



INGÉNIERIE DIRIGÉE PAR LES MODÈLES

Environnement de Calcul Domaine-Spécifique

par

Thomas BOCANDÉ

Tom AUDARD

Louiza CHEKRAOUI

Youssef EL ALAMI

Groupe PR-L12-1

Département Science
du Numérique

TOULOUSE-INP ENSEEIHT

Date : 4 décembre 2024

Table des matières

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Méta-modèle SchemaTable | 2 |
| 2.1 | La création du méta-modèle | 2 |
| 2.2 | Définition de contraintes avec OCL | 3 |
| 2.3 | Transformation de modèle à texte | 4 |
| 3 | Méta-modèle Algorithme | 5 |
| 3.1 | La création du méta-modèle | 5 |
| 3.2 | Le catalogue | 6 |
| 3.3 | Définition des contraintes OCL | 7 |
| 3.4 | Définition de syntaxes concrètes textuelles | 7 |
| 4 | Définition de syntaxes concrètes graphiques avec Sirius | 9 |
| 4.1 | Le méta-modèle Calcul | 9 |
| 4.2 | Syntaxe graphique SIRIUS | 10 |
| 5 | Mise en place de bibliothèques | 11 |
| 5.1 | Importation de CSV | 11 |
| 5.2 | Exportation en CSV | 11 |
| 6 | Conclusion | 12 |
| 6.1 | Conclusion personnelle de Tom | 12 |
| 6.2 | Conclusion personnelle de Youssef | 12 |
| 6.3 | Conclusion personnelle de Louiza | 12 |
| 6.4 | Conclusion personnelle de Thomas | 13 |
| 6.5 | Conclusion générale | 13 |

Table des figures

| | | |
|-----|---|----|
| 2.1 | Méta-modèle SchemaTable | 2 |
| 2.2 | Template de génération tableau HTML | 4 |
| 2.3 | Représentation des données sous forme de tableau HTML | 4 |
| 3.1 | Ancien Méta-modèle Algorithme | 5 |
| 3.2 | Méta-modèle Algorithme | 6 |
| 3.3 | Méta-modèle Catalogue | 6 |
| 3.4 | Algorithme 1 | 7 |
| 3.5 | Ecore de l'algorithme 1 | 7 |
| 3.6 | Algorithme 1 | 8 |
| 3.7 | Ecore de l'algorithme 2 | 8 |
| 4.1 | Méta-modèle Calcul | 9 |
| 4.2 | Calcul de moyenne - Sirius | 10 |

1. INTRODUCTION

Le but de ce projet était de proposer une suite basée sur la plate-forme Eclipse et les technologies EMF, permettant à un utilisateur de définir des schémas de donnée ainsi que des calculs automatisés sur ces schémas. Durant ce projet, nous avons commencé par réfléchir à la conception de différents méta-modèles, puis nous avons essayé de définir des transformations modèle à texte et des syntaxes concrètes textuelles et graphiques. Dans ce présent rapport, nous allons mettre en avant les points clés de notre réalisation et les choix de conception que nous avons essayé de mettre en place.

2. MÉTA-MODÈLE SCHEMATABLE

2.1 La création du méta-modèle

Nous avons commencé le projet en réfléchissant à comment modéliser un schéma de table comme voulu dans la feature 1. Pour cela, nous sommes partis de l'exemple donné dans le sujet (Cours conglomérat Allemand). Nous sommes arrivés au méta-modèle de la figure 2.1 qui permet de modéliser toutes les sous-features de la feature 1.

Dans ce méta-modèle, nous retrouvons ainsi en classe racine la classe *SchemaTable* qui est donc composée d'une unique colonne d'identifiants (*ColonneID*) et de colonnes (*Colonne*).

Pour les colonnes et la colonne d'identifiants, nous avons décidé de leur mettre en attribut leur identifiant, leur nom et leur nombre de lignes. La colonne d'identifiants est composée d'identifiants de ligne (*IDLigne*) qui ont une valeur et qui font référence aux données d'un schéma de table sur une même ligne. Les colonnes sont quant à elles composées de données (*Donnee*) qui ont une valeur en flottant, comme montré dans l'exemple du sujet.

Pour que l'utilisateur puisse définir des contraintes sur les colonnes, nous avons ajouté une classe *ContrainteColonne* qui sera contenue dans une *Colonne*. La classe *ContrainteColonne* possède en attribut un prédicat qui devra être vérifié sur la colonne qui contient cette contrainte. Une colonne peut se voir appliquer zero ou plusieurs contraintes.

Enfin, nous avons représenté le fait que l'utilisateur peut déclarer qu'une colonne dérive d'un algorithme via la référence croisée "deriveDe". L'utilisateur peut aussi déclarer qu'une colonne provient d'un autre schéma de table grâce à la classe *Reference*. Cette classe possède en attribut l'identifiant de la colonne référencée dans le schéma de table étranger ainsi que le nombre de lignes de la colonne référencée. Elle fait référence à un autre schéma de table.

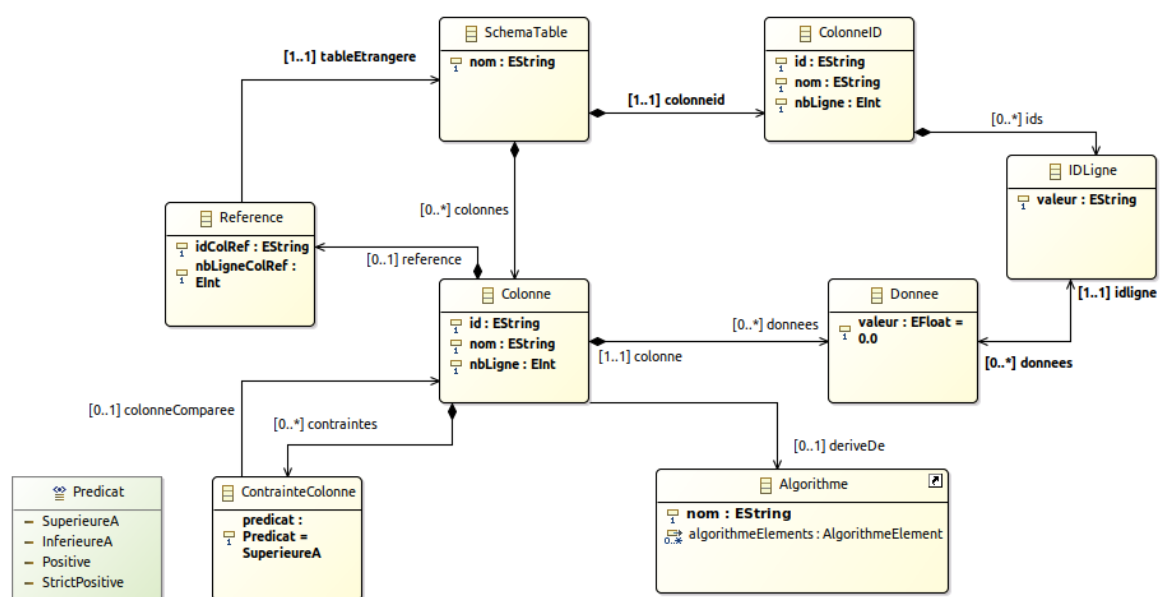


FIGURE 2.1 – Méta-modèle SchemaTable

2.2 Définition de contraintes avec OCL

Comme le méta-modèle ne permet pas de s'assurer qu'un schéma de table est correct, nous avons défini quelques contraintes grâce à OCL :

- La colonne d'identifiants a un identifiant unique ;
- Les colonnes ont le même nombre de ligne que la colonne d'identifiants ;
- Il y a autant d'identifiants de ligne que le nombre de lignes déclaré dans la colonne d'identifiants ;
- Les colonnes ont un identifiant unique ;
- Il y a autant de données que le nombre de lignes déclaré dans la colonne ;
- S'il y a une référence, elle est respectée (même nombre de ligne que la colonne de référence).
- Une contrainte sur une colonne est valide (i.e. si le prédicat est *SuperieureA* ou *InferieureA*, une colonne est bien référencée pour la comparaison.

Nous avons pu les tester sur 2 exemples : exempleProjet.xmi qui est le premier exemple du projet et exempleProjetVoulu.xmi qui est la table que l'ingénieur génère à partir de l'exemple du projet avec les références et les colonnes dérivées.

2.3 Transformation de modèle à texte

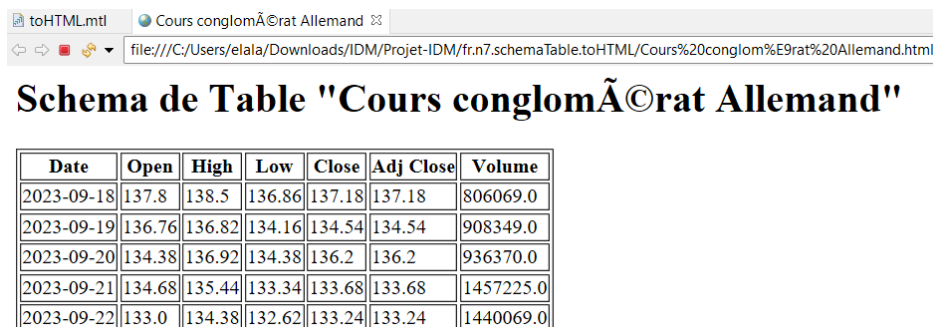
Après avoir terminé la création du modèle schéma de table, générer des exemples au format .xmi et défini les contraintes OCL, nous avons créé un projet Acceleo "fr.n7.schemaTable.toHTML" dans le but d'engendrer un fichier HTML à partir d'un modèle SchemaTable. L'objectif est de représenter les données sous la forme d'un tableau en HTML. Pour ce faire, nous avons créé la *template* de génération présentée dans la figure 2.2. Celui-ci spécifie le nom du SchemaTable, parcourant la colonne d'identifiants et les colonnes de données pour extraire les noms des colonnes. Ensuite, nous parcourons les lignes de la colonne identifiant pour récupérer l'identifiant de chaque ligne et les données associées. En exécutant le fichier "toHTML.mtl" avec "exempleProjet.xmi" comme modèle d'entrée qui contient les données de l'exemple donné dans le sujet du projet et nous trouvons le tableau suivant (figure 2.3).

```
[template public schemaTableToHTML(aSchemaTable : SchemaTable)]
[comment @main/]
[file (aSchemaTable.nom + '.html', false, 'UTF-8')]
<head><title>[aSchemaTable.nom/]</title></head>
<body>
  <h1>Schema de Table "[aSchemaTable.nom/]"</h1>
  <table border="1">
    <thead>
      [let colonneid : OrderedSet(ColonneID) = aSchemaTable.getColonnesID()]
      [for (cid : ColonneID | colonneid)]
      <th> [cid.nom/] </th>
      [/for]
    [/let]
    [let colonne : OrderedSet(Colonne) = aSchemaTable.getColonnes()]
    [for (c : Colonne | colonne)]
    <th> [c.nom/] </th>
    [/for]
  [/let]
</thead>
<tbody>
  [let ids : OrderedSet(IDLigne) = aSchemaTable.getIDLigne()]
  [for (id : IDLigne | ids)]
  <tr>

      <td> [id.valeur/] </td>
      [for (d : Donnee | id.donnees)]
      <td> [d.valeur/] </td>
      [/for]

    </tr>
  [/for]
</tbody>
</table>
</body>
[/file]
[/template]
```

FIGURE 2.2 – Template de génération tableau HTML



| Date | Open | High | Low | Close | Adj Close | Volume |
|------------|--------|--------|--------|--------|-----------|-----------|
| 2023-09-18 | 137.8 | 138.5 | 136.86 | 137.18 | 137.18 | 806069.0 |
| 2023-09-19 | 136.76 | 136.82 | 134.16 | 134.54 | 134.54 | 908349.0 |
| 2023-09-20 | 134.38 | 136.92 | 134.38 | 136.2 | 136.2 | 936370.0 |
| 2023-09-21 | 134.68 | 135.44 | 133.34 | 133.68 | 133.68 | 1457225.0 |
| 2023-09-22 | 133.0 | 134.38 | 132.62 | 133.24 | 133.24 | 1440069.0 |

FIGURE 2.3 – Représentation des données sous forme de tableau HTML

3. MÉTA-MODÈLE ALGORITHME

3.1 La création du méta-modèle

Afin de faciliter la création, la sauvegarde et la consultation des algorithmes dans une interface ergonomique et adaptée, nous avons élaboré le méta-modèle illustré dans la figure 3.1 qui permet de modéliser toutes les sous-caractéristiques de la fonctionnalité 2.

Un algorithme est définie par un nom et se compose des ressources qui le mettent en œuvre, ainsi que des détails de ses entrées, de ses sorties et sa documentation. En modélisant ce méta-modèle, nous avons introduit une classe *Algorithme* qui regroupe plusieurs éléments. En conséquences, nous avons créé une classe abstraite *AlgorithmeElement* qui est héritée par une classe *Ressource* représentant une ressource qui réalise l'algorithme. Cette ressource peut prendre la forme d'un fichier, d'un programme ou autre, avec le chemin vers cette ressource comme attribut.

Une ressource est composée de plusieurs ports, d'où la création de la classe *Port* avec le nom comme attribut. Ces ports peuvent être soit des ports d'entrée ou des ports de sortie, donnant lieu à une classe *PortEntree* et à une classe *PortSortie*. Cette dernière a une valeur récupérée par la classe *Sortie* qui a un nom et représente la sortie de l'algorithme. Nous avons également la classe *Entree*, spécifiant le nom, la valeur et si cette entrée est fixe ou non. Cela permet de proposer un algorithme avec un paramètre particulier. Les deux classes *Entree* et *Sortie* héritent de la classe *AlgorithmeElement*, car les entrées et les sorties sont des éléments de l'algorithme.

Une classe *Documentation* a été ajoutée pour permettre à l'utilisateur d'identifier ce que fait chaque entrée, décrire l'algorithme, etc.

Enfin, la classe *Catalogue*, regroupe la classe *Algorithme*, permettant d'organiser les algorithmes dans des catalogues partageables et réutilisables, accessibles aux utilisateurs.

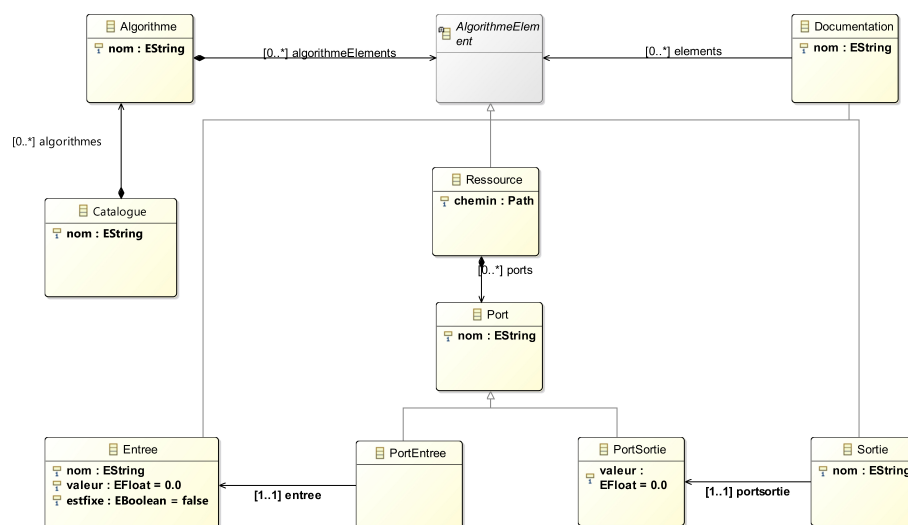


FIGURE 3.1 – Ancien Méta-modèle Algorithme

Par la suite, une fois le méta-modèle de Calcul créé, nous avons modifié le méta-modèle Algorithme pour prendre en compte des calculs et nous avons ainsi obtenu le méta-modèle de la figure 3.2. Cela permet à la ressource de référencer un calcul si elle est de type *CALCUL* et ainsi associer ses ports d'entrées à des arguments du calcul et son port de sortie à la sortie du calcul.

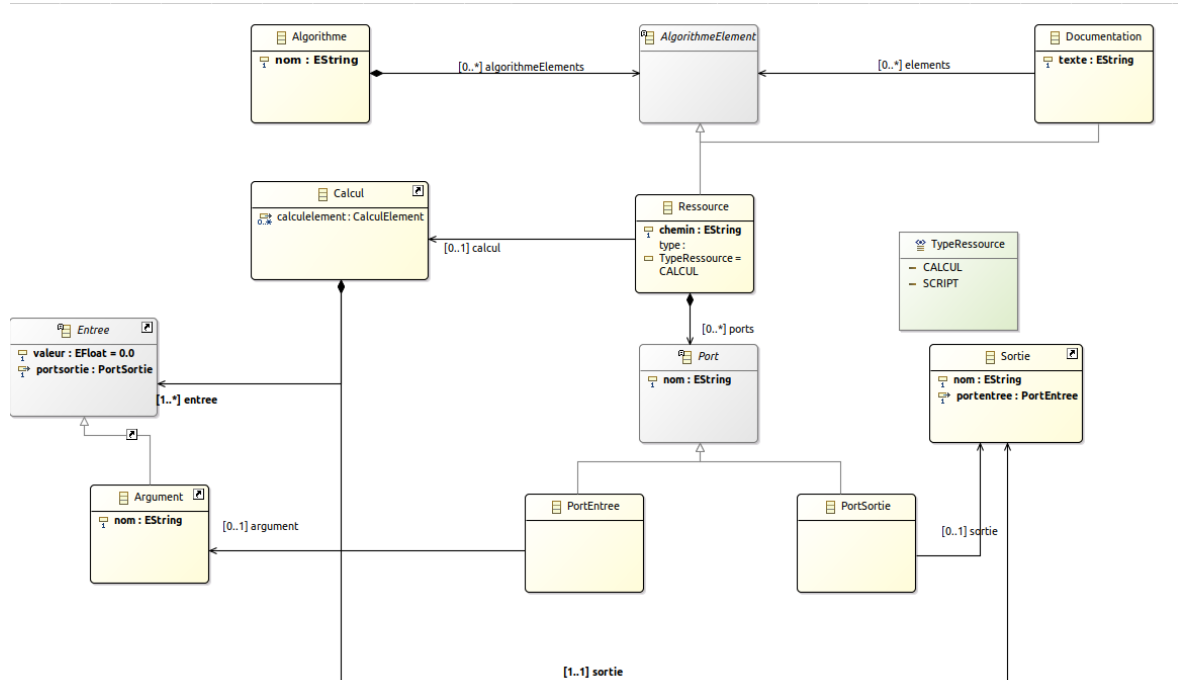


FIGURE 3.2 – Méta-modèle Algorithme

3.2 Le catalogue

Pour représenter le catalogue qui permet de regrouper les algorithmes, nous avons décidé de créer un nouveau méta-modèle au cas-où nous voudrions ajouter des fonctionnalités. (voir figure 3.3).

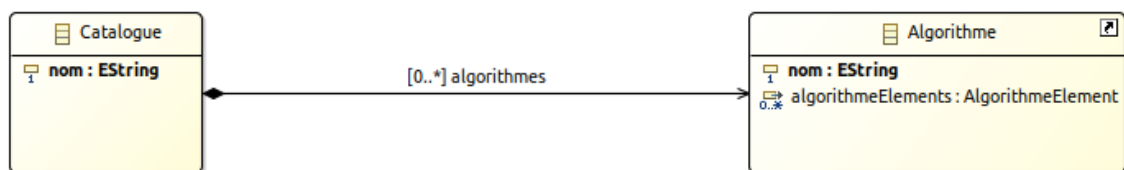


FIGURE 3.3 – Méta-modèle Catalogue

3.3 Définition des contraintes OCL

Pour les algorithmes et les catalogue, nous avons défini les contraintes OCL suivantes :

- Les algorithmes d'un même catalogue ont un nom unique ;
- Il y a au moins un port d'entrée et un port de sortie ;
- Si la ressource est de type *CALCUL*, elle référence un calcul.

Nous avons testé ces contraintes avec les exemples cités dans le sujet : algoMoyenne.xmi qui représente l'algorithme de la moyenne qui va donc référencer le calcul Moyenne (calculMoyenne.xmi) qui respecte le méta-modèle calcul. algoVariation.xmi qui représente l'algorithme de la variation en Python.

3.4 Définition de syntaxes concrètes textuelles

Afin de définir une syntaxe concrète textuelle pour le modèle *Algorithme* au travers d'une grammaire, nous avons utilisé l'outil *Xtext*. Le projet *fr.n7.algorithme.xtext* contient un fichier *Xtext* (*AlgorithmeXtext.xtext*) qui spécifie la syntaxe de l'algorithme avec une extension *.algo*. Dans ce fichier, nous avons écrit une grammaire pour décrire les deux algorithmes proposés dans le sujet. Les résultats obtenus, illustrés dans les figures 3.4 et 3.6, présentent l'Ecore correspondant, comme indiqué dans les figures 3.5 et 3.7 respectivement. Après avoir finaliser la syntaxe concrète textuelle *xtext*, et généré des modèles grâce à cette syntaxe, notre intention était d'élaborer une transformation modèle à modèle, utilisant une transformation ATL d'un modèle conforme au méta-modèle *AlgorithmeXtext* à un modèle conforme à *Algorithme*. Cependant, malheureusement, en raison de contraintes temporelles, nous n'avons pas eu le temps de mener cette tâche à son terme.

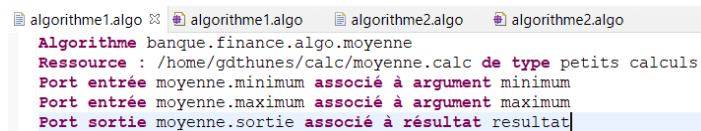


FIGURE 3.4 – Algorithme 1

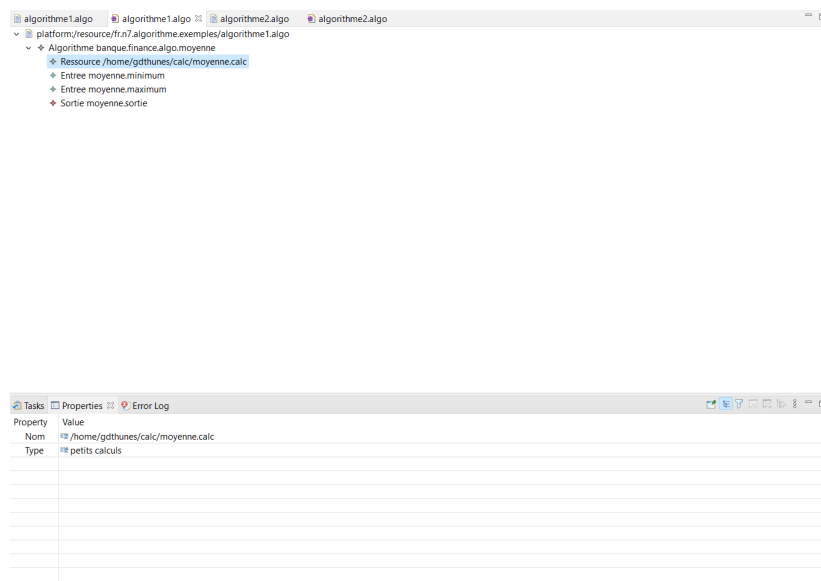


FIGURE 3.5 – Ecore de l'algorithme 1

FIGURE 3.6 – Algorithme 1

FIGURE 3.7 – Ecore de l’algorithme 2

4. DÉFINITION DE SYNTAXES CONCRÈTES GRAPHIQUES AVEC SIRIUS

4.1 Le méta-modèle Calcul

La troisième feauture proposée dans le sujet d'offrir à l'utilisateur la possibilité de spécifier des calculs à l'aide d'une syntaxe concrète graphique (Sirius dans notre cas). Pour répondre à ce besoin, nous avons dans un premier temps eu un travail de réflexion sur quel méta-modèle, allions-nous nous baser. Nous avons décidé de créer un méta-modèle à part entière *Calcul* qui sera par la suite référencée par une ressource utilisée dans un Algorithme. Cette conception nous paraissait plus cohérente et moins complexe que de tout assembler dans un seul méta-modèle.

Pour faciliter à l'utilisateur la réalisation des features de F3.1 à F3.6, nous avons construit le méta-modèle ci-dessous :

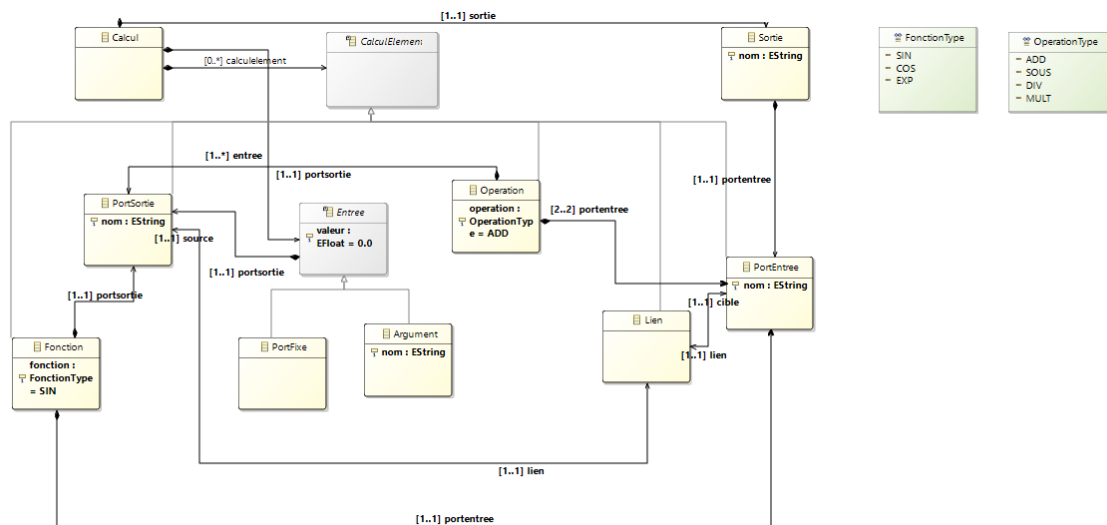


FIGURE 4.1 – Méta-modèle Calcul

Nous avons introduit une classe principale "Calcul" qui est composée d'une seule sortie et une ou plusieurs entrées et qui est constituée de plusieurs éléments comme : les ports, les opérations, fonction. . . Pour ce, nous avons introduit une classe abstraite *CalculElement* qui est héritée par tous les éléments cités ci-dessus. Pour mettre en lien les entrées, les opérands (blocs opérations), ports fixes (modéliser par une classe *PortFixe* qui hérite de la classe abstraite entrée avec un attribut valeur ainsi que la sortie, nous avons associé à ces éléments des ports d'entrées et de sorties et puis créer une classe *Lien* qui est référencée par un port d'entrée qui est considéré comme une cible et un port de sortie comme une source.

4.2 Syntaxe graphique SIRIUS

Dans l'éclipse de développement, nous avons commencé par créer un projet contenant un modèle de calcul : Celui proposé dans le sujet de projet pour calculer une moyenne à partir d'un min et un max importer d'un schéma de table. Après avoir préparé l'exemple .xmi, nous avons commencé à mettre en place le modèle de description de la syntaxe graphique. On a défini dans l'éclipse de déploiement le "*calcul.odesign*", il s'agit du modèle de description de l'interface de Sirius. Dans lequel nous avons représenté les opérations, les entrées et sorties par des Nodes sur lesquels se trouvent des "*BorderedNodes*" qui représentent les ports d'entrée et de sortie. Pour relier ces différents éléments, nous avons représenté la classe "Lien" par des Edge Relations avec comme cible les ports d'entrées et comme source les ports de sortie. Hélas, les liens ne s'affichent pas sur le schéma graphique même après modification de l'Ecore en ajoutant des e-Opposition aux relations avec la classe "Lien". La solution qui pourrait être apportée est d'ajouter des contraintes OCL sur les conditions des entrées/sorties des ports et les importer dans Sirius.

Voici le schéma du calcul de moyenne obtenu :

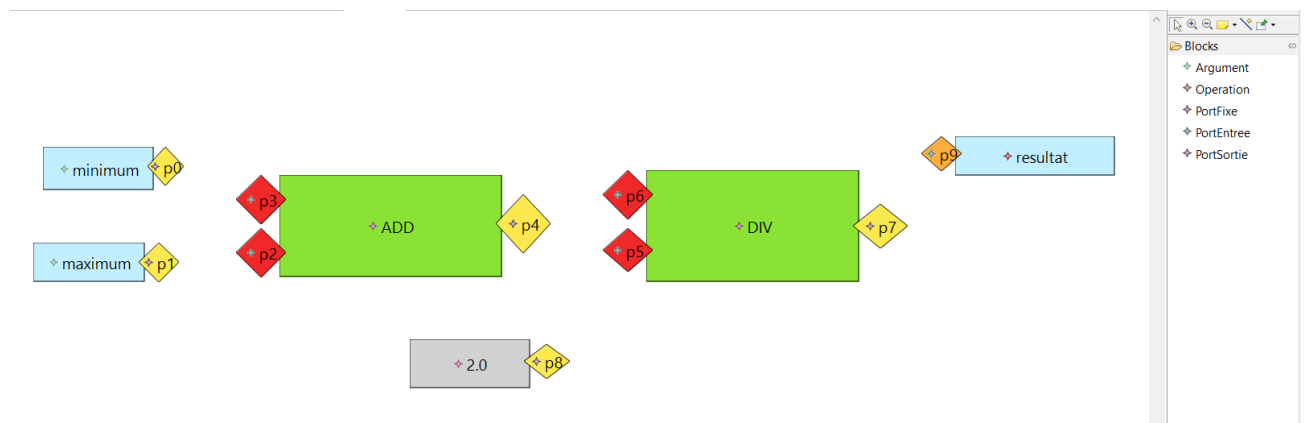


FIGURE 4.2 – Calcul de moyenne - Sirius

Nous avons aussi mis en place une palette qui permet à l'utilisateur d'ajouter des éléments à son calcul.

5. MISE EN PLACE DE LIBRAIRIES

5.1 Importation de CSV

La première fonctionnalité que nous avons décidé d'implémenter est l'import de fichier CSV et de les transformer en schéma de table compréhensible par l'utilisateur. Nous avons pensé que la manière la plus naturelle était de faire une transformation de texte à modèle via l'outil Xtext. Mais, il s'avère que définir une grammaire pour lire des données dans un .csv n'est pas si simple. En effet Xtext a du mal à parser des chaînes de caractère contenant des points, des tirets ou des slashes. Avec du recul il aurait été plus simple de créer un script java afin d'importer les données, si nous avions eu plus de temps.

5.2 Exportation en CSV

Ensuite nous avons choisi d'implémenter l'export de table en .csv. Cette fois nous avons fait la transformation grâce à une template .mtl et réaliser la transformation avec Acceleo.

6. CONCLUSION

6.1 Conclusion personnelle de Tom

J'ai eu beaucoup de mal à commencer le projet dû au sujet très vague et complexe. Le plus compliqué était en fait de savoir quoi faire et dans quel ordre. En commençant par faire les méta-modèles, j'ai réussi à mieux comprendre le but du projet. Je me suis plutôt occupé de la création des méta-modèles durant ce projet, même si j'ai aidé un peu sur les autres parties. Malgré tout, j'ai trouvé le sujet intéressant, mais je pense que plus de précision sur ce qui est vraiment attendu dans le sujet serait pour le mieux afin d'éviter de perdre beaucoup de temps, à savoir ce qu'il faut faire ou partir dans la mauvaise direction dès le départ. En effet, nous avons par exemple appris pendant l'oral que les données ne devaient pas être contenues dans le schéma de table et que celui-ci ne devait être "que la forme d'une table" de manière générale. Enfin, l'organisation du groupe a été assez compliquée selon moi dû aux habitudes de travail de chacun et au manque de communication.

6.2 Conclusion personnelle de Youssef

Au départ, j'ai eu mal à saisir l'objectif du sujet, je ne savais pas si nous devions nous concentrer uniquement sur les outils IDM, les méta-modèles, les transformations etc..., ou si, en plus de cela, nous devions développer une interface graphique qui permet d'importer des fichiers CSV et afficher des tables. . . Cependant, en discutant avec le groupe, nous avons décidé de nous concentrer entièrement sur les outils IDM. J'ai donc contribué en collaborant avec le groupe pour réfléchir à la conception des méta-modèles. Ensuite, j'ai effectué une transformation modèle à texte avec Acceleo, générant un script HTML pour créer un tableau à partir d'un modèle de schéma de table. J'ai développé aussi une grammaire *Xtext* pour définir la syntaxe d'un algorithme. J'ai tenté d'implémenter la transformation ATL pour le modèle algorithme généré par *Xtext*, ainsi qu'une transformation Acceleo pour générer un script Gnuplot à partir d'un schéma de table. Malheureusement, en raison d'erreurs dans Eclipse et de contrainte de temps, je n'ai pas finalisé ces deux tâches. En termes de l'organisation du groupe, la communication a été un peu complexe, surtout pendant les vacances de Noël. En conclusion, je pense que le projet était intéressant et que nous avons acquis beaucoup de connaissances.

6.3 Conclusion personnelle de Louiza

Dans un premier temps, la compréhension du sujet et comprendre ce qui nous a été demandé m'a pris un certain temps de réflexion. J'ai contribué à la réflexion et la mise en place des méta-modèles Algorithme et Calcul sur lesquels nous avons longuement réfléchi durant et hors les séances de TP dirigé. Après avoir mis en place la version finale du méta-modèle qui a pris place assez tard dans le temps, car nous avons modifié maintes fois le méta-modèle Algorithme et nous pensions réaliser la syntaxe graphique à partir de ce méta-modèle. Je me suis ensuite concentrée sur la syntaxe graphique, l'importation de l'instance du Ecore

"calculmoyenne.xmi" m'a pris énormément de temps, car je n'arriverai pas à créer un modèle, j'ai dû contourner le problème assez vite et commencé à travailler sur le calque du design. En ce qui concerne l'organisation de l'équipe et la communication au sein de cette dernière, ce n'était pas très fluide malgré la mise en place de plusieurs outils de communication comme un tableau Trello, un groupe Discord. . . et je pense que ça revient à la méthode de travail de chacun d'entre nous et en grande partie à la pression que nous avons vécu les dernières semaines avant et après les vacances. Malgré cela, nous avons réussi à comprendre certains points du sujet, à les aborder et les concrétiser.

6.4 Conclusion personnelle de Thomas

Au début la compréhension du sujet était un peu difficile, c'est pour cela que nous avons décidé par faire différents méta-modèles afin de mieux comprendre le sujet. Malgré cela nous n'avons eu du mal à voir la séparation entre les deux utilisateurs, celui qui créé les outils et celui qui les utilise. La période durant laquelle le projet c'est déroulé était aussi très chargé avec les autres projets et partiels, la communication interne au groupe en a été donc très affecté où chacun développait ses propres features dans son coin. On a eu aussi du mal à synchroniser nos versions de Java ce qui a demandé pas mal de temps consacrer au debug des différents projets par contre on a eu aucun soucis sur la version d'Eclipse utilisée. Nous avons cependant réussi à comprendre quelques notions et à développer la plupart des features.

6.5 Conclusion générale

Pour conclure, ce projet a été très enrichissant en termes de notions à comprendre, analyser et à maîtriser. En effet, il constitue ainsi une nouvelle expérience enrichissante dans le domaine de la modélisation. Cette fois (par rapport au mini-projet), il y avait en plus de l'enjeu d'utiliser la panoplie d'outils et de concepts offerts par Eclipse, la compréhension du sujet qui était assez difficile. La prise de décision en groupe sur la conception des méta-modèles sur lesquels notre travail a été basé n'a pas été évidente. La complexité du sujet nous a demandé un effort intellectuel significatif et nous avons passé beaucoup de temps sur la conception des premiers méta-modèles.

Sur le plan pratique, ce projet nous a permis de mûrir nos compétences et de consolider les notions vues lors du mini-projet afin de les utiliser dans un contexte plus complexe. Du point de vue travail en groupe, la prise de décision sur chaque étape du projet était une prise de décision collective, ce qui a demandé une bonne communication. Nous avons tous fait de notre mieux pour contribuer de manière significative à l'avancement du projet, mais le contexte dans lequel il a été réalisé n'était très propice pour arriver à réaliser toutes les features demandées parfaitement (période d'examens, autres projets en parallèles ...). Malgré ces péripéties, nous avons réussi à tous contribuer au projet grâce à notre détermination, chacun avec son point de vue et ses idées. Nous sommes convaincus que les compétences acquises au cours de ce travail collaboratif, seront précieuses dans nos futures collaborations professionnelles.