la création, la sauvegarde et la consultation de schémas de table.

## 3.2 Le métamodèle Algorithme

### 3.2.1 Première version

Dans la première version du modèle 'Algorithme.ecore', nous avons défini une hiérarchie d'éléments pour représenter des algorithmes, leurs ports d'entrée/sortie(F3.2), ainsi qu'une classe spécifique pour les algorithmes basés sur des scripts Java(On a décidé de travailler exclusivement avec Java.).

**Algorithm** : Une classe abstraite représentant tout algorithme générique. Elle a un nom, des ports d'entrée et des ports de sortie(F3.3).

**SmallCalcAlgorithm** : Une classe concrète représentant un algorithme générique existant de petit calcul(F3.4).

**JavaScriptAlgorithm** : Une classe concrète représentant un algorithme basé sur un script Java. Elle a une référence vers JavaFunction qui représente la fonction Java utilisée dans l'algorithme(Des algorithmes externes avec un path donné).

**Port** : Une classe abstraite représentant un port d'entrée ou de sortie. Chaque port a un nom.

**InputPort** : Une classe représentant un port d'entrée, avec une référence vers l'algorithme auquel il est associé.

**OutputPort** : Une classe représentant un port de sortie, avec une référence vers l'algorithme auquel il est associé.

**JavaFunction** : Une classe représentant une fonction Java avec un chemin vers le script Java correspondant.

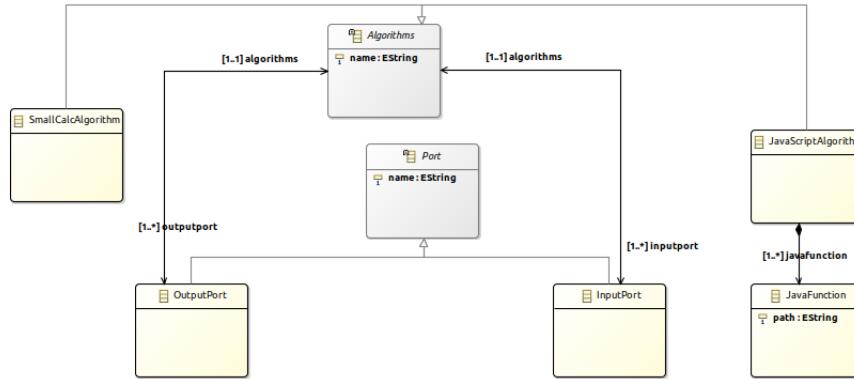


FIGURE. 2 – Le modèle 1 des Algorithmes

### 3.2.2 Deuxième version

En avançant dans les itérations du projet, les fonctionnalités élémentaires de la phase F3 nous ont permis de modifier la première version de notre modèle, en ajoutant une nouvelle classe et en modifiant les relations existantes entre nos différentes classes.

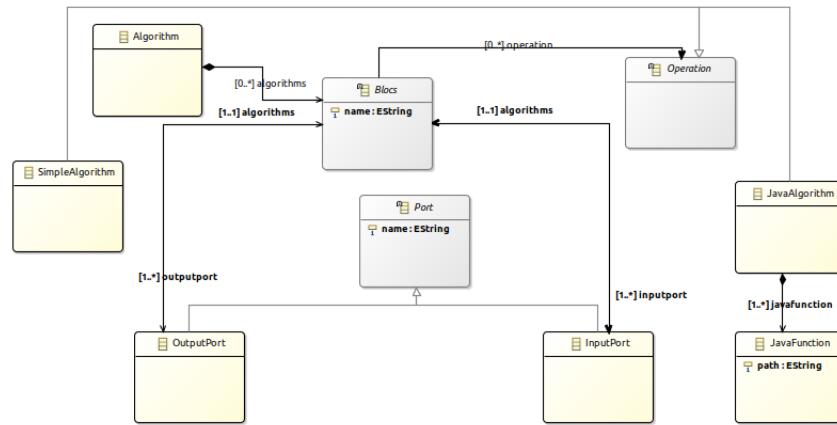


FIGURE. 3 – Le modèle 2 des Algorithmes

**Algorithm** : Notre processus vise à englober les différents blocs liés aux fonctions internes/externes de calcul en vue de concevoir la partie algorithmique.

**Operation** : une classe qui définit le contenu d'un bloc, pouvant être une fonction ou simplement un calcul simple.

En ajoutant de plus un changement de noms (pour quelques classes) pour améliorer la visibilité de nos fonctionnalités liées à la conception de nos différentes classes.

### 3.2.3 Troisième version

La troisième et la dernière version de la modélisation du modèle des Algorithmes a conduit à la suppression des classes InputPort et OutputPort, remplacées par la classe Variable, avec deux sous-classes VariableSortie et VariableEntree définissant la fonction (JavaFunction).

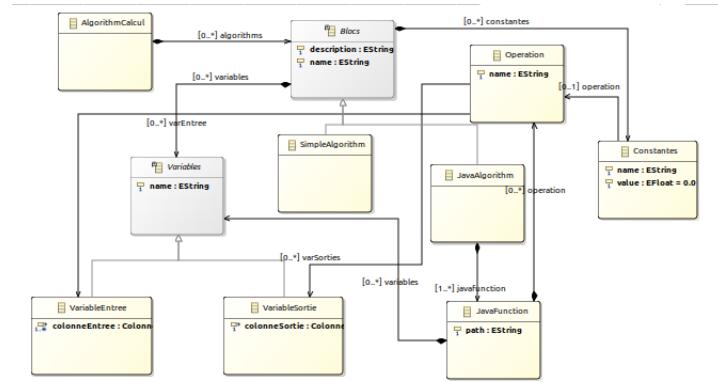


FIGURE. 4 – Le modèle final des Algorithmes

De plus, une classe Constantes a été ajoutée pour modéliser les constantes lors de la définition d'une fonction.

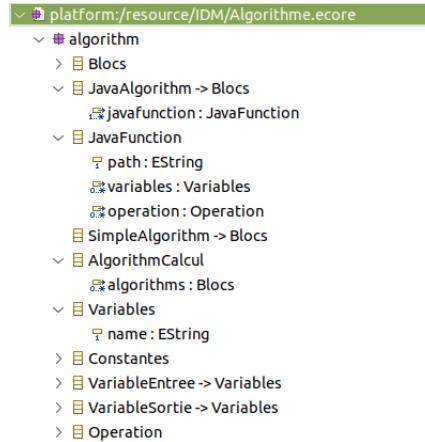


FIGURE. 5 – L'éditeur final des Algorithmes

Pour la classe Operation, liée à la classe JavaFunction, elle définit les opérations simples entre les variables d'entrée d'une fonction.

### Note :

Ces différentes étapes ont engendré de nombreuses discussions provenant de divers horizons et perspectives au sein de notre groupe. De multiples relations ont évolué, des noms et types de classes ont été modifiés dans nos modèles. Le processus de

test et de création des fichiers XMI à chaque itération nous a permis de valider les modifications apportées à chaque étape.

### 3.3 Les contraintes OCL du métamodèle schema

Nous avons opté pour l'utilisation de OCL pour modéliser les contraintes du métamodèles Schema et Algorithme

#### 3.3.1 Explication des contraintes OCL du métamodèle schema

Les contraintes définies dans le langage *OCL* (*Object Constraint Language*) pour le modèle ‘**Schema**’ s’articulent autour de plusieurs aspects de la modélisation des données. Tout d’abord, les contraintes portent sur la validité des noms, imposant une syntaxe spécifique pour les noms de schéma, de table, et de colonne. En outre, la contrainte ‘**TwoTablesHaveDifferentNames**’ garantit que deux tables distinctes dans le schéma ne peuvent pas avoir le même nom. De plus, la contrainte ‘**ColumnHasUniqueTitles**’ assure que toutes les colonnes d’une table ont des titres/identifiants uniques.

Une attention particulière est portée aux colonnes avec des contraintes telles que ‘**ValuesDifferents**’ qui spécifie des valeurs spécifiques pour certaines colonnes (par exemple, la colonne ‘**gender**’ doit contenir uniquement ‘**Male**’ ou ‘**Female**’). De plus, la contrainte ‘**ValuesCoherentes**’ s’assure que des contraintes de cohérence sont respectées, par exemple, la colonne ‘**age**’ ne doit pas contenir de valeurs négatives.

Les contraintes utilisateur sont également prises en compte avec la méthode ‘**evaluateUserConstraint**’ qui permet à l’utilisateur de spécifier des contraintes personnalisées. Ces contraintes peuvent varier, par exemple, de la validation d’intervalles spécifiques à des contraintes dépendant des titres de colonnes.

Enfin, la modélisation s’attache à des aspects spécifiques du schéma, tels que l’unicité de la colonne d’identifiant (‘**TableIDColumnDifferents**’) et la vérification de la cohérence entre les tables et les colonnes associées.

L’ensemble de ces contraintes vise à garantir l’intégrité et la cohérence du modèle de données, en imposant des règles spécifiques aux différentes entités du métamodèle schéma.

(a) Contraintes OCL Schema 1 (b) Contraintes OCL Schema 2 (c) Contraintes OCL Schema 3

FIGURE. 6 – Contraintes OCL Schema

### 3.3.2 Explication des contraintes OCL du métamodèle Algorithmme

Les contraintes OCL sur les algorithmes sont basiques, telles que celles liées à **FunctionSpecification**, exigeant que le chemin de la fonction soit une chaîne de longueur non nulle, **FunctionDifferentName** pour assurer la distinction entre différentes JavaFunctions dans notre algorithme, **VarEntreeSortiePositiveStr** qui impose que le nombre de variables d'entrée/sortie soit supérieur à 0, et **NomAlgo** qui s'applique à la présence de caractères inclus dans le nom d'un objet.

```

1 import 'Algorithme.ecore'
2
3 package algorithm
4
5 context Blocs
6 --
7 -- Example invariant with a custom error message to verify that
8 -- the 'constantes' property of all 'Algorithm::Bloc's instances is non-null
9 --
10 inv NonNull_constants:
11     self.constantes->size()>0
12
13 context JavaAlgorithm
14 --
15 -- Un algorithme en Java spécifie plusieurs fonctions Java en donnant
16 -- un path d'entrée qui doit être non vide.
17 --
18 inv FunctionSpecification:
19     self.javafunction->forAll(f | f.path <> null)
20
21 context AlgorithmCalcul
22 --
23 -- Un algorithme en Java spécifie plusieurs fonctions Java qui doivent
24 -- avoir un nom différent 2 à 2.
25 --
26 inv FunctionDifferentName:
27     self.algorithms->forAll(f1, f2 | f1 <> f2 implies f1.name <> f2.name)
28

```

FIGURE. 7 – Les contraintes 1 OCL sur les algorithmes

```

21@ context AlgorithmCalcul
22@ --
23@ -- Un algorithme en Java spécifie plusieurs fonctions Java qui doivent
24@ -- avoir un nom différent 2 à 2.
25@ --
26@ inv FunctionDifferentName:
27    self.algorithms->forAll(f1, f2 | f1 <> f2 implies f1.name <> f2.name)
28
29@ context JavaFunction
30@ --
31@ -- Une fonction en Java spécifie plusieurs variables d'entrées/sorties dont le cardinal
32@ -- doit être strictement positif.
33@ --
34@ inv VarEntreeSortiePositiveStr:
35    self.variables->select(v | v.oclIsTypeOf(VariableEntree))->size() > 0 and
36    self.variables->select(v | v.oclIsTypeOf(VariableSortie))->size() > 0
37
38@ context AlgorithmCalcul
39@ --
40@ -- Un algorithme de blocs a un nom et une description qui respectent la règle [A-Za-z_][A-Za-z0-9_]*.
41@ --
42@ inv NomAlgo:
43    self.algorithms->forAll(b | b.name.matches('[A-Za-z_][A-Za-z0-9_]+') and b.description <> null)
44
45 endpackage

```

FIGURE. 8 – Les contraintes 2 OCL sur les algorithmes

### 3.3.3 Lien entre les métamodèles Schema.ecore et Algorithme.ecore

**Pour la première version** Pour établir un lien entre la classe abstraite **Colonne** de *Schema.ecore* et la classe abstraite **Port** de *Algorithme.ecore*, on a pu créer une référence entre ces deux classes.

Au niveau de *Schema.ecore*, on dispose d'une classe abstraite nommée **Colonne** et on a ajouté une référence à la classe **Port** de *Algorithme.ecore* dans la classe **Colonne**. Cela a été réalisé en intégrant une propriété de référence dans la classe **Colonne** pointant vers la classe **Port**.

Au niveau de *Algorithme.ecore*, on avait une classe abstraite nommée **Port** et aucune modification particulière n'a été effectuée dans le fichier *Algorithme.ecore* vu que **Port** est correctement défini.

**Pour la deuxième version** On a trouvé un problème avec la référence faite sur le schéma et les algorithmes dans les deux sens. Ainsi, on a essayé de la faire seulement dans le sens des algorithmes, et cela a réussi, car on peut facilement lier les variables d'entrée/sortie aux colonnes.

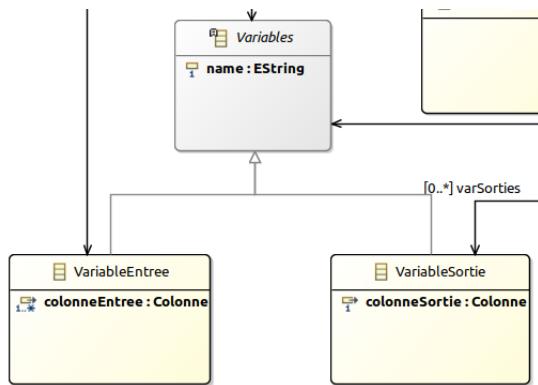


FIGURE. 9 – La liason entre Schéma/Algorithme

## 4 Documentation des algorithmes en HTML par outil acceleo

Notre code Acceleo est une implémentation visant à générer une documentation HTML à partir d'un modèle décrivant des algorithmes en Java. Le code organise l'information en sections clés telles que la description de l'algorithme, les fonctions, les entrées et sorties globales, ainsi que les constantes. Il utilise des boucles et des conditions pour traiter les cas où aucune fonction, entrée, sortie ou constante n'est définie. Le résultat de l'exécution de notre template Acceleo est un fichier HTML encodé en UTF-8, présentant de manière structurée les détails de l'algorithme, rendant la documentation lisible et accessible.

- **Fonctions** : On parcourt les fonctions associées à l'algorithme externe et on génère une table HTML affichant le nom de chaque fonction Java fournie.
- **Entrées et Sorties Globales** : Il fait de même pour les entrées et sorties globales, affichant les noms de chaque port.
- **Constantes** : Il génère une table des constantes définies dans l'algorithme avec leurs noms et valeurs.

Figures de Html fourni

## 5 Représentation des données d'un schème de tables dans un Tableau HTML en utilisant Python

Notre script Python illustre une méthode simple et efficace pour représenter des données tabulaires sous forme de tableau HTML. La fonction `generate_html_table` est conçue pour accepter des données structurées sous forme de dictionnaires, générant ainsi dynamiquement le code HTML nécessaire pour créer un tableau bien formaté. Dans notre code, des données boursières quotidiennes sont utilisées pour démontrer le fonctionnement du script, mais celui-ci peut être facilement adapté à d'autres ensembles de données.

Le code génère un tableau HTML avec une en-tête, défini par les clés des dictionnaires, et des lignes de données correspondant aux valeurs associées. L'esthétique du tableau est améliorée à l'aide de classes CSS, comme indiqué par l'utilisation de "styled-table".

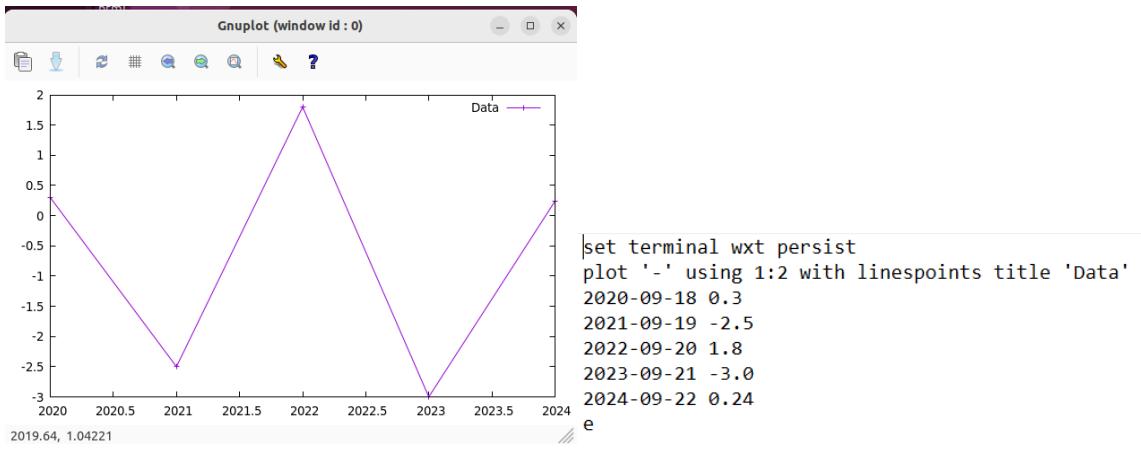
Le tableau HTML généré est écrit dans un fichier appelé "output\_table.html".

Figures ....

## 6 Représentation des données d'un schème de tables en utilisant un script gnuplot

L'utilisation d'un script **Gnuplot** pour représenter graphiquement les données d'un schème de tables offre une approche visuelle puissante et flexible. Dans le script **Gnuplot** fourni, une fonction **Python** nommée `generate_gnuplot_script` est implémentée pour faciliter la création d'un script **Gnuplot** à partir de données tabulaires.

Le script prend en compte deux colonnes spécifiques pour générer un graphique. Dans notre exemple, ces colonnes sont la date et la variation, mais elles peuvent être facilement ajustées pour correspondre à d'autres variables en fonction du schéma de table donné. Le script génère un fichier de script **Gnuplot** (ici nommé "`output_gnuplot_script.txt`") qui peut être exécuté par **Gnuplot** pour produire des graphiques interactifs ou statiques.



(a) Visualisation par Gnuplot des données de Table

(b) Script texte généré par le code Python

FIGURE. 10 – Permettre à l'utilisateur de générer un script gnuplot pour faire des graphiques

## 7 Les Transformations : Java - La librairie

### 7.1 Transformation : CSV - Modèle de schéma

On a mis en place une classe **CSVToSchema** pour répondre à la nécessité d'importer des données destinées à être utilisées ultérieurement dans un programme, en prenant en charge au moins le format CSV. En important les packages nécessaires, notamment ceux d'Eclipse EMF (Eclipse Modeling Framework) pour la manipulation de modèles EMF et la sérialisation en XMI, et en définissant une méthode main dans notre classe **CSVToSchema**, on assure le chargement du package Schema et l'enregistrement de l'extension ".xmi" pour être traitée par la factory **XMIResourceFactoryImpl**.

Figures du code java

Ensuite, on crée un ResourceSetImpl et une ressource EMF (le modèle **Schema**) avec un nom défini par l'utilisateur. On sollicite également l'utilisateur pour entrer le nom du schéma et le nom de la table via la console, en utilisant la classe **Scanner** pour récupérer ces entrées utilisateur. Une fois les informations nécessaires obtenues, le programme crée un objet **SchemaTable** et un objet Table en utilisant la factory **SchemaFactory**. On initialise ensuite les propriétés de la table, on ajoute la table au schéma, puis on ferme le scanner.

Figure du scanner

En poursuivant, on associe les noms de colonnes du fichier CSV à la table, en liant les valeurs de chaque ligne aux cases correspondantes. En dernier lieu, on procède à la sauvegarde du modèle Schema élaboré au format XMI. Pour réaliser cette tâche, on a mis en place diverses méthodes auxiliaires telles que :

- **associateColumnsNamesWithTable** : Cette méthode est responsable de lier les noms de colonnes du fichier CSV à la table. Elle crée des objets de type Colonne pour chaque nom de colonne et les associe à la table du schéma. De plus, elle identifie la colonne représentant l'identifiant (ID) et l'assigne à la propriété IDcolonne de la table.
- **associateColumnsWithTable** : Cette méthode complémentaire est dédiée à l'association des valeurs de chaque ligne aux cases correspondantes. Elle crée des objets de type Case pour chaque valeur de cellule et les associe à la colonne respective de la table. Ainsi, elle établit une relation entre les données brutes du CSV et la structure de la table dans le modèle EMF Schema.
- **readCSV** : Cette méthode est essentielle pour extraire les données du fichier CSV. Elle prend en paramètre le chemin du fichier CSV, utilise un objet `BufferedReader` pour lire le fichier ligne par ligne, et la méthode `readLine()` pour obtenir chaque ligne du fichier. Elle divise ensuite chaque ligne en une liste de chaînes en utilisant la virgule comme séparateur (correspondant à la structure d'un fichier CSV) et stocke ces listes dans une liste globale appelée `records`. Ainsi, cette méthode prépare les données CSV pour le traitement ultérieur.
- **processCSV** : Cette méthode est dédiée au traitement des données extraites du fichier CSV. Elle utilise les informations structurées dans la liste `records` par la méthode `readCSV`. Pour chaque ligne de données, elle appelle les méthodes `associateColumnsNamesWithTable` et `associateColumnsWithTable` pour associer les noms de colonnes à la table et les valeurs aux cases correspondantes. Cette méthode joue un rôle central dans la création du modèle EMF Schema à partir des données du fichier CSV.

Figure de methode associateColumnsNamesWithTable et associateColumnsWithTable

En récapitulant, notre programme java, de transformation/conversion de texte (travaillant avec des fichiers CSV), génère un modèle EMF Schema à partir d'un fichier CSV en donnant à l'utilisateur la possibilité de spécifier le nom du schéma, le nom de la table, et en associant les colonnes ainsi que les valeurs du fichier CSV à des éléments du modèle EMF.



FIGURE. 11 – Le modèle de Schéma de tables créé par la transformation JAVA à partir du fichier .csv

## 7.2 Transformation : Modèle de schéma - CSV

En utilisant les outils de Java, une transformation de notre modèle vers le format CSV est effectuée. En parcourant les différentes colonnes, nous stockons les noms dans la première ligne de notre fichier de sortie 'output.csv', puis parcourons case par case pour stocker les valeurs dans leur colonne respective.

	Standard	Standard	Standard
1	max	min	moyenne
2	'13666'	'5004'	
3			
4	'12888'	'55499'	
5			

FIGURE. 12 – Le génération du Java '.csv'

Le programme Java ci-dessous utilise le package EMF (Eclipse Modeling Framework) pour manipuler notre modèle. Nous chargeons le modèle depuis un fichier XMI, puis parcourons les éléments du schéma, convertissant les tables en fichiers CSV. Pour chaque colonne, les en-têtes sont écrits dans le fichier CSV, suivis des valeurs de chaque case. Le fichier de sortie est généré avec succès, facilitant ainsi la représentation tabulaire de notre modèle dans un format CSV.

```

private static void tableToCsv(Table table) throws IOException {
    // Ecriture des en-têtes de colonnes
    for (Colonne colonne : table.getColonnes()) {
        writeColumnHeaderToCsv(colonne);
    }
    writer.write("\n");

    // Ecriture des valeurs des colonnes
    int numRows = table.getColonnes().stream()
        .mapToInt(colonne -> colonne.getCases().size())
        .max()
        .orElse(0);

    for (int i = 0; i < numRows; i++) {
        for (Colonne colonne : table.getColonnes()) {
            writeColumnValueToCsv(colonne, i);
        }
        writer.write("\n");
    }
}

private static void writeColumnHeaderToCsv(Colonne colonne) throws IOException {
    String columnName = colonne.getTitle();
    writer.write(columnName + ",");
}

private static void writeColumnValueToCsv(Colonne colonne, int rowIndex) throws IOException {
    if (rowIndex < colonne.getCases().size()) {
        String value = String.valueOf(colonne.getCases().get(rowIndex).getValue());
        writer.writeValue(value + ",");
    } else {
        // Ajouter une valeur vide si la colonne n'a pas de valeur à cet indice
        writer.write("\n");
    }
}

```

FIGURE. 13 – Extrait de la transformation modèle2CSV

## 8 Manipulation du calcul des résultats

Pour la partie de calcul, nous avons opté pour une analyse de nos modèles en utilisant l'outil ATL (Atlas Transformation Language) en raison de sa puissance et de sa flexibilité dans la transformation de modèles EMF. L'étape consiste à fournir nos deux modèles en tant qu'entrées, le schéma de table initial et nos algorithmes. En sortie, nous obtenons le schéma de table enrichi des résultats calculés.

```

-- Règle pour remplir les valeurs des colonnes de type VariablesSortie
rule fillVariableSortie {
    from
        algo : algoJavaAlgorithm,
        javaFunction : algoJavaFunction,
        sortie : javaFunctionVariablesSortie,
        op : algoFunctionOperation
    to
        sortie : schemasortant!DeriveCol {
            title <- sortie.title;
            cases <- sortie.cases->collect(
                c | applyOperations(algo, javaFunction, c, op)
            );
            userDefinedConstraint <- sortie.userDefinedConstraint;
            nombreLignes <- sortie.nombreLignes;
        }
    }

helper def : applyOperations (algo : algoJavaAlgorithm, javaf : algoJavaFunction, c : schemasortant!Case, op : javaf!Operation) : schemasortant!Case =
    if javaFunction.path.size() = 0
    then transformCasec, vSortie, op, javaf, op.varEntree, op.varEntree
    else appliquerFonctc, vSortie, op, javaf, op.varEntree, op.varEntree
;

helper def : transformCase (c : schemasortant!Case, op : algo!Operation, javaf : algoJavaFunction, var1 : javaf!VariablesSorties) : schemasortant!Case =
    schemasortant!Case {
        value <- c.value;
        if op.name = 'ADD' then var1 + var2
        else if op.name = 'DIV' then var1 / var2
        else if op.name = 'MULT' then var1 * var2
        else if op.name = 'SOUST' then var1 - var2
        else c.value
    };

```

FIGURE. 14 – Extrait du calcul ATL

Le code ATL comprend plusieurs règles. La première règle, linkColumns, lie les colonnes dérivées du modèle d'algorithme au schéma initial. La deuxième règle, fillVariableSortie, remplit les valeurs des colonnes de type VariablesSortie en appliquant les opérations spécifiées dans les algorithmes. De plus, des fonctions auxiliaires, telles que applyOperations, transformCase, et appliquerFonct, sont utilisées pour appliquer

les opérations et transformer les cases en fonction du type d'opération.

L'entrée principale est définie par la règle main, où un algorithme et un schéma initial sont fournis en entrée, et le schéma final enrichi est produit en sortie.

La partie du projet impliquant le code ATL(choisi) n'a pas été menée à terme en raison de divers défis techniques rencontrés au cours de son développement. La complexité des opérations définies dans les algorithmes, couplée à la variété des types de données manipulées, a posé des obstacles significatifs dans la création de règles de transformation précises. De plus, des limitations de temps ont entravé la réalisation de tests exhaustifs pour valider la correspondance entre les résultats calculés et le schéma de table général. Ces complexités et contraintes ont rendu nécessaire une réévaluation approfondie de l'approche et des ressources nécessaires pour achever cette composante essentielle du projet.

## 9 Transformation à Xtext

Cette représentation graphique en Xtext offre une manière concise et structurée de spécifier des schémas tabulaires avec des tables, des colonnes et des cases.

```
grammar schema.Xtext with org.eclipse.xtext.common.Terminals

generate xtext "http://www.Xtext.schema"

Schema : 'schema' name=ID '{'
    tables+=Table*
    '}';
}

Table:
    'Table' name=ID 'contains' '{'
        colonnes+=Colonne*
    '}';
}

Colonne:
    'colonne' name=ID ':' '*' cases+=Case* '|';
}

Case:
    'case' number=INT ':' texte=STRING '|';
}
```

FIGURE. 15 – Transformation Xtext

La grammaire commence par définir un "Schema" avec un nom, suivi d'accolades encadrant la définition du schéma. À l'intérieur du schéma, on peut déclarer plusieurs "Tables", chacune ayant un nom et une liste de colonnes. Chaque "Colonne" est déclarée avec un nom et une liste de "Case". Chaque "Case" représente une cellule du tableau avec un numéro et un texte.

## 10 L'éditeur graphique : Sirius

Le code Sirius fourni définit une représentation graphique pour un langage de modélisation dédié dans le cadre d'algorithmes. Ce DSL est utilisé pour spécifier des opérations, des variables d'entrée et de sortie, ainsi que des constantes au sein d'algorithmes. La description Sirius utilise un modèle graphique pour permettre aux utilisateurs de créer, visualiser et éditer ces éléments dans un environnement graphique.

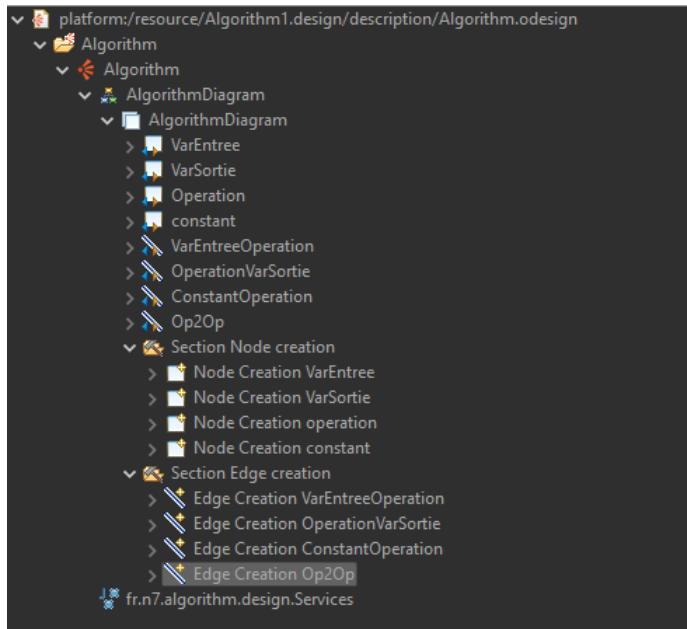


FIGURE. 16 – Editeur graphique Sirius

Le code Sirius commence par définir un groupe de description avec un nom ("Algorithm") et une version. Ensuite, il spécifie plusieurs représentations graphiques ("DiagramDescription") sous le nom "AlgorithmDiagram", chacune associée à des classes de domaine spécifiques telles que "VariableEntree", "VariableSortie", "Operation", et "Constantes".

Pour chaque type d'élément graphique, des styles sont définis pour déterminer l'apparence visuelle, tels que la couleur du bord, la couleur du label, et la couleur de fond...

Les sections "Node creation" et "Edge creation" définissent des outils pour créer des nœuds et des bords graphiques correspondant aux différents éléments du DSL,

tels que les variables d'entrée, de sortie, les opérations et les constantes. Ces outils utilisent des opérations spécifiques pour effectuer des changements de contexte et créer des instances dans le modèle sous-jacent.

## 11 Entraves et Challenges

Lors de la réalisation de ce projet, nous avons beaucoup appris, et cela a été influencé par différentes contraintes rencontrées à chaque étape. Sans oublier les questions du début du projet, telles que "Comment manipuler les fichiers CSV des données ? Doit-on transformer le fichier CSV en XMI ? Est-il nécessaire d'implémenter un code qui convertit notre modèle en texte en utilisant Xtext ?" afin de prendre des décisions éclairées. Parmi ces contraintes, nous pouvons citer :

- **La réalisation du méta-modèle final :** La réalisation d'un méta-modèle final pour faciliter le lancement des étapes suivantes a entraîné une perte de temps significative en raison de multiples modifications apportées aux méta-modèles.
- **Les erreurs fréquentes d'Eclipse :** Les erreurs fréquentes d'Eclipse ont été un véritable cauchemar, nous faisant perdre du temps à chaque occurrence. Une partie conséquente de notre temps a été dédiée à résoudre ces erreurs d'Eclipse, retardant ainsi la finalisation du projet.

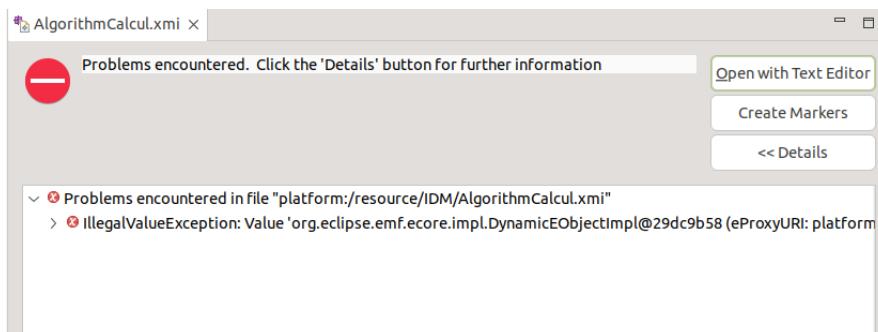


FIGURE. 17 – Des erreurs d'ouverture - 'xmi'

- **Les différentes versions d'Eclipse et la gestion du travail par membre :** Les différentes versions d'Eclipse utilisées initialement et la concaténation du travail effectué par chaque membre ont généré de nombreux problèmes. La gestion de cette diversité a nécessité des efforts supplémentaires pour assurer la

cohérence et la compatibilité du travail de l'équipe.

```

locals: { 'java/lang/Object', 'java/lang/Object', 'org/eclipse/emf.ecore/
EPackage' }
stack: { integer }
Stackmap Frame:
bci: @79
flags: {}
locals: { 'java/lang/Object', 'Schema/impl/SchemaPackageImpl', 'java/lang/
Object' }
stack: { }
Bytecode:
0x0000000: b200 1899 0011 b200 4012 1db9 0046 0200
0x00000010: c000 05b0 b200 4012 1db9 004a 0200 4b2a
0x00000020: c100 0199 000a 2ac0 0001 a700 0abb 0001
0x00000030: 59b7 004e 4c04 b300 18b2 0040 1250 b900
0x00000040: 4602 004d 2cc1 0052 9900 072c a700 06b2
0x00000050: 0054 c000 524e 2bb6 0059 2db6 005c 2bb6
0x00000060: 005d 2db6 0060 2bb6 0061 b200 4012 1d2b
0x00000070: b900 6403 0057 2bb6
Stackmap Table:
same_frame(@20)
append_frame(@45, Object[#111])
same_locals_1_stack_item_frame(@52, Object[#111])
append_frame(@79, Object[#1], Object[#111])
same_locals_1_stack_item_frame(@82, Object[#111])
at Schema.SchemaPackage.<clinit>(SchemaPackage.java:59)

```

FIGURE. 18 – Des erreurs de compilation/exécution JAVA

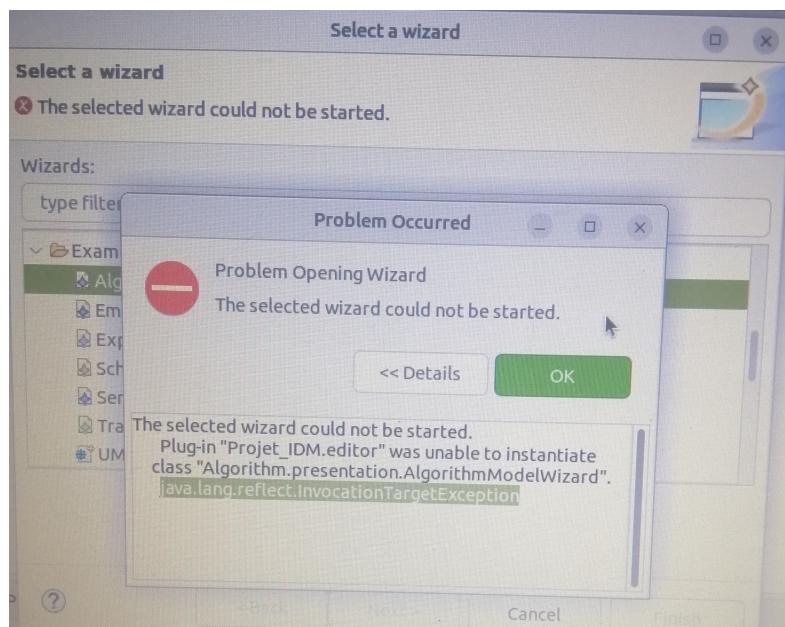


FIGURE. 19 – Des erreurs de Sirius

Bien que nous ayons été confrontés à un ensemble de contraintes, notre équipe

a consacré des efforts considérables pour surmonter ces défis. Nous avons investi du temps et de l'énergie dans la résolution de problèmes spécifiques, tels que l'optimisation du méta-modèle, la gestion des erreurs fréquentes d'Eclipse, et la coordination des différentes versions de l'environnement de développement. Ces efforts concertés ont permis de maintenir une dynamique positive malgré les obstacles, et nous avons réussi à progresser de manière significative sur le projet. Cette expérience a renforcé notre capacité à réagir efficacement face aux imprévus, démontrant ainsi notre engagement envers la réussite du projet.

## 12 Gestion Dynamique du Projet

Nous avons choisi de mettre en œuvre les méthodes agiles, en particulier la méthodologie Scrum, pour gérer notre projet. Khadija Akkar était la scrum-master.

### 12.1 Sprint 1 : Metamodèles et Contraintes OCL

#### Objectif du Sprint

Définir les métamodèles pour les schémas de table et les algorithmes, ainsi que les contraintes OCL associées.

#### Tâches

- Élaborer le métamodèle pour représenter les schémas de table à l'aide d'EMF/Ecore.
- Spécifier les contraintes OCL pour garantir la validité des schémas de table.
- Créer le métamodèle pour représenter les algorithmes et leurs composants.
- Établir des contraintes OCL pour garantir la cohérence des algorithmes définis.

#### Résultats

- Métamodèles définis pour les schémas de table et les algorithmes.
- Contraintes OCL spécifiées.

## 12.2 Sprint 2 : Documentation HTML des Algorithmes avec Acceleo

### Objectif du Sprint

-Mettre en œuvre la documentation HTML des algorithmes en utilisant Acceleo.

### Tâches

-Génération de Documentation HTML avec Acceleo.

### Résultats Attendus

-Documentation HTML générée automatiquement pour chaque algorithme.

## 12.3 Sprint 3 : Représentation des Données en Tableau HTML avec Python

### Objectif du Sprint

-Implémenter la représentation des données d'un schéma de table dans un tableau HTML en utilisant Python.

### Tâches

-Écriture du Script Python pour la Représentation HTML : Développer un script Python qui prend en entrée des données conformes à un schéma de table et génère un tableau HTML.

-Tests et Validation : Effectuer des tests pour s'assurer que le script fonctionne correctement avec différentes données et schémas.

### Résultats

-Script Python capable de générer un tableau HTML à partir de données de schéma de table.

## 12.4 Sprint 4 : Représentation des Données avec gnuplot

### Objectif du Sprint

Mettre en place la représentation des données d'un schéma de table à l'aide d'un script gnuplot.

### Tâches

-Écriture du Script gnuplot : Élaborer un script gnuplot qui prend en entrée des données conformes à un schéma de table et génère une représentation graphique.

-Tests et Validation : Valider le script gnuplot en utilisant différentes données et schémas.

### Résultats

-Script gnuplot capable de représenter graphiquement les données d'un schéma de table.

## 12.5 Sprint 5 : Transformations CSV - Modèle de Schéma et Vice Versa en Java

### Objectif du Sprint

Implémenter les transformations entre les fichiers CSV et les modèles de schéma.

### Tâches

-Transformation CSV vers Modèle de Schéma en Java : Écrire un programme Java qui prend un fichier CSV en entrée et crée un modèle de schéma correspondant.

-Transformation Modèle de Schéma vers CSV en Java : Développer un programme Java qui prend un modèle de schéma en entrée et génère un fichier CSV.

### Résultats

Transformations CSV vers Modèle de Schéma et vice versa implémentées en Java.

## 12.6 Sprint 6 : Transformation à Xtext et Éditeur Graphique Sirius

### Objectif du Sprint

-Intégrer Xtext pour la définition de langages spécifiques au domaine (DSL) et développer un éditeur graphique avec Sirius pour la visualisation et l'édition des modèles.

### Tâches

-Création d'un Éditeur Textuel avec Xtext : Développer un éditeur textuel basé sur Xtext pour permettre aux utilisateurs de définir les schémas.

-Création d'un Éditeur Graphique avec Sirius : Utiliser Sirius pour créer un éditeur graphique pour la visualisation et l'édition des modèles de schéma de table.

### Résultats

-Éditeur textuel Xtext intégré dans Eclipse.  
-Éditeur graphique Sirius non fonctionnel.

## 12.7 Outils utilisés

Nous avons choisi **GitHub** comme plateforme principale pour la publication et la gestion de notre code source, garantissant ainsi une traçabilité et une collaboration optimale entre les membres de l'équipe.

Parallèlement, l'utilisation de **Trello** a été essentielle pour planifier et suivre nos tâches de manière visuelle, offrant une vue d'ensemble cohérente de l'avancement du projet.

Les réunions hebdomadaires, et même une fréquence accrue pendant les périodes de vacances, ont été organisées de manière efficiente grâce à des outils de visioconférence tels que **Zoom** et **Discord**. Ces initiatives ont contribué de manière significative à la cohésion de l'équipe et à la réussite de notre projet.

"CSV 2 schema"
 safae1202 pushed 1 commit to <a href="#">main</a> • 456af74...0c7aefa • 8 days ago
*****
 AkkarKhadija24 pushed 1 commit to <a href="#">main</a> • b705853...456af74 • 8 days ago
OCL-Manip
 AkkarKhadija24 pushed 1 commit to <a href="#">main</a> • 1d1ad23...b705853 • 9 days ago
*****
 AkkarKhadija24 pushed 1 commit to <a href="#">main</a> • 828522c...1d1ad23 • 13 days ago
Algorithme-last-commit
 AkkarKhadija24 pushed 1 commit to <a href="#">main</a> • 9c26d91...828522c • 13 days ago
final version algorithm
 WissalABOUMEJD pushed 1 commit to <a href="#">main</a> • 4b1c5e6...9c26d91 • 15 days ago
avancement algorithme
 WissalABOUMEJD pushed 1 commit to <a href="#">main</a> • 9f11423...4b1c5e6 • 15 days ago
Modification algorithme
 WissalABOUMEJD pushed 1 commit to <a href="#">main</a> • 020538f...9f11423 • 17 days ago
"Schema-Algo liaison"
 safae1202 pushed 1 commit to <a href="#">main</a> • c6b737a...020538f • 18 days ago
"SchemaOCL"
 safae1202 pushed 1 commit to <a href="#">main</a> • 0c46d77...c6b737a • 18 days ago

FIGURE. 20 – Extrait de nos activités sur Github

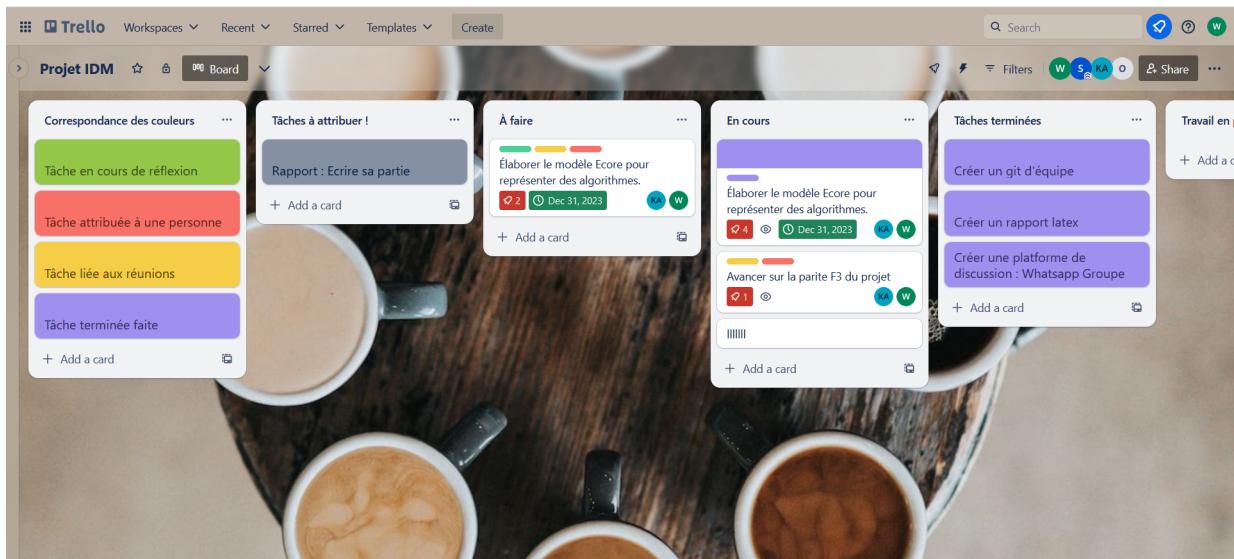


FIGURE. 21 – Extrait 1 de nos activités sur Trello

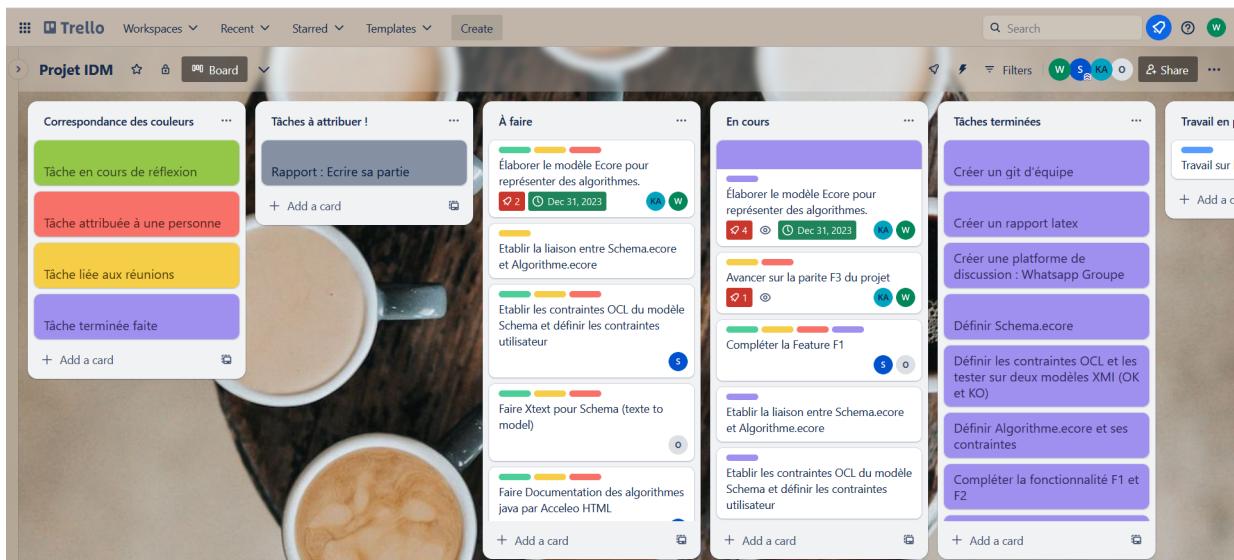


FIGURE. 22 – Extrait 2 de nos activités sur Trello

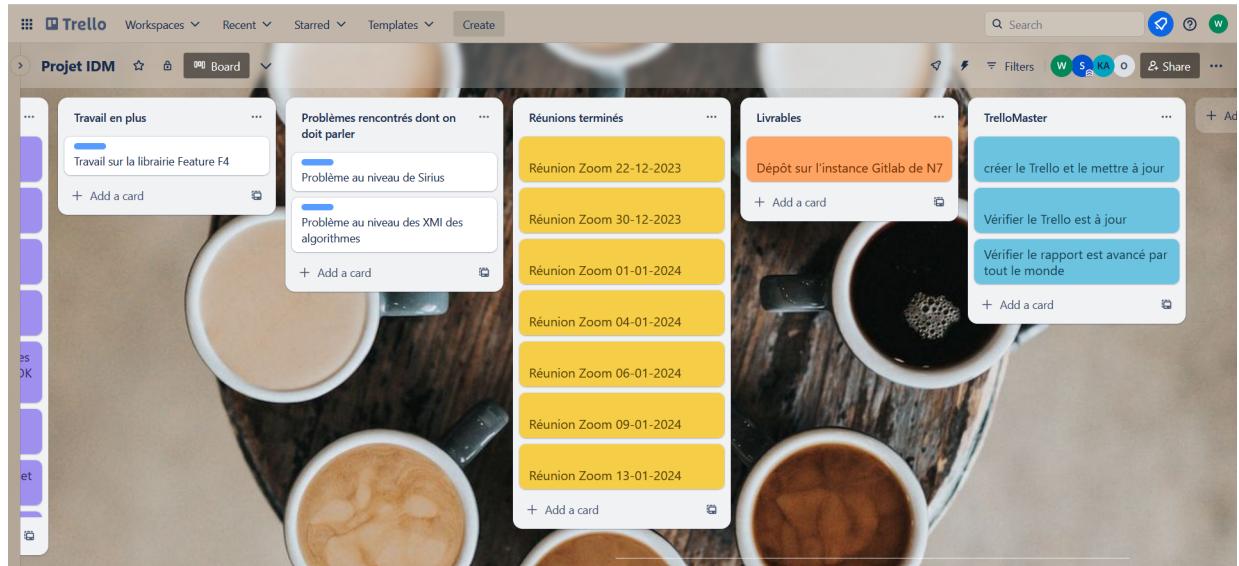


FIGURE. 23 – Extrait 2 de nos activités sur Trello

## 13 Conclusion

La conduite de ce projet IDM a constitué une expérience significative, mêlant habilement les aspects de gestion de projet et les défis techniques inhérents à son développement. Notre équipe de quatre personnes a démontré une compétence dans la résolution de problèmes, la prise de décisions stratégiques, et la capacité à naviguer avec succès à travers divers obstacles techniques.

Sur le plan de la gestion de projet, notre collaboration a été caractérisée par une répartition judicieuse des tâches, le respect des échéances et une communication régulière. Nous avons établi une coordination efficace au sein de l'équipe, défini clairement les rôles et responsabilités, et organisé des réunions régulières pour assurer une progression harmonieuse du projet.

Du point de vue technique, la résolution des défis rencontrés a été l'occasion de perfectionner nos compétences individuelles et collectives. La mise en œuvre des fonctionnalités spécifiques a requis une expertise approfondie dans divers domaines technologiques. Les décisions architecturales, le choix des technologies et la gestion des ressources ont été prises avec rigueur et discernement.

En conclusion, ce projet ne se limite pas à une simple réalisation technique, mais représente une étape significative dans notre développement professionnel et académique. Les enseignements tirés de cette expérience nous ont permis de renforcer nos compétences, de prendre des décisions éclairées et de mieux comprendre les réalités de la gestion de projet. Ces compétences acquises contribueront positivement à notre trajectoire professionnelle future. Nous sommes reconnaissants de cette opportunité qui a été une véritable source d'enrichissement et de croissance pour notre équipe.

En conclusion, notre projet IDM axé sur la création d'un environnement de calcul domaine-spécifique constitue une étape significative dans notre développement professionnel et académique. Les enseignements tirés ont renforcé nos compétences techniques, affiné notre capacité à prendre des décisions éclairées et approfondi notre compréhension des défis inhérents à la gestion de projets. Ces compétences nouvellement acquises contribueront positivement à notre trajectoire professionnelle future, et nous sommes reconnaissants de cette opportunité qui a été une source d'enrichissement et de croissance pour notre équipe.