UNIVERSITÉ CÔTE D'AZUR
SOPHIATECH

Projet : Langage et Compilation

Etudiant: *GANGBADJA Paul Moïse*

Formation: *L3 Intelligence artificielle* – Enseignant: *Julien Deantoni*
Date: *1er mai 2023*

Le but de ce projet est de mettre en place une DSL afin qu'à partir d'un fichier **csv**, générer une page **html** dont l'exécution par le navigateur présente un ou plusieurs graphes. La librairie utilisé pour les graphes est *chartjs*.

1. Lien vers le repo github

Lien vers le repo github du project est https://github.com/Enigmatik100/chartit-dsl

2. Description du langage

2.1. Meta model.

Dans notre méta-modèle nous avons les concepts suivants :

1. Program : qui représente notre program, le point d'entrée.
2. DataFile : qui représente le fichier à utiliser
3. Chart : qui représente le graphe
4. Column : qui représente une colonne du fichier.
5. Constant : qui est une classe abstraite dont hérite les types string, int et double.
6. Condition : qui contient une référence vers une colonne, une constante et un opérateur. Un graphe peut contenir 0 ou plusieurs conditions pour filtrer les données.

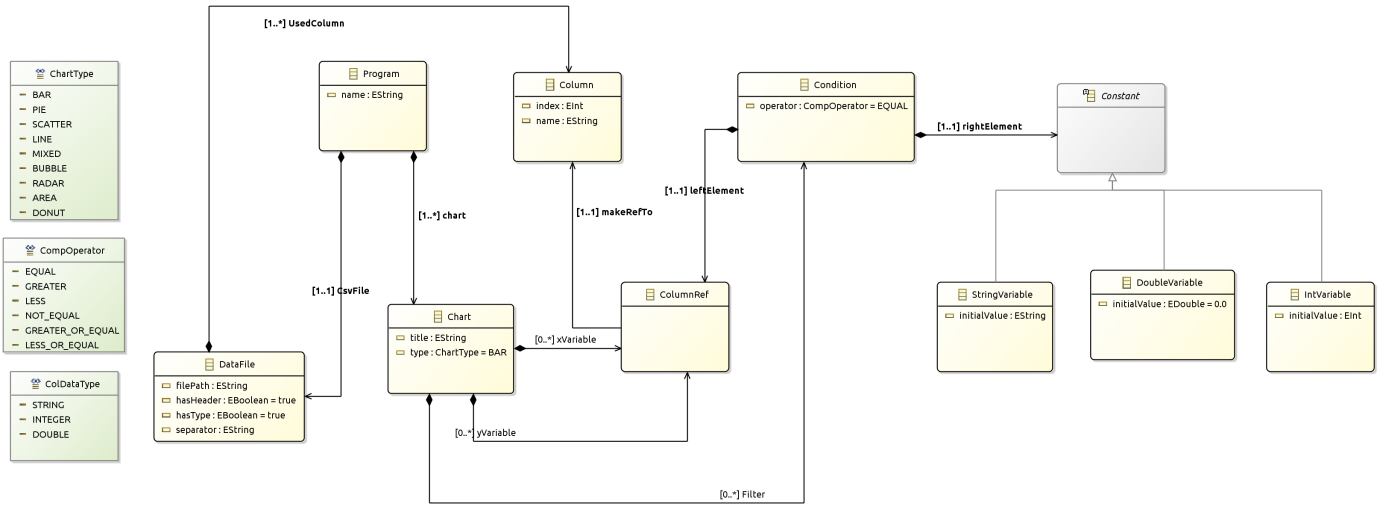

FIGURE 1 – méta model du langage

2.2. Syntaxe concrète.

L'environnement nous génère par défaut une syntaxe concrète. Nous avons donc essayé de mettre à jour afin d'avoir une syntaxe plus simple et plus intuitive.

La version modifiée de la syntaxe concrète se présente comme suit.

```
// automatically generated by Xtext
grammar fr.unice.cotedazur.l3ia2023.mycs.ChartDsl with org.eclipse.xtext.common.Terminals

import "http://www.example.org/chartitproject"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Program returns Program:

        CsvFile=DataFile
        chart+=Chart ( chart+=Chart)*
    ;


Constant returns Constant:
    IntVariable | StringVariable | DoubleVariable;

EString returns ecore::EString:
    STRING | ID;

Chart returns Chart:
    {Chart}
    'Chart'
    '{'
        ('title:' title=EString)?
        ('type:' type=ChartType)?
        ('xAxis:'  xVariable+=ColumnRef (  xVariable+=ColumnRef)*  )?
        ('yAxis:'  yVariable+=ColumnRef (  yVariable+=ColumnRef)*)?
        ('Filter:' Filter+=Condition (  Filter+=Condition)*  )?
    '}';

DataFile returns DataFile:
        ('Load' 'data' 'from' filePath=EString)?
        ('header' hasHeader=EBoolean)?
        ('types' hasType=EBoolean)?
        ('delimiter' separator=EString)?
        'Select' 'Columns' '[' UsedColumn+=Column ( "," UsedColumn+=Column)* ']'
    ;

enum ChartType returns ChartType:
                BAR = 'BAR' | PIE = 'PIE' | SCATTER = 'SCATTER' | LINE = 'LINE' | MIXED = 'MIXED' | BUBBLE = 'BUBBLE' | RADAR = 'RADAR' | AREA = 'AREA' | DONUT = 'DONUT';

ColumnRef returns ColumnRef:
         makeRefTo=[Column|EString]
    ;

Condition returns Condition:
        leftElement=ColumnRef
        (operator=CompOperator)?
        rightElement=Constant
    ;

Column returns Column:
    {Column}
    '('
        ('index' index=EInt)? 'as' name=EString
    ')';

EInt returns ecore::EInt:
    '-'? INT;

enum CompOperator returns CompOperator:
                EQUAL = '==' | GREATER = '>' | LESS = '<' | NOT_EQUAL = '!=' | GREATER_OR_EQUAL = '>=' | LESS_OR_EQUAL = '<=';

IntVariable returns IntVariable:
    {IntVariable}
    'int'
    '('
        (initialValue=EInt)?
    ')';

StringVariable returns StringVariable:
    {StringVariable}
    'str'
    '('
        (initialValue=EString)?
    ')';

DoubleVariable returns DoubleVariable:
    {DoubleVariable}
    'double'
    '('
        ( initialValue=EDouble)?
    ')';

EDouble returns ecore::EDouble:
    '-'? INT? '.' INT (('E'|'e') '-'? INT)?;

EBoolean returns ecore::EBoolean:
    'true' | 'false';
```

FIGURE 2 – La syntaxe concrète

Voici ci-dessous un exemple d'utilisation de notre langage. L'extension de notre langage est « .chart »**.**

```
Load data from "cheminAbsoluVersLeFichierCSV"
header true // pour indiquer si le fichier csv à une entête contenant les noms des colonnes
types true // pour indiquer si le fichier csv contient les types des colonnes
delimiter ";" // indiquer le séparateur du fichier csv
Select Columns [ (index indexDeLaColonneZeroIndex as nomDeLaCleJsonSansEspace), ]

Chart {
title: "Titre du graphe"
type: PIE // Type de graphe (PIE, SCATTER, LINE, BAR etc..)
xAxis: colonneEnAbscisse // sur l'axe horizontale
yAxis: colonneEnOrdonnees // sur l'axe verticale
Filter: carbo > int(200) carbo < int(250) mfr != str("G") mfr != str("K")
}
```

2.3. Description du langage et implémentation pour la génération du html.

Pour générer le **html** à partir du code ci-dessus pour tester le langage, nous avons défini et utiliser les fonctions suivantes dans le fichier xtend :

1. `ArrayList<ArrayList<String>> readCSV(String path, String csvDelimiter, String commonDefault)`
   Dans cette fonction, nous avons lu le fichier csv, ensuite nous avons stocké le contenu dans une liste de liste après le découpage de chaque ligne suivant le délimiteur. Nous avons remplacer les données manquantes par **CommonDefault** par défaut nous avons utilisé **"unknown"**.
   Le but est de pouvoir définir après une politique de gestion des données manqantes pour chaque colonne.
2. `String createJsonData(ArrayList<ArrayList<String>> data,`
   `EList<Column> columns, String delimiter, String commonDefaultMissingValue)`
   Dans cette fonction, nous créons la donnée json à utiliser en fonction des colonnes selectionnées par l'utilisateur. Nous laissons de côté les lignes contenant au moins une donnée manquante en ce qui concerne les colonnes selectionnées. Cette donnée est appelée **data**.
3. `String generateGraphJsScript(Chart chart, ArrayList<String> headers, int index)`
   Dans cette fonction, nous générons le code javascript pour la génération d'un graphe. Cette fonction est appelée autant de fois qu'il y a de graphes.
   Le script chartjs généré prend en compte le titre du graphe et le nom des labels des axes.
4. `def String filteredData(Chart chart, int index)`
   Dans cette fonction nous générons le code javascript basé sur la donnée json créé au début. En effet pour filtrer la donnée, nous avons décidé plutôt de filtrer la donnée json **data** et d'utiliser la donnée filtrée pour la configuration du graphe si ce graphe contient de filtre. Si le graphe spécifique n'a pas de filtre, la donnée utilisée est **data**.
5. Ceci constitue le template de la page html à générer avec les toutes les informations récuprérées.

```
var pageContent  = '''
<!DOCTYPE html>
<head>
<title>«root.name»</title>
<style>
/* Resets */
* {
margin: 0;
padding: 0;
box-sizing: border-box;
}
h1 {
text-align: center;
```

```
line-height: 100px;
margin: 0;
}
«IF charts.size() > 1»
#chartCard {
width:auto;
background: rgba(255, 255, 255, 1);
display: flex;
align-items: center;
justify-content: center;
flex-wrap: wrap;
}

.myChartBox{
width: 800px;
padding:5px 15px;
border: solid 3px rgba(232, 244, 248, 1);
background: white;
}
«ELSE»
#chartCard {
margin: 0px 100px;
background: rgba(255, 255, 255, 1);

}
«ENDIF»
</style>
</head>
<body>
<h1> ChartIT Project </h1>
<section>
<div  id="chartCard">
«FOR id : generateChartIdName(charts)»
<div class="myChartBox">
<canvas id="myChart«id»"></canvas>
</div>
«ENDFOR»
</div>
</section>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<!-- <script src="https://raw.githubusercontent.com/chartjs/Chart.js/master/docs/scripts/utils.j

<script>

// Default json data based on selected column
«jsonData»

// filter data
«FOR i: 0..charts.size() - 1»
«IF charts.get(i).filter.size() > 0 »
«filteredData(charts.get(i), i)»
«ENDIF»
«ENDFOR»



«FOR i: 0..< charts.size()»
«generateGraphJsScript(charts.get(i), headers, i)»
```

```
«ENDFOR»

</script>

</body>
</html>
'''
```

6. Nous avons ajouté un peu de **css** pour gérer la disposition des graphes.

3. Scénari basiques et features

3.1. scénario basique 1.

```
Load data from "/home/enigmatik/eclipse-workspaces/runtime-EclipseApplication/testchartlang/data/cars.csv"
header true
types true
delimiter ";"
Select Columns [ (index 0 as modele), (index 4 as power)]

Chart {
    title: 'Scénario 1'
    type: BAR
    xAxis: modele
    yAxis: power
}
```

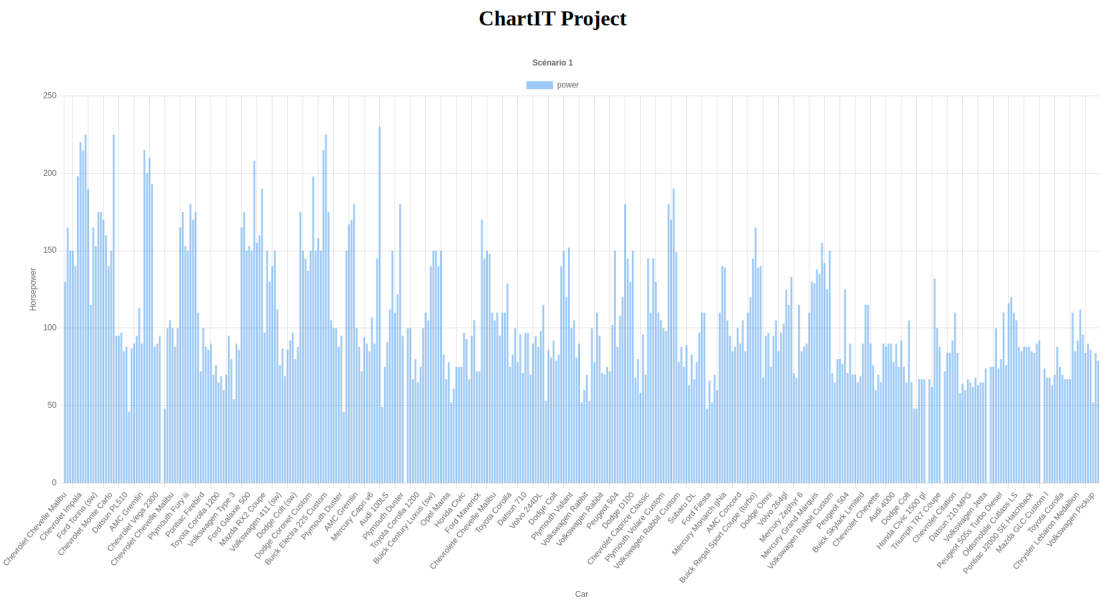

FIGURE 3 – Scénario 1

3.2. scénario basique 2.

```
Load data from "/home/enigmatik/eclipse-workspaces/runtime-EclipseApplication/testchartlang/data/factbook.csv"
header true
types true
delimiter ","
Select Columns [ (index 0 as country), (index 37 as population), (index 41 as cellular)]

Chart {
    title: 'Scénario 2'
    type: LINE
    xAxis: country
    yAxis: population cellular
}
```

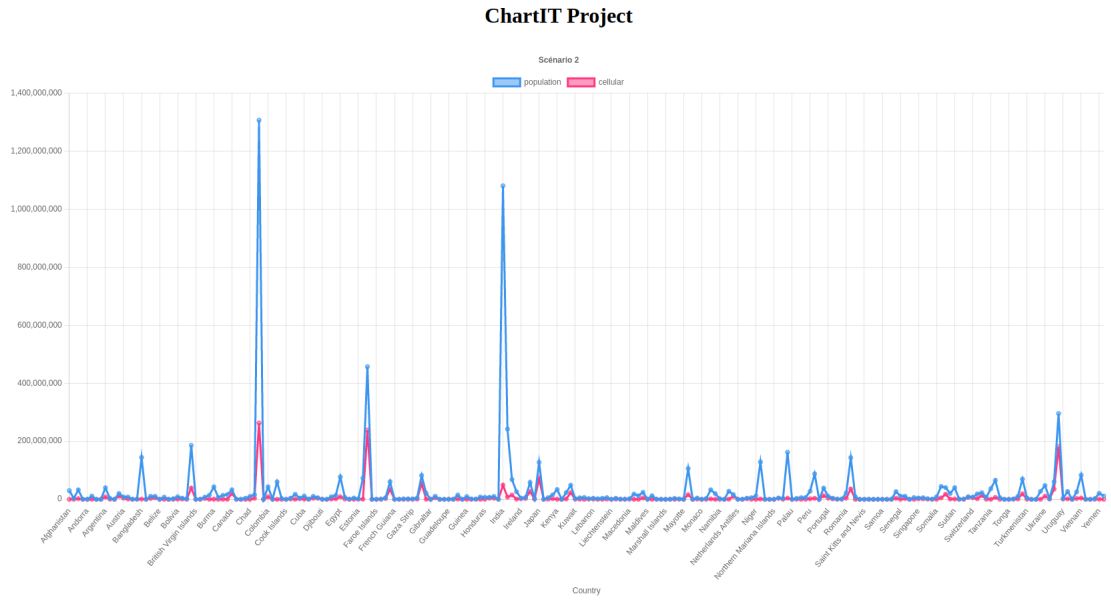

FIGURE 4 – Scénario 2

3.3. Des graphiques avec des filtres sur les données.

```
Load data from "/home/enigmatik/eclipse-workspaces/runtime-EclipseApplication/testchartlang/data/factbook.csv"
header true
types true
delimiter ","
Select Columns [ (index 0 as country),  (index 7 as prodElectricity), (index 6 as electricityConsumption)]

Chart {
    title: "La production d'électricité par pays"
    type: BAR
    xAxis: country
    yAxis: prodElectricity
}

Chart {
    title: "Les pays dont la production d'électricité est inférieure 200.000.000 kwh"
    type: BAR
    xAxis: country
    yAxis: prodElectricity
    Filter: prodElectricity < int(200000000)
}

    Chart {
    title: "La production d'électricité en fonction de la consommation d'électricité"
    type: SCATTER
    xAxis: prodElectricity
    yAxis: electricityConsumption
}


Chart {
    title: 'Nombre de telephone en fonction de la population'
    type: SCATTER
    xAxis: prodElectricity
    yAxis: electricityConsumption
    Filter: prodElectricity < int(200000000) electricityConsumption > int(10000000)
}
```

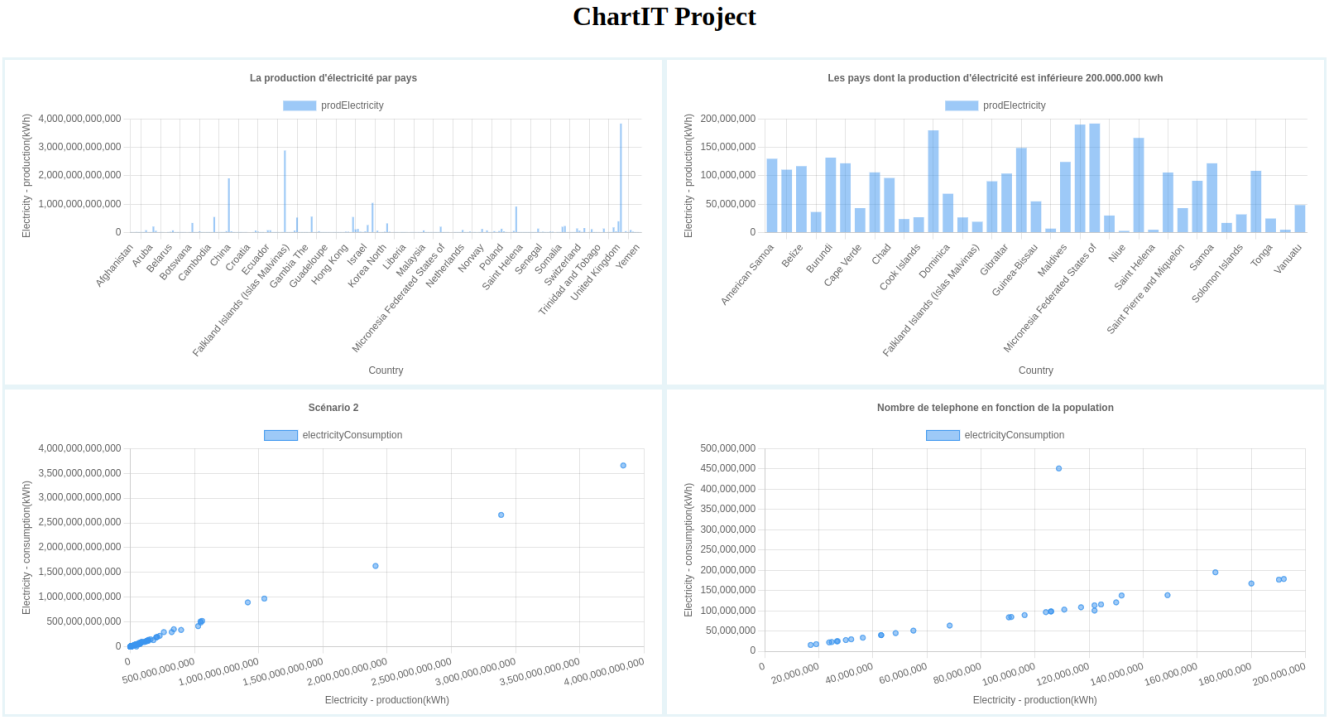

FIGURE 5 – Scénario 3

```
Load data from "/home/enigmatik/eclipse-workspaces/runtime-EclipseApplication/testchartlang/data/cereal.csv"
header true
types true
delimiter ";"
Select Columns [ (index 0 as cereal),  (index 3 as calory), (index 6 as carbo) , (index 1 as mfr)]

Chart {
    title: "Quantité de carbone comprise entre 200 et 250 par céréales avec mfr différent de G et K"
    type: PIE
    xAxis: cereal
    yAxis: carbo
    Filter: carbo > int(200) carbo < int(250) mfr != str("G") mfr != str("K")
}


Chart {
    title: "La quantité de calories par céréale"
    type: BAR
    xAxis: cereal
    yAxis: calory
    Filter: calory > int(110)
}
```

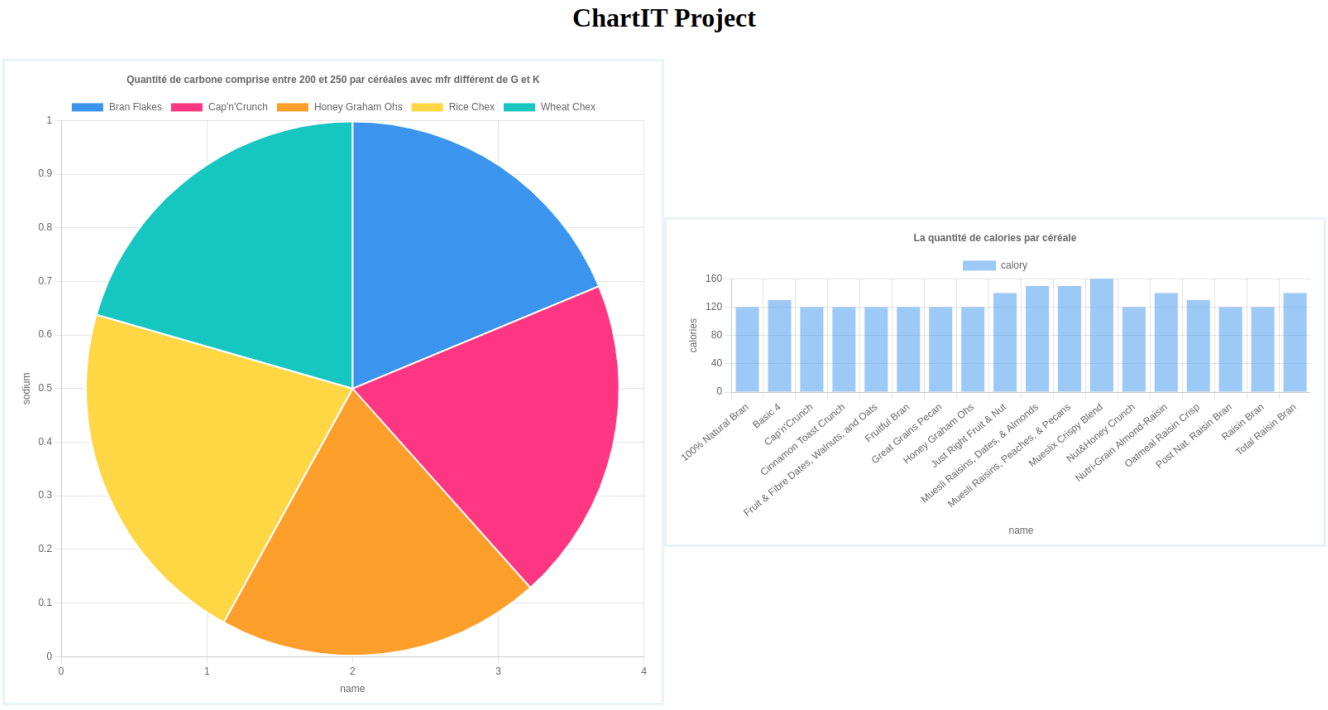

FIGURE 6 – Scénario 4

4. Conclusion

A l'issue de ce projet, nous avons obtenu des résultats plûtot satisfaisants. Même si au début, le projet paraissait plûtot compliquer à réaliser, au fur et à mesure qu'on avançait, le projet est devenue plus compréhensible. Nous avons beaucoup appris depuis la mise en place d'un méta modèle, passant par la syntaxe concrète et la grammaire.