I Introduction

Le but de ce projet est de faciliter la mise en oeuvre de règles de gestion des ressources permettant l'élasticité dans le Cloud Computing en élaborant un Domain Specific Language à l'aide de XText, et en créant un éditeur de diagramme grâce à Sirius, destinés aux administrateurs du Cloud qui n'ont pas forcément de bagages en terme de développement informatique.

II Concepts

1. Elasticité

L'élasticité dans le Cloud est le fait d'adapter les ressources en fonction de la charge, on peut traduire cela par le nombre de requêtes HTTP sur un serveur, le pourcentage d'utilisation du CPU, de la RAM, du disque, etc...
Aujourd'hui, cela est rendu possible via des machines virtuelles mises à disposition à travers le réseau, comme le fait Amazon par exemple.

1.1 Elasticité matérielle

Celle-ci permet de jouer sur le plan matériel, c'est-à-dire sur la capacité et le nombre de machines virtuelles. Il y a deux formes :

- l'horizontal scaling : il s'agit d'ajouter ou de retirer des machines virtuelles en fonction de la charge.
- le vertical scaling : il s'agit d'ajouter ou de retirer des ressources sur une machine virtuelle existante en fonction de la charge. Par exemple, le CPU ou la RAM.

1.2 Elasticité logicielle

Celle-ci permet d'ajuster les fonctionnalités d'un logiciel en fonction de la charge demandée.
Là aussi, on retrouve deux formes :

- l'horizontal scaling : on va supprimer ou ajouter une fonctionnalité, un service, en fonction de la charge. Cela ne se traduit évidemment pas par une modification du code même de l'application, du service ou de la fonctionnalité fournie, mais simplement par son accès par l'utilisateur.
- le vertical scaling : on va dégrader ou bien "élever" une fonctionnalité, un service, en fonction de la charge. Un exemple : on a un service qui permet de proposer de la publicité sous trois formes qui sont la vidéo, l'image et le texte. Quand la charge est très faible, on va proposer des publicités vidéo, en revanche quand elle est très forte, on va proposer des publicités textuelles.

L'élasticité matérielle est intéressante mais elle peut manquer parfois de granularité et de réactivité : en effet, le temps de lancement d'une machine virtuelle avec tous les services nécessaires n'est pas négligeable, et elle n'est pas très souple car elle ne permet pas d'ajuster précisément les ressources, lancer une deuxième machine virtuelle simplement parce qu'on a dépassé de peu les ressources maximales d'une première machine virtuelle peut poser question. En revanche, elle ne dégrade pas la qualité du service rendu à l'utilisateur, mais elle a un coût en terme d'énergie et d'argent.

En revanche, l'élasticité logicielle présente l'avantage d'être très réactive (temps de réaction de bas niveau) et d'avoir un niveau de granularité petit : en effet l'objectif est de dégrader une fonctionnalité pour la rendre moins "gourmande" ce qui libère des ressources, mais qui présente l'inconvénient de dégrader le service rendu à l'utilisateur.

2. Domain Specific Language

C'est un langage de programmation dédié à un domaine en particulier, pour notre cas, il s'agit de l'élasticité dans le Cloud.

3. Métamodèle

Un métamodèle est un modèle qui va décrire les propriétés d'un autre modèle, il a donc une couche d'abstraction supérieure au modèle.

III Sprint 1

1. Tâches

Pour ce premier Sprint, nous devions :

- créer la grammaire associée au langage Elascript
- monter en compétence sur XText
- monter en compétence sur Sirius
- mettre en place la grammaire sur XText

1.1 Grammaire

Voici la grammaire que nous avons élaborée puis implémentée dans XText :

```
//Rules
Script ::= Statement+
Statement ::= Command | Parallelized
Parallelized ::= SPLIT Body JOIN
Body ::= Statement+ (PARALLELSEPARATOR Statement+)+
Command ::= Function LP ParamList RP SEQUENCIALSEPARATOR
Function ::= LETTER (LETTER | NUMBER)*
ParamList ::= Param ( COMMA Param )*
Param ::= NUMBER+

//Lexems
LP ::= '('
RP ::= ')'
LETTER ::= [a-zA-Z]
NUMBER ::= [0-9]
TIMER ::= 'wait'
SEQUENCIALSEPARATOR ::= ';'
PARALLELSEPARATOR ::= '||'
COMMA ::= ','
SPLIT ::= '['
JOIN ::= ']'
```

1.2 Explications

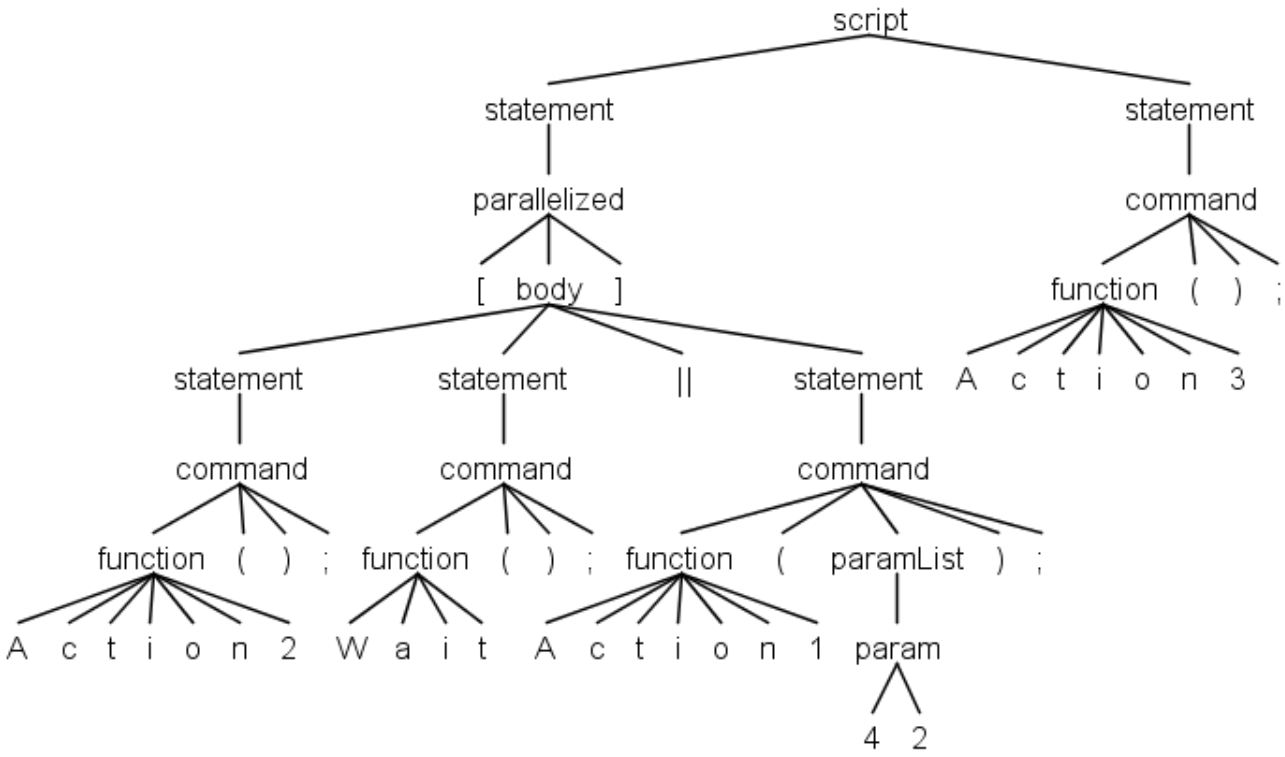
Le DSL est composé un script qui contient une suite de "statement".
Un "statement" peut être une commande, c'est une fonction suivi d'une parenthèse ouvrante suivi d'une liste de paramètres, suivie d'une parenthèse fermante suivi d'un point virgule. Chaque fonction possède au moins une lettre puis autant de chiffres ou de lettres que souhaités, c'est de cette façon qu'on les nomme.
Une liste de paramètres comporte au moins un paramètres, et s'il y en plus d'un, on les sépare par des virgules. Les paramètres sont composés d'au moins un chiffre.

Un “statement” peut aussi être ce que l’on a appelé “parallelized”, c’est ce qui sert à synchroniser des actions. Ils possèdent un crochet ouvrant suivi d’un corps suivi d’un crochet fermant. Un corps est une suite de “statement” (au moins un) suivi d’une liste de “statements” séparés par deux pipes.

1.4 Exemple de code reconnu

```
[
Action2(); Wait();
||
Action1(42);
]
Action3();
```

1.5 Arbre syntaxique associé



2. Difficultés rencontrées

Nous n’avons pas eu de problèmes particuliers pour réaliser la grammaire, en revanche il y a un petit temps d’apprentissage pour XText, et nous n’avons pas encore pris en main complètement Sirius. De plus, il y a très peu de tutoriels pour ces deux outils, mais les cours disponibles sur Campus nous aiguillent.

IV Conclusion

Nous avons terminé la grammaire qui est fonctionnelle, il subsiste un petit problème d’alias à régler sur XText, et nous pourrons ensuite passer à Sirius (nous avons déjà une ébauche).