



**ECOLE NATIONALE SUPÉRIEURE D'ÉLECTROTECHNIQUE, D'ÉLECTRONIQUE,  
D'INFORMATIQUE, D'HYDRAULIQUE ET DES TÉLÉCOMMUNICATIONS**

Département : Sciences numériques

Parcours : Systèmes Logiciels

---

## **PROJET D'IDM : GESTION DE STOCKS**

---

*Soutenu par :*

Imane EL AYBOUDI  
Salwa ALIMOUSSA  
Marwane KARAOUI  
Mohamed Amine BENKIA  
Kossila CHABANE

*Encadré par :*

Pr. PANTEL MARC  
Pr. DUPONT GUILLAUME

Année universitaire 2024-2025

---

## **Contexte du projet :**

L'informatique joue un rôle central dans la gestion et le traitement des données, mais les outils traditionnels comme les tableurs ne sont pas toujours adaptés à des utilisateurs non-experts en informatique. Ils peuvent être complexes à manipuler et manquent de solutions automatisées pour des tâches spécifiques.

Ce projet donc propose une approche innovante pour résoudre ce problème en utilisant des technologies modernes comme Eclipse et EMF. L'objectif est de permettre à des experts de leur domaine de créer facilement des outils sur mesure pour structurer, analyser et exploiter leurs données, sans avoir besoin de trop de détails en technique. Cette solution vise à rendre le traitement des données plus accessible et efficace.

# Table de matières :

<b>Table des figures</b>	<b>4</b>
<b>1 Conception du métamodèle</b>	<b>6</b>
1.1 Description du Métamodèle . . . . .	6
1.2 Crédation d'un modèle conforme au métamodèle . . . . .	6
1.2.1 1.Application . . . . .	6
1.3 Diagramme Ecore du Métamodèle . . . . .	7
<b>2 Développement du projet</b>	<b>9</b>
2.1 Crédation d'un modèle . . . . .	9
2.2 Chargement du modèle en un objet Java pour le manipuler dans les parties suivantes . . . . .	11
2.3 Traduction et évaluation des expressions des colonnes dérivées . . . . .	11
2.4 Transformation des fichiers CSV en un Hashmap de type Json . . . . .	11
2.5 Vérification de la validité des données importées par l'utilisateur . . . . .	11
2.6 Application du traitement final de l'application sur les données . . . . .	12
2.7 Interface utilisateur . . . . .	12
2.7.1 Import d'une table . . . . .	12
2.7.2 Caluler une nouvelle colonne . . . . .	13
2.7.3 Sauvegarder les résultats . . . . .	13
2.7.4 Scénario d'utilisation . . . . .	13
2.8 Modèle de syntaxe graphique avec Sirius . . . . .	14
2.9 Contrainte avec Java. . . . .	18
2.10 Modèle XText . . . . .	20
2.11 Conclusion . . . . .	22

# Table des figures

1.1	Diagramme Ecore du métamodèle.	8
2.1	Modèle d'instanciation du métamodèle	10
2.2	Interface utilisateur	12
2.3	clic sur le bouton importer	13
2.4	Fichier des données à importer pour effectuer les calculs.	13
2.5	clic sur le bouton calculer	14
2.6	Fichier des résultats après application des calculs	14
2.7	Clic sur le bouton Télécharger	14
2.8	Arborescence des éléments.	15
2.9	Vue graphique du modèle.	15
2.10	Section des Outils.	16
2.11	Palette et vue graphique.	17
2.12	.	18
2.13	.	18
2.14	.	18
2.15	.	19
2.16	.	19
2.17	.	19
2.18	Projet Xtext	20
2.19	Message de création du modèle	20
2.20	Métamodèle généré par Xtext	21
2.21	Modèle de validation	22
2.22	Message de Validation	22

## Introduction Générale

Dans de nombreux domaines, les professionnels doivent manipuler des données complexes sans forcément avoir de solides compétences en informatique. Les outils comme les tableurs sont pratiques, mais souvent trop rigides ou compliqués pour répondre à des besoins spécifiques.

Ce projet propose une solution pour simplifier ces tâches en permettant à chaque utilisateur de créer des outils personnalisés. Grâce aux technologies modernes d'Eclipse et EMF, l'objectif est de concevoir une plateforme où l'on peut facilement structurer des données, automatiser des calculs et vérifier leur cohérence. Cette démarche vise à mettre la puissance de l'informatique à la portée de tous, en transformant des idées métiers en outils concrets, simples et efficaces.

# Chapitre 1

## Conception du métamodèle

### 1.1 Description du Métamodèle

### 1.2 Crédation d'un modèle conforme au métamodèle

Le métamodèle définit une structure permettant de gérer des **tables de données** composées de différents types de colonnes, ainsi que des mécanismes pour calculer ou référencer des données de manière flexible. Il repose sur plusieurs entités principales qui pour fournir un environnement adapté à la gestion et à l'automatisation des calculs pour une meilleure gestion des stocks.

#### 1.2.1 1.Application

La classe Application : Conteneur global regroupant les tables de l'application.

### 2. Table

La classe Table représente une table de données.

- **Attributs :**
  - id : Identifiant unique de la table .
  - Nom : Nom de la table en chaîne de caractères.
- **Références :**
  - colonnes : Liste de colonnes associées à la table car chaque table peut contenir des colonnes.

### 3. Colonne

La classe Colonne est abstraite et modélise une colonne générique dans une table. Les colonnes peuvent être simples, dérivées ou référencées.

- **Attributs :**
  - id : Identifiant unique de la colonne.
  - Nom : Nom de la colonne.
  - TypeDonnees : Type des données pour indiquer le type permis pour chaque colonnes afin d'éviter les erreurs de calculs.

### 4. ColonneSimple

Une colonne standard qui ne dépend ni d'expressions ni d'autres colonnes. Elle hérite de la classe Colonne.

## 5. ColonneDérivée

Une colonne dont les valeurs sont calculées dynamiquement à partir d'une expression.

— **Références :**

- `expression` : Expression utilisée pour calculer les valeurs.

## 6. ColonneRéférencée

Une colonne qui fait référence à une autre colonne dans une table différente.

— **Références :**

- `table` : Table contenant la colonne référencée.
- `colonne` : Colonne spécifique qui est référencée.

## 7. Expression (Abstraite)

Une expression modélise un calcul. Elle sert de base pour des sous-types comme les expressions binaires ou les accès à des variables.

## 8. ExpressionBinaire

Une expression binaire combine deux opérandes (`opandeG` et `opandeD`) et un opérateur.

— **Références :**

- `opandeG` : Opérande gauche.
- `operateur` : Opération à appliquer (addition, soustraction, etc.).
- `opandeD` : Opérande droite.

## 9. Opérateur

Un opérateur, tel que l'addition ou la multiplication, est défini par une énumération `Operations`.

## 10. AccesVariable

Un type d'expression qui permet d'accéder directement aux valeurs d'une colonne spécifique.

— **Operations** : Enumération des opérations mathématiques possibles (addition, soustraction, multiplication, division).

### 1.3 Diagramme Ecore du Métamodèle

Pour mieux comprendre la structure et les relations entre les différents concepts du métamodèle, nous présentons ci-dessous un diagramme Ecore. Ce diagramme met en évidence les principales entités qu'on a utilisé dans notre projet ainsi que les relations entre elles.

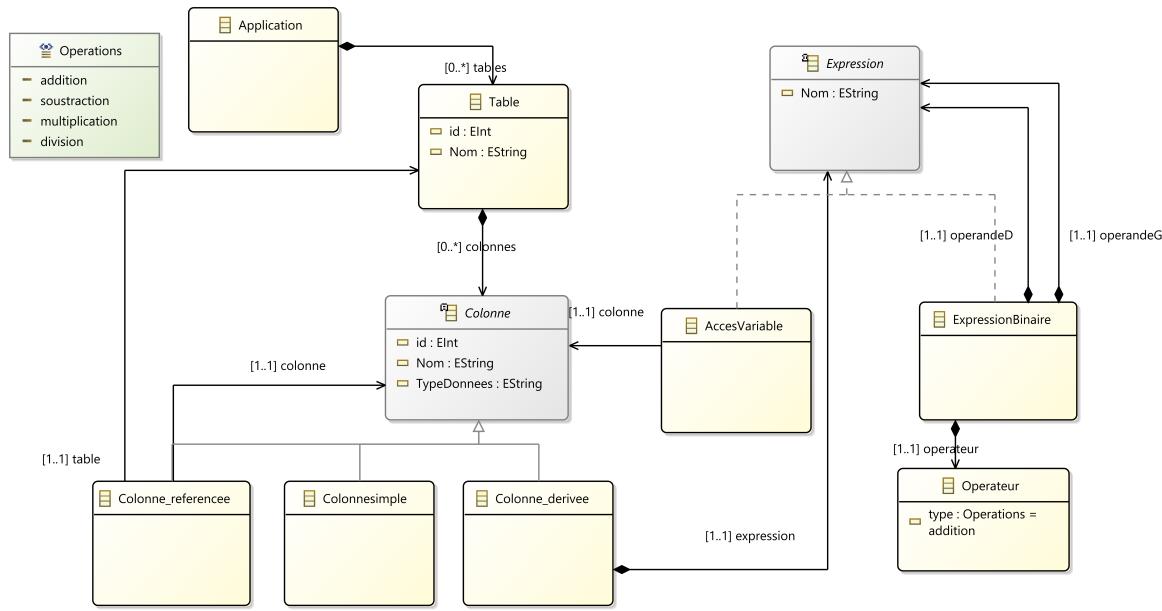


FIGURE 1.1 – Diagramme Ecore du métamodèle.

Le diagramme montre clairement :

- Les associations entre les tables et leurs colonnes.
- Les types de colonnes (simples, dérivées, référencées) et leurs spécificités.
- La gestion des expressions et des opérateurs pour les calculs d'.

## Chapitre 2

# Développement du projet

### 2.1 Crédit d'un modèle

Après avoir créé notre métamodèle, nous avons créé un nouveau modèle composé de deux tableaux : une table **Produit** et une table **Commande**. La table **Produit** est composée de trois colonnes :

- **NOM\_PRODUCT** : qui est de type simple et contient les noms des produits existants au niveau du stock.
- **QUANTITE\_PRODUCT** : qui représente la quantité de chaque produit en stock.
- **PRIX\_UNITAIRE** : qui est une colonne simple contenant le prix unitaire de chaque produit du stock.

Pour la table **Commande**, elle contient cinq colonnes de types différents :

- **NOM\_PRODUCT\_COMMANDE** : de type référence, elle fait référence à la colonne **NOM\_PRODUCT** de la table des produits.
- **PRIXCOMMANDÉ** : qui contient une expression de calcul définie par la formule **PRIXPRODUCT**  $\times$  **QUANTITE\_DU\_PRODUCT\_COMMANDE**.
- **PRIXPRODUCT** : qui contient le prix unitaire de chaque produit au niveau de la commande.
- **NUMEROCOMMANDÉ** : qui contient le numéro de la commande, permettant de distinguer les commandes les unes des autres.

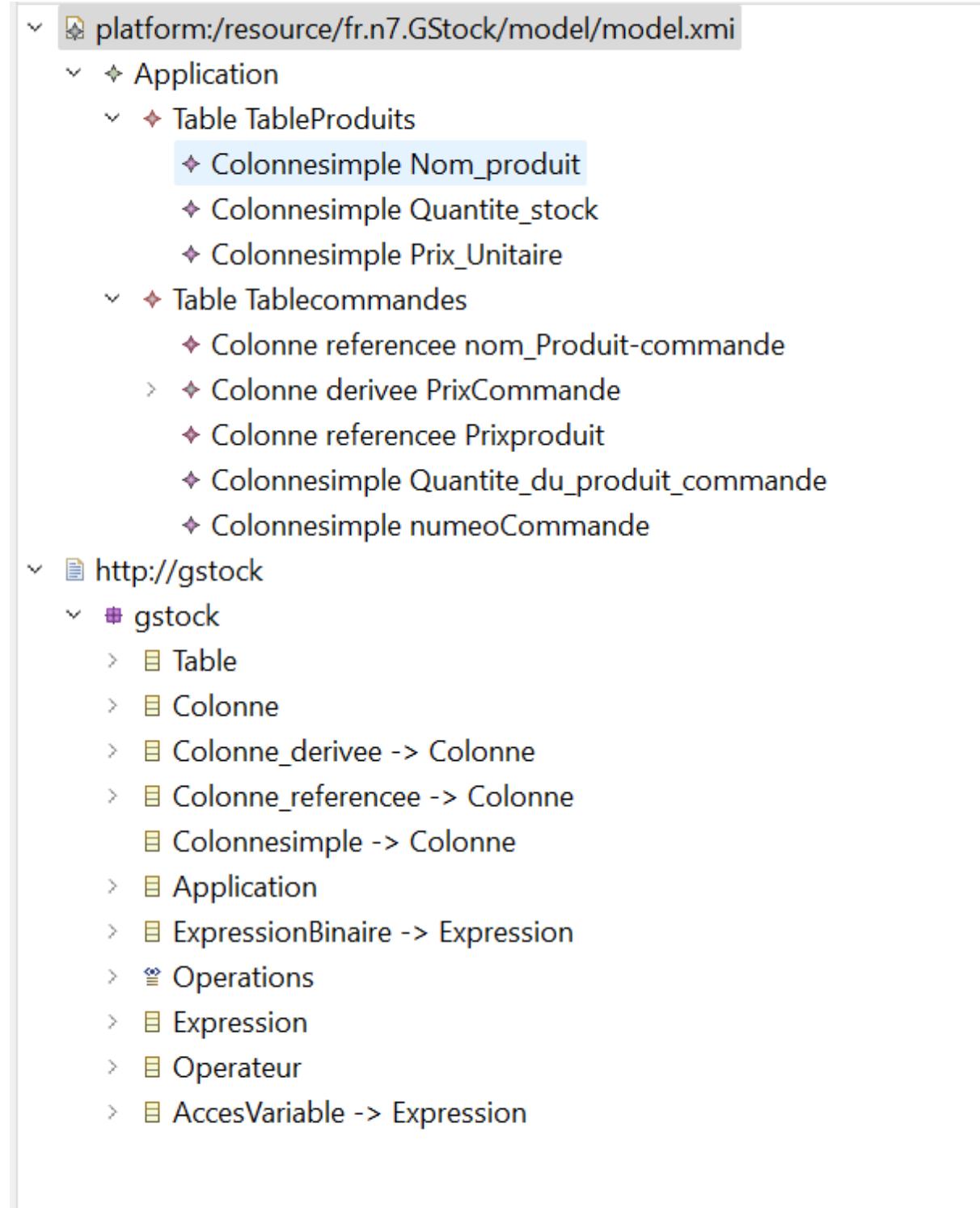


FIGURE 2.1 – Modèle d’instanciation du métamodèle

## 2.2 Chargement du modèle en un objet Java pour le manipuler dans les parties suivantes

Nous avons implémenté une classe ModelLoader pour charger un fichier XMI en un objet Java en utilisant EMF. Le processus commence par l'enregistrement de l'extension xmi avec un gestionnaire de ressources et l'association explicite de l'URI `http://gstock` au métamodèle défini dans GstockPackage. Ensuite, le fichier XMI est chargé dans un ResourceSet, et son contenu est converti en un objet Java de type Application. Ce mécanisme permet d'interpréter les données du modèle XMI en fonction du métamodèle.

## 2.3 Traduction et évaluation des expressions des colonnes dérivées

Nous avons créé un fichier DerivedColumnEvaluator qui parcourt les tables du modèle que nous avons défini comme base de notre application. Ce fichier extrait les expressions associées aux colonnes dérivées, effectue les calculs nécessaires sur les données fournies par l'utilisateur final, et ajoute les résultats directement dans la structure de données en mémoire (HashMap) contenant les tables de l'utilisateur.

Pour réaliser ces opérations :

- La classe parcourt l'objet Java représentant le modèle.
- Elle identifie les colonnes dérivées et extrait leurs expressions.
- Ces expressions sont converties en logique Java pour être appliquées sur les données utilisateur.
- Les tables des données fournies par l'utilisateur sont ensuite modifiées directement en y ajoutant les colonnes dérivées calculées.

## 2.4 Transformation des fichiers CSV en un Hashmap de type Json

Nous avons créé une classe Java SVHandler pour convertir un fichier CSV en un hashmap de format JSON. Pour faire cela, nous avons parcouru le fichier JSON ligne par ligne, puis nous avons transformé ce fichier en une hashmap qui a pour chaque clé le nom d'une table et pour chaque valeur une liste de lignes. Cette liste est une hashmap qui contient le nom de chaque colonne comme clé et la valeur est la donnée dans la cellule correspondante de cette colonne. Après, nous avons utilisé la méthode setPrettyPrinting() de Gson pour générer un JSON bien formaté et lisible qu'on peut manipuler après pour faire des calculs . Après cette transformation, nous avons retourné le hashmap correspondant au fichier CSV du début de la transformation.

## 2.5 Vérification de la validité des données importées par l'utilisateur

Pour assurer la validité des calculs que nous allons faire sur les données que l'utilisateur va fournir on a jouté une classe CSVValidator qui parcourt le fichier CSV ligne par ligne et vérifie si toutes les valeurs présente au niveau du fichier sont positives, car on ne peut pas travailler avec des valeurs négative par exemple pour le prix des produits ou bien les quantités des produits donc pour faire des calculs correct il faut assurer que les valeurs fournit par l'utilisateur sont valides et pour les calculs non valide comme utiliser deux colonnes qui n'ont pas des valeurs entières on a géré cela aussi au niveau des contraintes java du métamodèle.

## 2.6 Application du traitement final de l'application sur les données

Au niveau de notre application, nous avons donné la possibilité à l'utilisateur d'ajouter un traitement qu'il souhaite appliquer en complément des calculs automatiques effectués à partir du modèle XMI et alors, il faut qu'il choisisse juste les colonnes qu'il veut utiliser au niveau du calcul et l'opérateur entre ces deux colonnes. Pour faire cela, nous avons définie une classe CSVProcessor au niveau de laquelle nous avons créé deux méthodes la première addDerivedColumnsToTables qui parcourt les données et ajoute le calcul des colonnes dérivées basées sur le modèle et la deuxième méthode addDynamicColumn qui Ajoute une colonne dynamique calculée à partir des colonnes sélectionnées.

## 2.7 Interface utilisateur

Nous avons créé une interface graphique en utilisant la bibliothèque graphique Sing. L'interface permet de :

- importer une table (\*.csv) correspondant à un model (\*.xmi)
- calculer les colonnes dérivées selon le modèle
- calculer de nouvelles colonnes
- sauvegarder la table résultante (\*.csv)

L'interface (figure 2.7) se compose de 3 trois bouton

- "Importer" qui d'importe et calcule les colonnes dérivées d'une table
- "Calculer" qui ajoute une nouvelle colonne dérivée à une table
- "Télécharger" qui sauvegarde la table modifier (ou non) dans un fichier.csv

De plus elle est composer de deux menus défilants qui permettent de sélectionner trois tables dans l'ordre de choir :

- le premier opérande (ici une colonne) pour calculer une colonne dérivée
- l'opérateur de la classe **Operations**
- le deuxième opérande du calcul

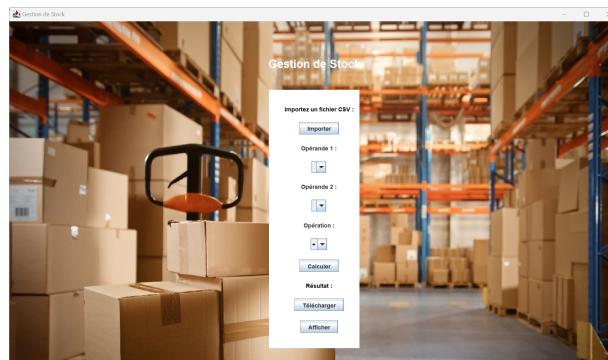


FIGURE 2.2 – Interface utilisateur

### 2.7.1 Import d'une table

Pour importer une table on clique sur le bouton "Importer" ce qui ouvre une fenêtre classique de gestion de fichier de là on choisit un fichier de type csv. Le fichier étant choisi on le convertis en un objet java que l'on pourra manipuler. A partir de cette objet java on calcule les colonnes dérivées que l'on place aussi dans cette objet.

### 2.7.2 Caluler une nouvelle colonne

Pour calculer une nouvelle colonne on sélectionne les deux opérandes et l'opérateur. Puis on clique sur le bouton calculer, ceci entraîne le calcul de la colonne à partir de celle de l'objet et la stockera dans ce même objet.

### 2.7.3 Sauvegarder les résultats

Quand l'utilisateur à finit de travailler il peut sauvegarder ses résultats en cliquant sur le bouton télécharger. Une autre fenêtre s'ouvre (la même que lors de l'import) et l'utilisateur doit choisir un fichier dans lequel sauvegarder résultats. Le fichier ainsi sélectionner on y écrit le contenu de l'objet au format csv.

### 2.7.4 Scénario d'utilisation

Le scénario d'utilisation usuel se décrit ainsi l'utilisateur importe sa table, effectue une suite de transformations dessus à l'aide du bouton calculer et enfin sauvegarde la table résultante.

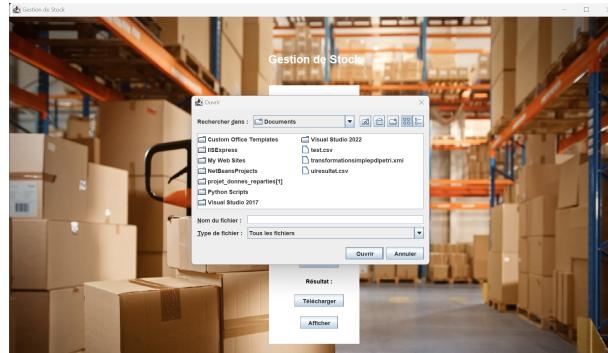


FIGURE 2.3 – clic sur le bouton importer

TableProduits
ID,Produit,Prix_Unitaire,Quantite_stock
001,ProduitA,10,100
002,ProduitB,20,150
003,ProduitC,15,200
Tablecommandes
ID,nom_Produit-commande,Prixproduit,Quantite_du_produit_commande
001,ProduitA,10,2
002,ProduitB,20,3
003,ProduitC,15,11

FIGURE 2.4 – Fichier des données à importer pour effectuer les calculs.

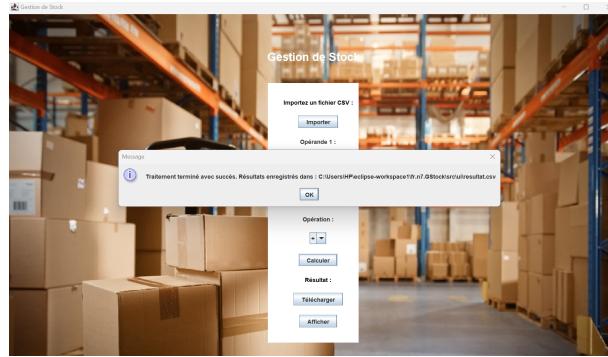


FIGURE 2.5 – lic sur le bouton calculer

TableProduits
ID,Produit,Prix_Unitaire,Quantite_stock
001,ProduitA,10,100
002,ProduitB,20,150
003,ProduitC,15,200

Tablecommandes
ID,nom_Commande,Prixproduit,Quantite_du_produit_Commande
001,ProduitA,10,2
002,ProduitB,20,3
003,ProduitC,15,11

FIGURE 2.6 – Fichier des résultats après application des calculs

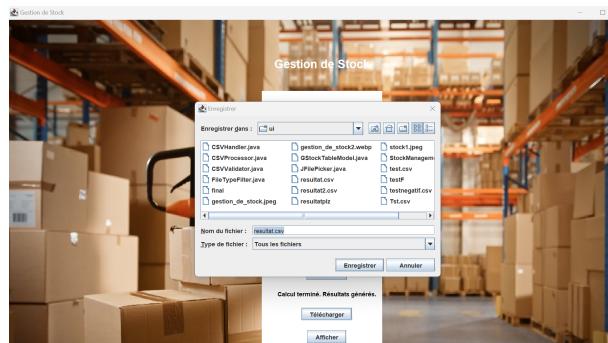


FIGURE 2.7 – Clic sur le bouton Télécharger

## 2.8 Modèle de syntaxe graphique avec Sirius

Après avoir créé notre métamodèle et l’instancié avec différents modèles, en particulier le modèle typique pour notre application de gestion de stock, nous sommes passés à la configuration de la partie graphique du projet, qui a pour but de visualiser graphiquement les modèles et de permettre d’en créer à travers une palette.

Nous avons réussi à créer un projet Sirius (viewpoint specification project) qui contient le fichier le plus important, `gstock.odesign`, dans lequel nous avons créé les différents éléments du métamodèle et les relations entre eux. Ainsi, pour visualiser à chaque fois les modifications faites sur le `.odesign`, nous avons créé un modèle (`tables.gstock`) sous le projet `samples`, qui a pour extension (`xml`).

L'architecture de notre .odesign est basée sur des containers, des nodes et des relations basées sur des edges où :

- Les tables sont des containers pour les colonnes représentées par des nodes.
- Les colonnes dérivées sont particulièrement des containers qui contiennent une expression, laquelle est à son tour un container pour les nodes opérateurs et opérandes.
- Les relations basées sur des edges sont utilisées pour relier les colonnes référencées à une table et une colonne, et également pour relier les opérandes aux colonnes qu'elles réfèrent.

Pour mieux comprendre cette architecture, l'image ci-dessous représente l'arborescence des éléments :

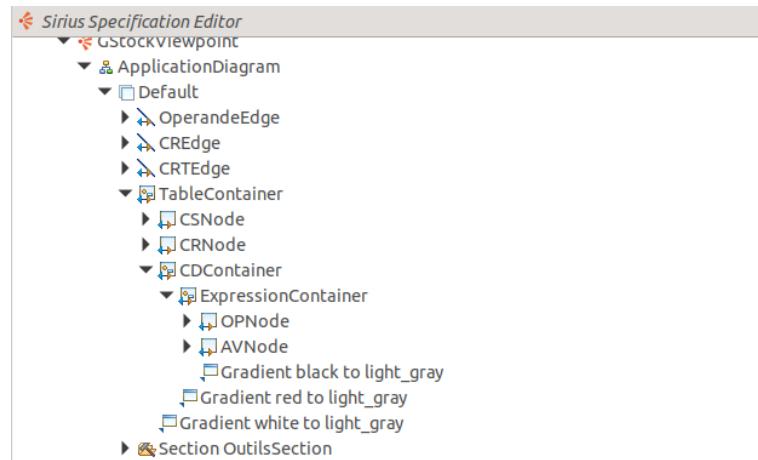


FIGURE 2.8 – Arborescence des éléments.

Et pour visualiser la vue graphique résultante après avoir appliqué cette représentation sur la racine Application du modèle tables.gstock, on obtient le diagramme suivant :

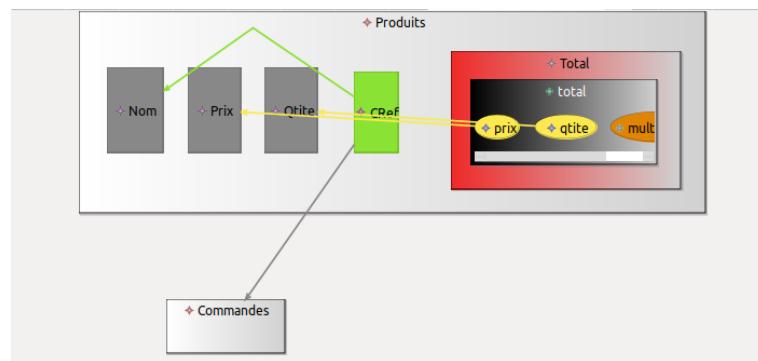


FIGURE 2.9 – Vue graphique du modèle.

Après la création des éléments nécessaires pour représenter un modèle, nous sommes passés à la définition des outils de chaque élément afin de garantir à l'utilisateur la possibilité de créer un schéma de table avec des drag-and-drop depuis la palette associée, sans comprendre la fonctionnalité EMF ou les métamodèles.

Nous avons ajouté alors la section `tools`, où nous avons spécifié l'outil de chaque élément et l'outil des relations. Nous avons réussi à créer correctement les outils pour les éléments, c'est-à-dire que les éléments sont bien créés avec tous les champs nécessaires à remplir, ainsi que positionnés uniquement dans les emplacements dédiés (par exemple : une expression ne peut pas être positionnée dans une colonne simple).

Pour les relations basées sur des edges, qui sont l'outillage associé aux références, nous avons réussi à spécifier leurs sources et leurs cibles, mais malheureusement, leur représentation graphique ne s'affiche pas. Nous avons montré le problème à notre encadrant, mais celui-ci n'a pas été résolu et reste moins clair que prévu.

L'image ci-dessous montre l'architecture de notre section Outils :

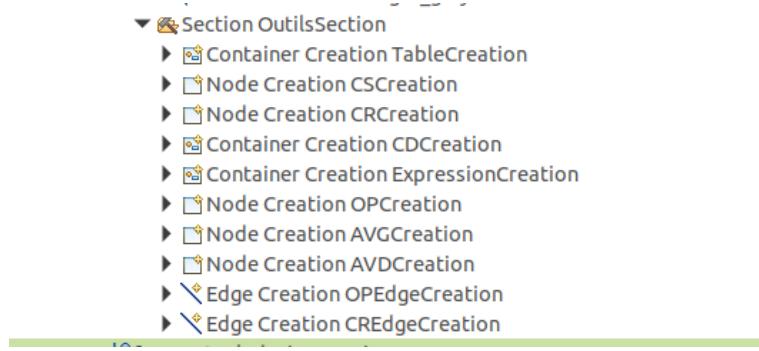


FIGURE 2.10 – Section des Outils.

Et cette image montre notre palette avec les modèles graphiques créés depuis celle-ci :

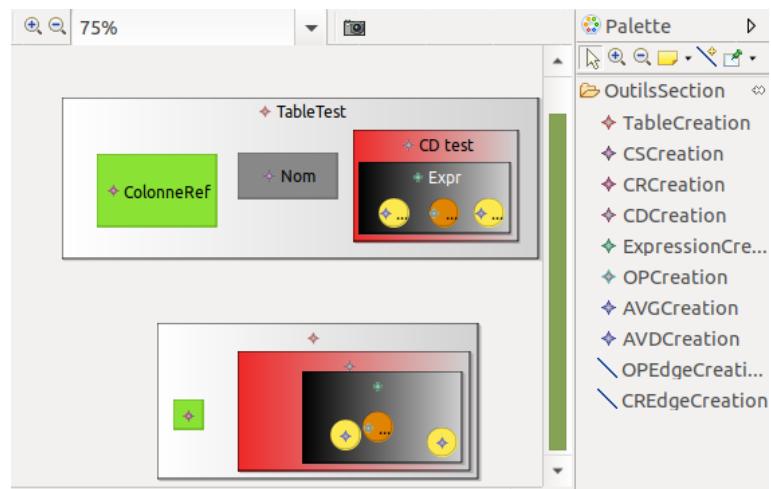


FIGURE 2.11 – Palette et vue graphique.

## 2.9 Contrainte avec Java.

Nous avons défini des contraintes pour traiter les différents lacunes dans les modèles initiés du métamodèle telles que :

- Nom obligatoire et formaté : Le nom (Nom) doit être non vide et il doit respecter un format spécifique (e.g, prix\_, Total, Produits Commandés etc.)
- Unicité des identifiants : Les identifiants (Id) des Tables et des Colonnes doivent être uniques dans leur contexte. (Déjà assuré depuis le métamodèle en spécifiant un Id sous forme d'attribut unique.)
- Coexistence des colonnes dans une table : Une Table ne peut pas contenir deux colonnes ayant le même nom (Nom).
- Interdiction des boucles : Dans les relations entre Colonne\_referencee, les dépendances ne doivent pas former une boucle, par exemple : Colonne A référence B et B référence A (e.g. éviter les références cycliques).
- Conformité des types : Le champ Type des colonnes doit être valide (Int ou String) et doit être non vide.
- Validité des expressions dans les colonnes dérivées : Dans une Colonne\_derivee, l'expression associée doit être valide : les opérateurs (Operateur) doivent être définis (addition, multiplication, etc.). Les opérandes doivent référer à des colonnes avec un type entier.

Voici les messages montrant chacun le résultat d'application des contraintes sur différents modèles. Chaque modèle ne respecte un aspect des contraintes :

1. Nom invalide :

Résultat de validation pour model/ApplicationT.xmi :  
Erreur dans 0 [gstock.impl.TableImpl@163e4e87 (id : 0, Nom : @table)] :  
Le nom de la table est invalide ou vide. Il doit respecter le format [A – Za – z][A – Za – z0 – 9]\*  
Validation terminée.

FIGURE 2.12

2. Même Identifiant :

The Id "1" of ColonneSimple Quantite collides with that one of Table T1.

FIGURE 2.13

3. Même nom dans une même Table :

Résultat de validation pour model/ApplicationMemeNom.xmi :  
Erreur dans gstock.impl.CollonnesimpleImpl@5dd6264 (id : 12, Nom : Quantite, Type : null) :  
La colonne 'Quantite' n'est pas unique dans la table 'T1'.  
Validation terminée.

FIGURE 2.14

4. Boucle dans colonne referencee :

Résultat de validation pour model/Application1.xmi :

**Erreur 1 :**

`gstock.impl.ColonneReferenceImpl@419c5f1a(id : 102, Nom : ColonneB, Type : null) :`  
Une boucle a été détectée dans les références de colonnes, commençant à la colonne  
`'gstock.impl.ColonneReferenceImpl@419c5f1a(id : 102, Nom : ColonneB, Type : null)'.`

**Erreur 2 :**

`gstock.impl.ColonneReferenceImpl@769e7ee8(id : 103, Nom : ColonneC, Type : null) :`  
Une boucle a été détectée dans les références de colonnes, commençant à la colonne  
`'gstock.impl.ColonneReferenceImpl@769e7ee8(id : 103, Nom : ColonneC, Type : null)'.`

**Erreur 3 :**

`gstock.impl.ColonneReferenceImpl@12b0404f(id : 101, Nom : ColonneA, Type : null) :`  
Une boucle a été détectée dans les références de colonnes, commençant à la colonne  
`'gstock.impl.ColonneReferenceImpl@12b0404f(id : 101, Nom : ColonneA, Type : null)'.`

Validation terminée.

FIGURE 2.15

5. le champ Type vide :

Erreur dans `gstock.impl.ColonnesimpleImpl@1ffe63b9` (id : 11, Nom : Quantite, Type : null) :  
La colonne 'Quantite' doit avoir un type non nul.

FIGURE 2.16

6. colonne derivee avec une expression invalide :

Résultat de validation pour model/Applicationcmd.xmi : Erreur dans `gstock.impl.ColonneDeriveeImpl@2b6faea6`(id : 14, Nom : Total, Type : Int) : La colonne 'Total' contient des données incohérentes avec son type : IntValidation terminée.

FIGURE 2.17

## 2.10 Modèle XText

La syntaxe abstraite d'un DSML exprimée en Ecore ne peut pas être manipulée directement. Le projet EMF d'Eclipse permet d'engendrer un éditeur arborescent et les classes Java pour manipuler un modèle conforme à un métamodèle. Cependant, ces outils ne sont pas très pratiques lorsque l'on souhaite saisir un modèle de manière conviviale.

Il est donc souhaitable d'associer à une syntaxe abstraite une syntaxes concrètes pour faciliter la construction et la modification des modèles. Pour la définition des syntaxes concrètes textuelles, nous utiliserons l'outil Xtext qui permet non seulement de définir une syntaxe textuelle pour un DSL, mais aussi de disposer d'un environnement de développement pour ce langage intégré dans Eclipse.

Voici la structure du projet Xtext :

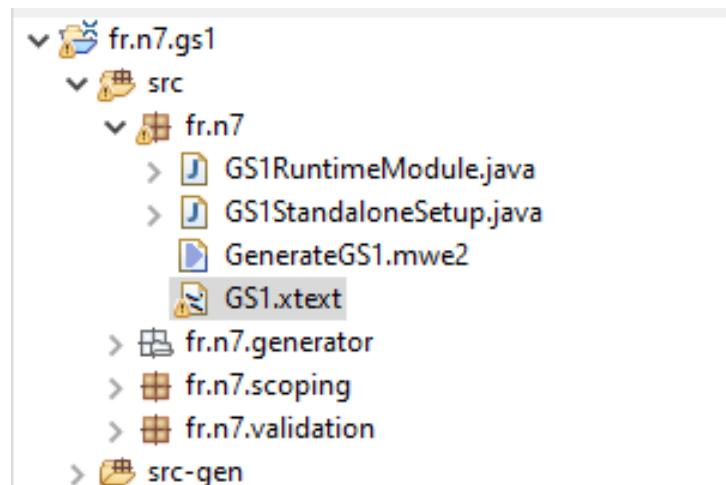


FIGURE 2.18 – Projet Xtext

Quand on définit notre syntaxe dans le fichier GS1.xtext, on exécute GenerateGS1.mwe2 et on obtient un message indiquant la création du modèle.

```
923 [main] INFO clipse.emf.mwe.utils.GenModelHelper - Registered GenModel 'http://www.eclipse.org/...
1514 [main] INFO text.xtext.generator.XtextGenerator - Generating fr.n7.GS1
1922 [main] INFO erator.ecore.EMFGeneratorFragment2 - Generating EMF model code
1962 [main] INFO clipse.emf.mwe.utils.GenModelHelper - Registered GenModel 'http://www.n7.fr...
5614 [main] INFO text.xtext.generator.XtextGenerator - Generating common infrastructure
5652 [main] INFO .emf.mwe2.runtime.workflow.Workflow - Done.]
```

FIGURE 2.19 – Message de création du modèle

Le métamodèle se génère dans le dossier model. Voici le modèle obtenu :

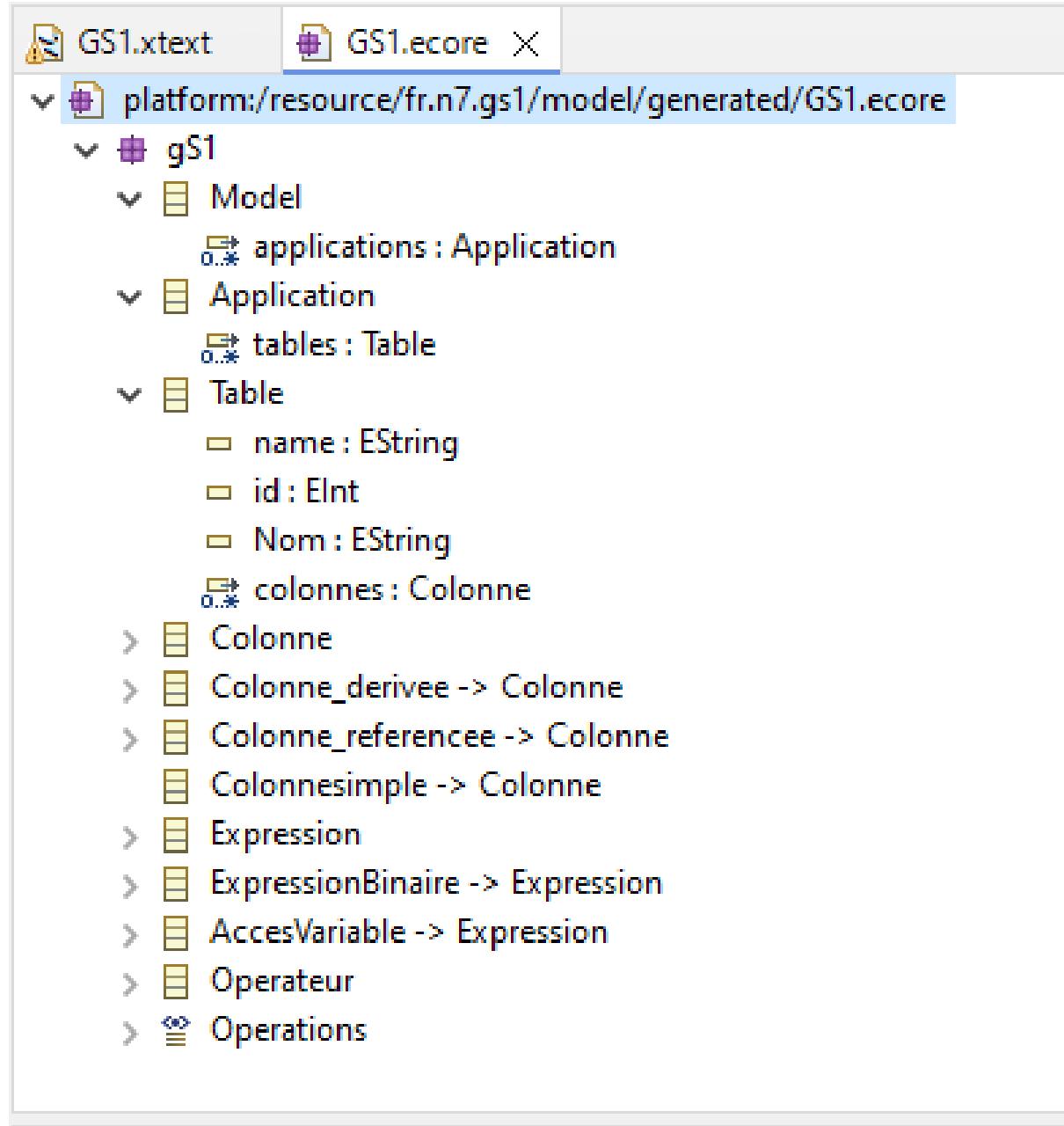


FIGURE 2.20 – Métamodèle généré par Xtext

Ensuite, pour tester la validité de notre métamodèle généré, on passe à l'Eclipse de déploiement en sélectionnant Run As Eclipse Application. Puis, on écrit un exemple de modèle pour le métamodèle que nous avons défini. À partir du modèle créé, nous obtenons un résultat qui s'aligne avec le métamodèle proposé.

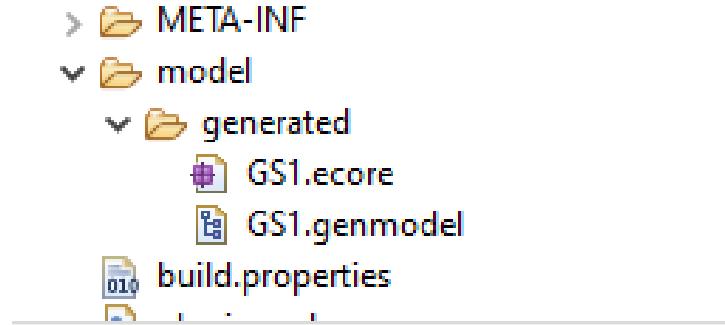


FIGURE 2.21 – Modèle de validation

Et puis on constate que notre modèle est bien valide

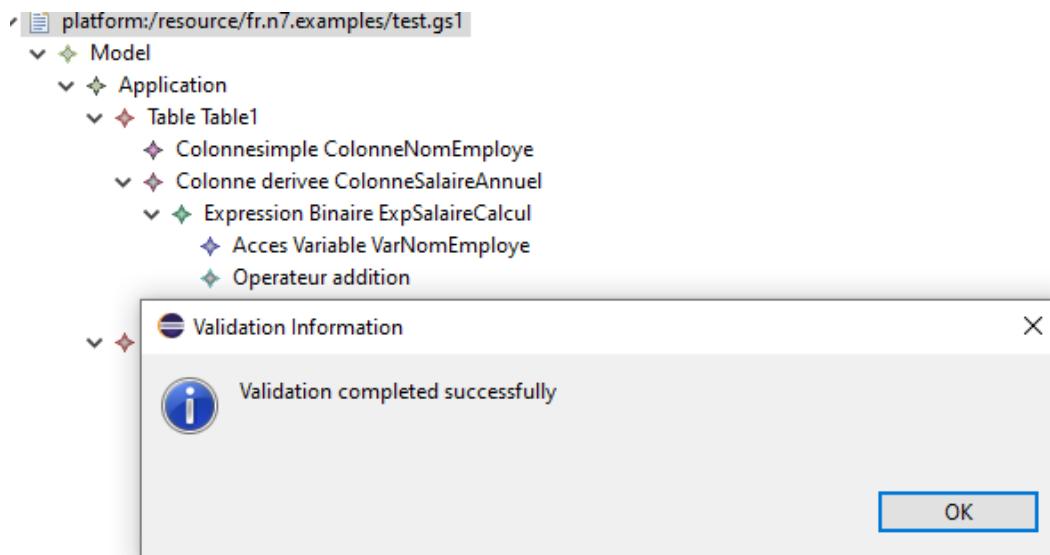


FIGURE 2.22 – Message de Validation

## 2.11 Conclusion

Durant ce projet on a créer un métamodel de table. On a aussi défini une syntaxe graphique qui permet de créer des models. De plus nous avons définit des contraintes sur ces tables pour l'éventualité où on les exploiterai afin de généré des scripts. Enfin nous avons réalisé une interface graphique qui permet d'éditer une table.

On pourrait ajouter une génération de script permettant d'éditer une table à partir d'une expression défini par l'utilisateur. Implémenter une interface graphique permettant directement d'éditer les différents entrées des colonnes. Permettre la prise en charge d'autre format pour implémenter une table.

De plus notre implémentation est limité dans la mesure où on ne considère pas les expressions unaire on ne peut donc pas utiliser les fonctions usuelles (exponentielle, sinus, cosinus ...). Même si on pourrait se ramenner à une expression unaire à partir d'une expression binaire.