RAPPORT DE PROJET

MÉTHODE DE COMPILATION

« La domotique, c’est fantastique »

Auteurs :

Jean-Baptiste LAUTE

Jean-Michel FAZZARI

# La grammaire

Axiome → t\_BAL\_PROG\_OUV Appareils Interfaces Scenarii Commandes t\_BAL\_PROG\_FER

Appareils → t\_DECLA\_APP\_OUV Details\_appareil t\_DECLA\_APP\_FER

Details\_appareil → Recu\_appareil Details\_appareil

| ε

Recu\_appareil → Un\_Appareil

| Autre\_appareils

| Definir\_appareil

Un\_Appareil → t\_Type Variables\_appareil POINT

Variables\_appareil → t\_ident Suitet\_ident\_app

Suitet\_ident\_app → VIRGULE t\_ident Suitet\_ident\_app

| ε

Autre\_appareils → t\_autre\_appareil PAROUV t\_sous\_Type PARFERM Variables\_appareil POINT

Suite\_Autre\_appareils → VIRGULE t\_ident Suite\_Autre\_appareils

| ε

Definir\_appareil → DEFINIR t\_ident EGAL CROCHETOUV Suite\_ident CROCHETFERM POINT

Suite\_ident → t\_ident Autre\_ident

Autre\_ident → VIRGULE t\_ident Autre\_ident

| ε

Interfaces → t\_DECLA\_INT\_OUV Details\_interface t\_DECLA\_INT\_FER

Details\_interface → Recu\_interfaces Details\_interface

| ε

Recu\_interfaces → t\_Interface Variables\_interface POINT

Variables\_interface → t\_ident SuiteInterface

SuiteInterface → VIRGULE t\_ident SuiteInterface

Scenarii → t\_DECLA\_SCE\_OUV Details\_scenarii t\_DECLA\_SCE\_FER

Details\_scenarii → t\_SCE\_OUV t\_ident PGQ Details\_scenario t\_SCE\_FER t\_ident PGQ Details\_scenarii

| ε

Details\_scenario → Content\_action Details\_scenario

| ε

Content\_action → Condition\_si

| Boucle\_Pourtout

| t\_executer\_scenario t\_ident POINTVIRGULE

| t\_message PAROUV corps\_message Suite\_message PARFERM POINTVIRGULE

| t\_ident POINT t\_action POINTVIRGULE

Condition\_si → SI PAROUV t\_ident POINT ETAT t\_comparateurs t\_etat PARFERM ALORS Details\_scenario Suite\_si

Suite\_si → FSI POINTVIRGULE

| SINON Details\_scenario FSI POINTVIRGULE

Boucle\_Pourtout → POURTOUT t\_ident DEUXPOINTS Type\_Pourtout FAIRE Details\_scenario FAIT POINTVIRGULE

Type\_Pourtout → t\_Type

| t\_ident

| t\_autre\_appareil Affiche\_autre\_appareil

Suite\_message → VIRGULE corps\_message Suite\_message

| ε

corps\_message → t\_texte

| t\_ident

| t\_ident POINT ETAT

Affiche\_autre\_appareil → PAROUV t\_sous\_Type PARFERM

| ε

Commandes → t\_DECLA\_COM\_OUV Details\_commande t\_DECLA\_COM\_FER

Details\_commande →Recu\_commande Details\_commande

| ε

Recu\_commande → ASSOCIER t\_ident EGAL Asso

| PROGRAMMER t\_ident EGAL Prog

Suitet\_ident\_Asso → t\_ident Rec\_Suitet\_ident\_Asso

Prog → Date\_Comm POINT

| CROCHETOUV Suitet\_ident\_Prog CROCHETFERM POINT

Date\_Comm → AROUV t\_date VIRGULE t\_date VIRGULE t\_date VIRGULE t\_date VIRGULE t\_date PARFERM

Suitet\_ident\_Prog → Date\_Comm REC\_Suitet\_ident\_Prog

REC\_Suitet\_ident\_Prog → VIRGULE Date\_Comm REC\_Suitet\_ident\_Prog

| ε

# Nos choix…

Nous avons décidé de tirer le meilleur parti de la programmation Jcup. Pour ce faire, nous avons alliés l’analyse ascendante, et l’analyse descendante. Ainsi, nous avons pu a loisir utiliser les attributs hérités ou attribués afin d’obtenir le meilleur résultat.

### La gestion des erreurs

Première étape

Analyse lexicale

Les mots non reconnus ainsi que les commentaires sont supprimés dans cette première étape.

Seul, les mots clés , les idents et les messages sont captés et vont vers le parser.

Deuxième étape

Analyse syntaxique

Lorsque le parser rencontre une erreur de syntaxe, elle est affiché avec la ligne et le numéro du script de départ.

Troisième étape

Ident\_unique :

Nous avons créé une liste pour collecter les idents à la volée et lorsque l’on en rencontre sur le parcours descendant , une fonction affiche un message d’erreur si l’ident existe déjà.

Format\_Date

Nous nous sommes appuyer sur la méthode de parsing d’une date de java SimpleDateFormat

Si le parsage se fait sur une date erronée, il affiche un message d’erreur

Action 1 et 2

Comme le Result était utilisé pour la quatrième partie (production du code java) nous avons décidé de faire de l’analyse descendante en relevant le type des idents concernés et en comparant ces types avec les actions autorisées.

Pour ce faire, nous avons constitué deux HashMap :

L’un met en relation une clé du type avec un tableau d’action : actionHash(String type, [] String TableauAction), il est initialisé par les actions autorisées.

L’autre met en relation une clé de l’ident avec le type : appareilHash(String ident, String type) et il est alimenté au fil du parcours descendant de la grammaire.

Ainsi, à partir d’un ident rencontré sur lequel on propose une action, on peut rechercher son type avec appareilHash et grâce à son type, on peut obtenir le tableau de ces actions autorisées.On les comparent a l’action proposée. On imprime un message d’erreur lorsque l’action n’est pas trouvé dans ce tableau d’action.

Pour l’action 1, il nous fallait comparer à la liste d’ident préalablement constitué si l’ident était existant

### La génération des fichiers

Afin de générer les fichiers nécessaires au fonctionnement du logiciel de domotique, nous avons pris le choix de créer deux variables texte alimentées tout au long du processus, et de créer et renseigner un fichier qu’une fois le processus terminé. Cela nous permet de ne faire qu’un seul accès disque plutôt que d’ouvrir un fichier en écriture pour la durée de l’analyse.

# ...Et leurs conséquences

Malheureusement, il reste des cas non gérés. Notre plus grande difficulté se trouvait, et se trouve encore dans la gestion des incréments de variables. Nous n’avons pas à ce jour réglé cette problématique. De ce fait, nous pouvons nous retrouver avec des erreurs quand la grammaire reconnaît ses conditionnelles.

Nous nous retrouvons avec un fichier CUP volumineux. Nous avons choisis de détailler notre grammaire un maximum, pour nous laisser le loisir d’être le plus précis dans nos méthodes de développement, mais le revers de la médaille c’est que nous avons un code assez complexe, et difficile à maintenir.

# Répartition des tâches

La grammaire à été réalisée en commun. Nous avons tous les deux critiqué et enrichi nos réflexions afin d’être le plus efficace possible.

La gestion des erreurs à été réalisée par Jean-Baptiste.

La génération du code à été réalisée par Jean-Michel :

«  La génération contient pour moi deux parties bien distinctes. Les scenarii, et le reste. Autant je n’ai pas eu de difficultés à générer les déclarations, les interfaces, etc.. Autant les scenarii m’ont donnés BEAUCOUP de fil à retordre, et notamment la gestion des nom de variables dans les boucle for si elles sont imbriquées dans une conditionnelle. Pour le reste, j’estime n’avoir pas eu trop de mal à le réaliser. »